Introduction to Programming ArcObjects with VBA

(Final)

Jeremiah Lindemann
Lisa Markham
Robert Burke
Janis Davis
Thad Tilton

CONTENTS

1 Introduction

contents

EXERCISE 1: INSTALL THE CLASS DATABASE

In this exercise you will install the database that you will use for certain other exercises in the course.

STEP 1: INSTALL THE CLASS DATABASE

☐ Ask your instructor whether the database has already been installed.

☐ If not, turn to the last page of the exercise book and remove the database CD-ROM from the sleeve attached to the back cover.

☐ Insert the CD-ROM into the CD-ROM drive.

☐ From the *Start* menu, start *Windows Explorer*.

Before running the setup program, you will change the view setting of Windows Explorer.

☐ From the *Tools* menu, click *Folder Options*.

☐ Click the *View* tab.

☐ In the *Advanced settings* area, verify that the *Hide file extensions for known file types* check box is unchecked.

☐ Click *OK*.

Now you will execute the Setup program on the CD-ROM to install the training data.

☐ Navigate through the tree structure to the CD-ROM and click on the CD-ROM drive.

☐ In the contents of the CD-ROM, double-click *Setup.exe*.

☐ On the *Welcome* panel, click *Next*.

☐ Ask your instructor for the destination folder for the database.
Destination Folder: _________________________.

If the destination folder is C:\Student:	If the destination folder is *not* C:\Student:
☐ Click *Typical*.	☐ Click *Custom*.
☐ Click *Next*.	☐ Click *Browse*.
☐ Click *Finish*.	☐ Enter the path to the destination folder.
	☐ Click *OK*.
	Now you are ready to install the class database.
	☐ Click *Next*.
	☐ Click *Finish*.

The database is installed in the destination folder. From now on the destination folder for the installation will be referred to as your working folder.

☐ Remove the CD from the CD-ROM drive and keep it with this book.

EXERCISE END

2

The VBA development environment

contents

EXERCISE 2: EXPLORE THE VISUAL BASIC EDITOR

This exercise will familiarize you with the ArcInfo Desktop development environment. You will begin by exploring the Visual Basic Editor, then customize ArcMap by creating some new interface controls and simple Visual Basic procedures. In the final steps, you will save and test your customizations.

EXERCISE SHORTCUT

1. Open ArcMap and navigate to the Visual Basic Editor. Explore the available toolbars and the Customize Dialog Box within the Editor. Examine the modules found in the Project Explorer.

2. Using the Customize Dialog Box in ArcMap, create a UIButton named *MyFirstControl*. Drag the button onto the ArcMap interface and view its source code.

3. When the control is clicked, write code to display three message boxes. The first will show today's date with an information icon. The second will report the current time in the title of the message box and the third will have the user respond to a yes/no question (use your imagination).

4. Create a standard module and write a private sub procedure, named *TotalPrice* inside the module, that will allow the user to type in a dollar amount into a InputBox. Display the total price (including sales tax) in a message box. This procedure will only require a single line of code. This one line, however, will consist of several nested Visual Basic functions.

5. Using the Customize Dialog Box, add to the ArcMap interface a new macro command that runs the procedure created in Step 4.

6. Within the *MyFirstControl* click event, comment out the code using the Comment Block button. Write code to call the *TotalPrice* procedure from the controls click event.

7. Save the current map as a map template, and create a new map based on the template. This will illustrate that the code will be saved in the template but the controls on the ArcMap interface will still point to the project.

8. Export your standard module.

STEP 1: START ARCMAP, EXPLORE THE VISUAL BASIC EDITOR

You will begin by creating a new map.

☐ Start *ArcMap*.

☐ If the startup dialog appears, choose to open a *new map* (empty map).

☐ When the new map opens, click the *Tools* > *Macros* > *Visual Basic Editor*.

The Visual Basic Editor is where you will write all code associated with the current map, the normal template, or a base template. ArcMap is one of many applications that uses the Visual Basic for Applications (VBA) development environment. Once you are familiar with using this environment in one application, you will find it basically the same in any other application that uses it.

☐ Right-click on any gray portion near the top of the *Visual Basic Editor* interface. Notice that you can add and remove toolbars on the Editor's interface by checking them on from this context menu.

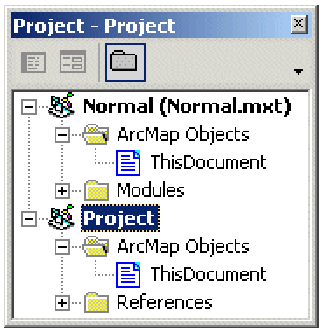
☐ Bring in and examine all the available toolbars.You will learn more about the functionality contained on these toolbars throughout the course.

☐ Right-click on the Editor interface and click *Customize*. A customize dialog box similar to the one used by ArcMap should appear. As with the ArcMap interface, you can customize the Visual Basic Editor by adding or removing controls or by changing their properties.

☐ Close the customize dialog box.

As with any other customization, Visual Basic code in ArcMap must be stored at one of three different levels. Globally available code is stored in the Normal Template (Normal.mxt). Code available only to a particular map is stored in the map document (*.mxd). Optionally, you may choose to create a map from a predefined base template (*.mxt) and store code there. Base templates are designed to give you a head start in creating maps of a particular type. For example, your organization might work with zoning maps, endangered species habitat maps, and a variety of others. To give a standard look to each type of map, you might choose to create a template for each type. In your habitat map template, for example, you might store base layers such as elevation contours, a default north arrow and scale bar style, as well as customized commands and macros to work with this data.

☐ From the Visual Basic Editor *View* menu, choose *Project Explorer*. If it was not already visible, the Project Explorer window should now appear.



Notice the two top-level listings in the Project Explorer, Normal and Project. Because you did not produce your new map from a predefined map template, these are the only two places where you can store customizations.

☐ Expand the listing for *Normal* (click the '+' next to it), then expand the listing for ArcMap Objects.

☐ Double-click the *ThisDocument* code module. Any code that you write here will be stored in the normal.mxt template and will be available to all maps that you create or open.

☐ Expand the listing for *Project*, then expand the listing for *ArcMap Objects*. Double-click to open the *ThisDocument* module. Code written here will only be available for this particular map document.

Close both code modules. You will return to these modules later.

Step 2: Add a new control to the ArcMap interface

In this step, you will use the Customize dialog box to create a new control on the ArcMap interface. You will later return to the Visual Basic Editor to write code for your control.

☐ Close the *Visual Basic Editor*.

☐ Right-click on any gray portion of the *ArcMap* interface and choose *Customize* from the context menu that appears.

☐ In the *Customize* dialog, click on the *Commands* tab. On the left side of the dialog, you should see a list of ArcMap command Categories. Scroll down and highlight the *UIControls* category.

If you had created UIControls previously, you would see them listed on the right side of the dialog and would be able to drag them to the interface from here. Instead, you will create a new UIControl.

☐ Make sure the *Save In* pulldown menu in the lower-left of the dialog is set to *Untitled* (the name of your current map at this point). Click the *New UIControl* button.

☐ There are four types of UIControls that you can create: Button, Tool, EditBox, and ComboBox. Choose *UIButtonControl*, then press *Create*.

A new listing should appear in the commands portion of the dialog called Project.UIButtonControl1; this indicates that your new control is saved in the current map (project) and is the first button control created in the project.

☐ Click on *Project.UIButtonControl1* in the commands list and change the name to **`MyFirstControl`**. You may need to wait a second and click it again in order to change the name.

Notice that the name reverts back to Project.MyFirstControl so you can tell where the customization is being stored.

☐ Drag the new control from the *Customize* dialog box onto any toolbar.

☐ Right-click on the control and choose *View Source* from the context menu that appears. The *Visual Basic Editor* will open, and you will be dropped into the click event procedure for MyFirstControl.

Notice that the wrapper lines for the click event procedure were created for you, and any code that you write inside these lines will execute when the button is clicked.

Question 1: Which code module contains the code for this control? ___________
Why?____
Hint: The control is named *Project.MyFirstControl*.

STEP 3: EXPLORE VISUAL BASIC FUNCTIONS

In this step, you will write code for your new UIButtonControl, then write a standalone procedure (macro). First you will look at using a Message Box, a predefined Visual Basic function useful for a quick return of information.

☐ In the click event procedure for the UIButtonControl you just created, type in the following code:

```
Msgbox "Hello"
```

☐ Return to the *ArcMap* interface and click the *UIButtonControl* you created (MyFirstControl). You should see a message box appear saying "Hello" to you.

☐ Now change the message box to display the date as below:

```
Msgbox Date
```

Visual Basic has several functions such as "Date" that you can use to perform tasks such as formatting data values, returning system information, or obtaining user input. These functions are all well documented in the Visual Basic Help

☐ From the *Help* menu in the Visual Basic Editor, choose *Microsoft Visual Basic Help*.

☐ When the *Visual Basic Reference* appears, click the *Contents* tab.

☐ Expand the heading for *Visual Basic Language Referenc*e.

Hint: Double-Click, or click the "+".

☐ Expand the heading for *Functions*. Visual Basic software's functions are listed here alphabetically.

Question 2: Navigate to the Help page for the MsgBox Function. What is the one required argument for the MsgBox Function? ______________________________

Question 3: How is the optional *buttons* argument specified? ________________
____

You will now use the MsgBox function again, this time to get a *yes* or *no* response from the user.

MsgBox has only one required argument, a message (prompt) to display. The optional arguments can be used to create different styles of message boxes or to provide a message box title or an associated Help file. The *buttons* argument is used to specify different icons, buttons, or both to appear on the message box. This argument can be satisfied with an integer or with the proper combination of Visual Basic constants. In other words, a message box with a question icon and with yes and no buttons could be created with either of the following lines of code:

```
MsgBox "Continue?", 36
MsgBox "Continue?", vbYesNo + vbQuestion
```

By looking at the Help, you will see that the MsgBox function will not return *True* when yes is clicked or *False* when no is clicked. Instead, one of a predefined set of constants may be returned. For a list of these constants, return to the Help documentation for the MsgBox function. Return values can be referred to literally (e.g., 7) or by using the corresponding Visual Basic constants (e.g., vbYes).

☐ Close the *Visual Basic Reference*.

☐ Delete all the message boxes you just created in the click event procedure of the UIButtonControl.

☐ Use your knowledge of Visual Basic functions to write code to do the following when the control is clicked: (1) show today's date in a message box, this time with an *information* icon, (2) report the current time in another message box with a title that reads Current Time, then (3) have the user respond to a yes/no question (use your imagination).

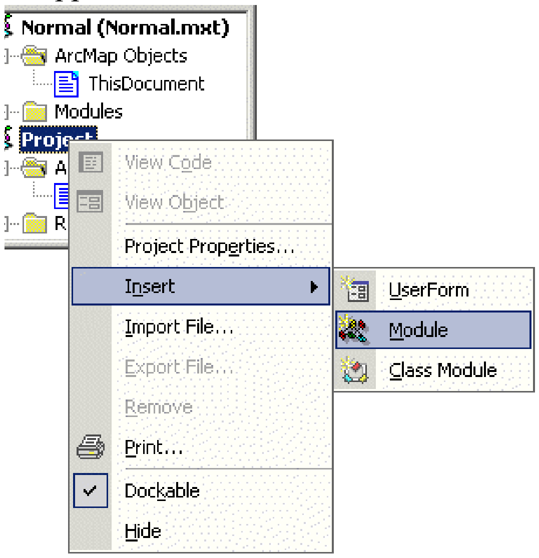
Hint: Use your lecture notes and the Visual Basic Help for predefined functions.

☐ Test your code by bringing ArcMap to the front of your display and clicking the button. Does it work?

STEP 4: WRITE CODE IN PROJECT MODULES

Now you will produce a new procedure and write code that is not associated with a command or event of any kind. Your new procedure will allow the user to type in a dollar amount, then have the total price (including sales tax) displayed. This procedure will only require a single line of code. This one line, however, will consist of several nested Visual Basic functions.

☐ In the *Project Explorer*, right-click on the *Project* heading. Choose *Insert* > *Module* from the context menu that appears.



A new (empty) code module is added to the project; by default, it is named Module1.

☐ In the new module, type the following code:

```
Private Sub TotalPrice
```

☐ Press **Enter**. Notice the rest of the procedure definition is added for you. You have just defined a sub procedure named TotalPrice that has a private scope (it can only be called from within this code module).

☐ Begin the following MsgBox statement to report the total price: **`MsgBox "Your total price is " &`**

☐ Now, place the InputBox function at the end of your MsgBox statement. This will pop up an Inputbox and allow the user to enter a purchase price. Consult the Visual Basic Help for function syntax if necessary.

> NOTE: Remember, in order to get a return value from a function, you must surround your parameter list with parentheses.

```
MsgBox "Your total price is " & InputBox("Price:")
```

☐ Test your code. Make sure your cursor is inside the TotalPrice procedure, then click the *Run Macro* button (the one with the triangle icon).

A simple input box should appear.

☐ Type in a value and press *OK*.

A message box should report the total price as what you just entered.

☐ Next, you will add code to calculate the total price by taking the user input and multiplying it by the appropriate sales tax. Use a formula like the following, which works for California sales tax:

```
MsgBox "Your total price is " & InputBox("Price:") * 1.075
```

☐ Test your code. Does it calculate the proper amount?

☐ What happens if you click *Cancel* on the input box? What if you enter a string instead of a number?

You cannot always assume that your user will enter an appropriate value. Techniques for checking input will be described later.

STEP 5: RUN MACROS FROM THE ARCMAP INTERFACE

As a developer, running your procedures from the Visual Basic Editor is a good way to test your code. You would not, however, expect an end user to interact with your code in this way. In this step, you will experiment with additional ways of running a procedure that is not associated with a control (macro).

☐ In *ArcMap*, click the *Tools* > *Macros* > *Macros*.

From the Macros dialog that appears, a user could run any public procedure available to the project.

☐ Choose *<All Standard Projects>* from the *Macros in:* the pulldown list to view all available macros.

Question 4: Why is the TotalPrice procedure not listed here? ________________
____

☐ Close the macros dialog and return to the *Visual Basic Editor*.

☐ Open the Module1 code module under *Project* and navigate to the TotalPrice procedure. Change the procedure definition from Private to *Public*.

☐ Open the *Macros* dialog again. TotalPrice should now be listed here. Select this procedure and click Run to execute it.

This is an easier way for a user to run a macro. There is, however, a much easier way to access macros that need to be executed often from the ArcMap or ArcCatalog interface. Remember to define macros as public if you want them to be widely accessible.

☐ Open the *Customize* dialog box. Click the *Commands* tab.

☐ In the *Categories* list, locate and select the category called *Macros*. All available macros in the project will be listed on the right side of the dialog.

☐ Select the *TotalPrice* macro and drag it onto a toolbar. A new command is added to the interface that will run this macro.

☐ Right-click the new interface command.

☐ Choose *Change button image*, then select an icon from the context menu that appears.

☐ Close the *Customize* dialog box.

☐ Click the command to run your macro.

STEP 6: RUN A PROCEDURE WITH CODE

Sometimes, you will need to execute procedures using code. Remember, a procedure may be called from any other procedure in the same module, regardless of the scope (public or private). A procedure can be called from outside the module only if it has been defined as public.

☐ Go to the Visual Basic Editor and open the *ThisDocument* module under *Project*.

☐ At the top of the module, pulldown the object list (left) and choose *MyFirstControl*.

☐ In the procedure list (right), choose *Click*. Your cursor should now be in the MyFirstControl_Click event procedure. For modules with several procedures, you can use these pulldown lists to quickly navigate to a particular piece of code.

Instead of running the current code, you will call your TotalPrice macro.

☐ Highlight (select) all the code between the wrapper lines (select everything except the procedure definition and the End Sub lines).

☐ On the *Edit* toolbar, click the *Comment Block* button Comment Block.

A *comment* is a line of code that is ignored by Visual Basic. To prevent a line of code from being interpreted, precede the line with a single quote ('). By default, commented lines of code will appear green in the Visual Basic Editor.

☐ Add the following line of code to execute the TotalPrice procedure:

```
Call TotalPrice
```

☐ Bring *ArcMap* to the front of your display.

☐ Click your *MyFirstControl* button to run the TotalPrice procedure.

☐ Click your macro control.

Notice that both controls are referencing the same code.

You are able to call the TotalPrice procedure in Module1 from the ThisDocument module because TotalPrice is defined as a public procedure. Next, you will change the procedure scope to verify this.

☐ In the *Visual Basic Editor*, open *Module1* and navigate to the *TotalPrice* procedure. Replace the Public keyword with Private.

☐ Return to ArcMap and click the controls again. Visual Basic should return an error stating the macro could not be found.

☐ Open the *Customize* dialog and remove the macro from the interface, leaving the UIControl.

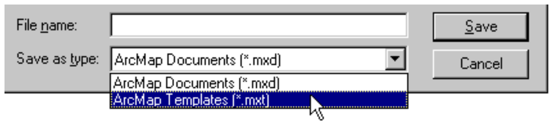
☐ Close the *Customize* dialog box.

STEP 7: SAVE YOUR CUSTOMIZATIONS AS A TEMPLATE, CREATE A NEW MAP

Map templates can be used in ArcMap to define a standard map layout (map layers, size and position of map elements, default north arrow and scale bar styles, graphics and text, etc.). New maps that are created from a template will *inherit* all elements defined in the template. In addition to map elements, a template can also contain customizations such as new controls, procedures, or interface modifications. In this step, you will save your current map as a map template, then create a new map based on the template.

☐ In ArcMap, from the *File* menu, choose *Save As*.

☐ In the file navigation dialog that appears, choose *ArcMap Templates (*.mxt)* from the *Save as type:* pulldown list.



☐ Save the template to *C:\Student\IPAO\Maps* and name it **`NewTemplate.mxt`**.

If you wanted, you could save your map as both a map template (*.mxt) *and* as a map document (*.mxd).

Next, you will open a new map that is *not* based on your template to verify that your customizations are not present.

☐ Click the *New Map File* button on the ArcMap interface.

ArcMap opens a new map document without asking you to specify an intermediate template. Are your custom controls on the interface? If they are, you probably added the customizations (Step 4) to the normal.mxt template instead of the project.

This time, create a map based on your template.

☐ From the *File* menu, choose *New*.

A dialog should appear that shows you a list of available templates that come with ArcMap. If you did not want to use an intermediate template, you could simply choose Normal.mxt.

☐ Click the *Browse* button on the dialog and select your *NewTemplate.mxt* template in your student directory. The new map that opens should have the customizations you saved into the template.

☐ Start the *Visual Basic Editor*. Check the Project Explorer to see how your code is saved for this map.

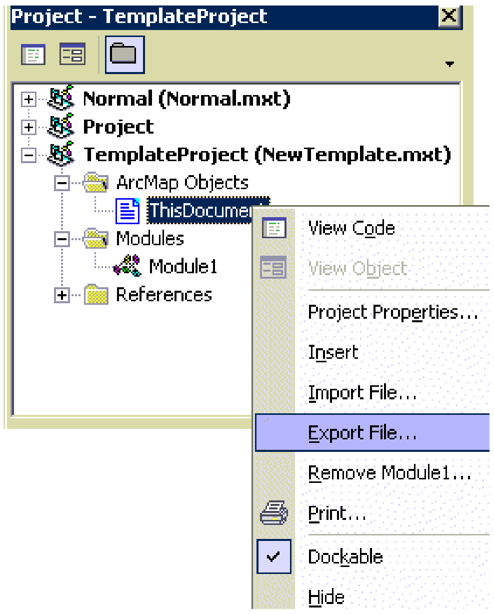
> NOTE: If you try and run the macro from the command on the interface, it is currently referencing Project.Module1.TotalPrice (hover your mouse over the button to view this). This is because you originally created the macro command within your project. Therefore your command will no longer work, because it is referencing the project, and not the template where your code is stored. You can recreate the command using the Customize Dialog Box, and have it point to the appropriate macro, in your template, in order for it to work.

STEP 8: EXPORT CODE

As you have learned, code can be saved in a map document or template. You also have the option of exporting your code to separate files on disk. When you need to use the code in another map, you can simply import the required code modules.

☐ In the *Visual Basic Project Explorer*, expand the *TemplateProject* listing and locate and select *Module1*.

☐ Right-click and choose *Export File* .... Notice that you can also import a file into the project from this context menu.



☐ In the file navigation dialog that appears, save your module in your personal directory.

☐ If you are not continuing to the challenge step below, save your map and exit ArcMap.

In addition to saving your map (.mxd), it is a good idea to also export your code modules, particularly if you plan on using the code in other maps. Exporting your code modules also protects your investment of development time; if a map is corrupted or deleted, your code might otherwise be lost.

CHALLENGE: FORMAT OUTPUT

☐ Use the Visual Basic Help to find functions for formatting text and numbers.

☐ Use one of these functions to format the dollar amount displayed in the TotalPrice message box as currency (e.g., $45.50 instead of 45.50444).

EXERCISE END

ANSWERS TO EXERCISE 2 QUESTIONS

Question 1: Which code module contains the code for this control? Why?

Answer: The ThisDocument code module contains the code for the control that you created. The ThisDocument code module will contain all code associated with the UIControls that you create. It is the standard module that will be found every time you open ArcMap or ArcCatalog.

Question 2: Navigate to the Help page for the MsgBox Function. What is the one required argument for the MsgBox Function?

Answer: A message (prompt).

Question 3: How is the optional buttons argument specified?

Answer: Optional buttons are always specified in brackets [ ]. In this case the *buttons* argument is asking for a numeric value.

Question 4: Why is the TotalPrice procedure not listed here?

Answer: The TotalPrice procedure was declared with the Private keyword, which you change to Public to make it accessible. Using the Private keyword only allows the procedure to be accessed from within the module it is contained. Refer to lecture page 2-30 for a discussion on procedure scope.

QUIZ

☐ Describe two ways to add deleted controls back to a toolbar. ____________________
____
____

☐ To drag a control off a toolbar, the Customize dialog box should be closed (True/False).

☐ A toolbar may contain Commands, UIControls, and Macros (True/False).

☐ New toolbars are created with the:

- Add Toolbar button
- Customize dialog box
- Window menu
- Insert > Toolbar.

☐ List three places in which ArcMap customizations may be stored. _______________
____
____
____

☐ List the four UIControl types. ____
____
____
____
____

☐ If Normal.mxt adds a control, the current map document may remove it (True/False).

☐ Base templates are named with a .mxd extension, for example, MyTemplate.mxd (True/False).

SOLUTIONS

```
'Project - ThisDocument module
Private Sub MyFirstControl_Click()
'  MsgBox "Today is " & Date, vbInformation
'  MsgBox "Time is " & Time, , "Current Time"
'  MsgBox "Are you happy to be here?", vbYesNo
   Call TotalPrice
End Sub

'Module1
Public Sub TotalPrice()
   MsgBox "Your total price is " & InputBox("Price: ") * 1.075
'  **Challenge ...
'  MsgBox "Your total price is " & FormatCurrency(InputBox("Price: ") * 1.075, 2)
End Sub
```

QUIZ SOLUTIONS

1. *Describe two ways to add deleted controls back to a toolbar.* Controls may be added manually by dragging and dropping them from the customize dialog box, or by selecting the toolbar in the customize dialog (Toolbars tab) and clicking Reset.
2. *To drag a control off a toolbar, the Customize dialog box should be closed (True/False)*. False. When the customize dialog box is open, the user interface is in *design mode*.
3. *A toolbar may contain Commands, UIControls, and Macros (True/ False*). True. Toolbars can contain all of these. Additionally, they might contain menus or custom controls.
4. *New toolbars are created with the:* Customize dialog box. On the toolbars tab of the dialog, the New Toolbar button can be used to make a new toolbar.
5. *List three places in which ArcMap customizations may be stored.* ArcMap customizations (user interface modifications and code) may be stored in the normal template (Normal.mxt) for global customization, in a base template (*.mxt) for customizations applied to a group of maps, or in the current map document (*.mxd) for local customizations.
6. *List the four UIControl types.* UIButtonControl, UIToolControl, UIEditBoxControl, and UIComboBoxControl.
7. *If Normal.mxt adds a control, the current map document may remove it (True/Fals*e). True. Customizations in the current map (*.mxd) will take precedence over customizations stored in the normal template.
8. *Base templates are named with a .mxd extension; for example, MyTemplate.mxd (True/False)*. False. ArcMap templates are always named with an mxt extension; map documents are named with mxd.

3

Visual Basic code: How, where, and when?

contents

EXERCISE 3: CREATE A USER FORM

In this exercise, you will create a simple form that converts numbers representing degrees, minutes, and seconds into their decimal degrees equivalent. After creating the form, you will save the ArcMap document and also export the form to its own file (*.frm). A form file contains the form, all of its controls, and the associated code. Forms can be imported into other ArcGIS Desktop applications or even edited using (stand-alone) Visual Basic. This gives you the ability to easily share forms with others.

EXERCISE SHORTCUT

1. Open ex03.mxd and create a UserForm. Add controls to the form to allow the user to enter numbers representing degrees, minutes, and seconds.
2. While your form is in design mode, set initial form and controls properties, such as the name, caption, and font. Assign the UserForm ShowModal property to false.
3. Add code to the cmdApply_event procedure to set the text property of txtDD equal to the degrees, minutes, and seconds value found in the text boxes, that has been converted to decimal degrees. Code the UserForm Double-Click event with a message box that displays today's date and your name. In the Userform Initialize event, display a message box informing the user that the form converts DMS vales into decimal degrees.
4. Add comments to the UserForm, such as the form's purpose.
5. Save the MxDocument and export your code.
6. Run and test the code. Use the ArcMap Identify tool to compare your form's computed decimal degrees with values in the attribute table.
7. Create a sub procedure in the project's ThisDocument module that displays the name of the application in a message box. Use an InputBox to allow the user to change the caption of the application.

STEP 1: CREATE AND DESIGN A NEW FORM

In this step, you will create a form that allows the user to enter numbers representing degrees, minutes, and seconds. These three values will be collected and passed into a mathematical equation for conversion into the equivalent decimal degree value.

☐ Start *ArcMap*.

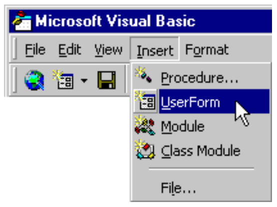
☐ Navigate to **`C:\student\ipao\maps`** and open **`ex03.mxd`**.

☐ Start the *Visual Basic Editor*.

☐ In the *Project Explorer*, click on the current project (*ex03.mxd*) to make it active.

You will add a new form module to the current project (as opposed to the normal template). This form will only be available to this map (ex03.mxd).

☐ On the *Insert* menu, choose *UserForm*.



A new form should appear in a Forms folder in the current project; by default, the form will be called UserForm1. This form module consists of two components: the form designer, where you will arrange form controls, and a form code module, where you will provide code for the form and all of its controls. You can switch between the form designer and the form code by choosing Code or Object from the View menu.

☐ Make sure *UserForm1* is open and you are working with the form designer.

> **NOTE:** If your form is not visible, expand the Forms folder in the Project Explorer, then double-click UserForm1 to open it.

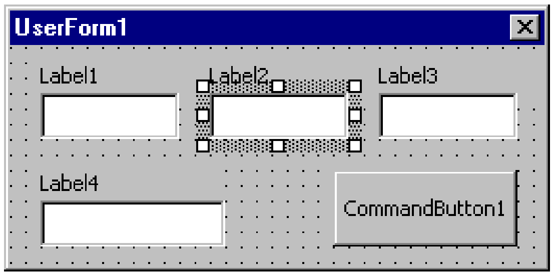
Next, you will add several controls to the form. After adding the controls, you will arrange them and set properties for each control.

☐ Select the *CommandButton* control in the *Toolbox*.

☐ Click to place a new Command button on the form.

☐ Using the *Toolbox*, add four label controls and four text box controls to the form.

☐ Reposition the controls and resize the form to appear as shown below.



STEP 2: SET INITIAL FORM AND CONTROL PROPERTIES

In this step, you will set the initial properties for the form and all of its controls. To access these properties at design time, select a control (or the form itself) in the form designer, then view or modify properties in the Properties window. If the Properties window is not visible, you may launch it from the Visual Basic Editor's View menu.

☐ Select the form by clicking on the *UserForm1* title bar (selection handles should appear around the form).

☐ In the *Properties* window, for the Name property, type `frmDMStoDD`

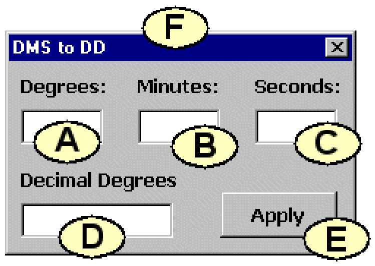
☐ For the *Caption* property, type `DMS to DD`. Notice this caption now appears on the form's title bar.

☐ Set the *ShowModal* Property to *False*.

> NOTE: A *modal* form (ShowModal = true) is one that a user must dismiss before proceeding to other forms or returning to the application. *Modeless* forms (ShowModal = false) are those in which the user can move between the form, the application, and any other open forms. Modal forms can be used in situations where leaving the form while it is open may cause an error (e.g., not specifying a required parameter).

Next, you will set the initial properties for each control on the form. Note that you will not be concerned with all the controls properties because some have no bearing on this project. For example, the default name will be used for labels (e.g., Label1) simply because you will never need to reference these controls in your code. Controls that are referenced will be given a more intuitive name (e.g., cmdApply).

☐ Use the graphic below to assign the appropriate name and caption properties for the controls.



Control Names
- A) txtDegrees
- B) txtMinutes
- C) txtSeconds
- D) txtDD
- E) cmdApply
- F) frmDMStoDD

Next, you will select all nine controls and set their font property simultaneously.

☐ Select all nine controls by holding down `Shift` and clicking on each one of the controls. (Or draw a box around all nine controls to select all the controls at one time.)

☐ In the *Properties* window, double-click the *Font* property. The Font dialog will appear.

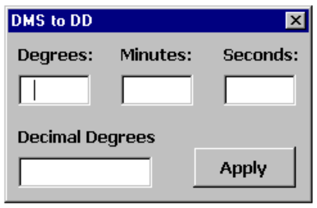
☐ For *Font size*, choose *10*.

☐ For *Font style*, choose *Bold*.

☐ Click *OK* to apply this font style to all the selected controls.

> NOTE: Any new control's font will default to the form's font property. To save time, you may find it useful to change the form's font style *before* adding controls.

When complete, your form should look similar to the one shown below.



STEP 3: WRITE CODE FOR THE APPROPRIATE FORM AND CONTROL EVENTS

In this step, you will associate code with the Click event of cmdApply.

☐ Double-click *cmdApply* on the form designer.

The Code Editor window appears. The cursor appears in the Click event for cmdApply. This code will execute when the user clicks the Apply button at run time.

> NOTE: It is a good idea to indent your code for readability.

☐ Within the *cmdApply_Click event*, add the following code:

```
txtDD.Text = (txtDegrees.Text) + (txtMinutes.Text / 60) + (txtSeconds.Text / 3600)
```

This code sets the text property of txtDD equal to the degrees, minutes, and seconds value (in the text boxes) converted to decimal degrees.

Experiment with the Object and Procedure boxes at the top of the Code Editor window. Remember that these pulldown menus are another method of specifying the object/event combination you want to code.

☐ Explore the various controls and the events to which they can respond.

☐ Challenge: Write code to make the computer beep when text is entered in the txtDegrees text box.

Next, code the form's Double-Click and Initialize event procedures to display some simple message boxes.

☐ Code the UserForm *Double-Click* event with the following code:

```
MsgBox "This program was created on <Today's Date> by <Your Name>"
```

☐ Code the UserForm *Initialize* event with the following code:

```
MsgBox "This form converts DMS into Decimal Degrees", vbInformation, "Convert DMS to
DD"
```

STEP 4: ADD COMMENTS TO CLARIFY YOUR CODE

In this step, you will add some comments to document the form.

☐ Move to the top of the form module.

☐ Add the following three comment lines:

```
'Name:
```

```
'Date:
'Program purpose:
```

☐ Enter your name, the date, and the purpose of this module.

☐ Add one more comment inside the cmdApply_Click event procedure from Step 3.

☐ For the comment, briefly describe the formula and how it works.

STEP 5: SAVE YOUR MAP, EXPORT THE FORM

In this step, you will save the project and export the form.

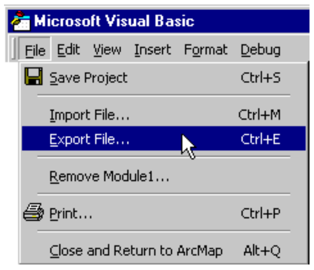
☐ In *ArcMap*, click *File* > *Save As*.

☐ Navigate to *C:\Student* and save the project as `DMS_To_DD.mxd`

☐ Open the *Visual Basic Editor*.

☐ Select (highlight) *frmDMStoDD* in the *Project Explorer*.

☐ From the *File* menu, click *Export File*.



☐ Navigate to *C:\Student* and save the form as `frmDMStoDD.frm`

Your form is now saved in two locations: It is part of the DMS_To_DD.mxd project and is also saved as a form file, frmDMStoDD.frm. You can open the Map (.mxd file) to run the form or can load the form file into another map document (or even a standalone Visual Basic project). Upon exporting a form, the form's design (controls, layout, and properties), as well as all of its associated code, is written to disk.

NOTE: When importing and exporting forms, you will only need to specify a single file (*.frm). In reality, two files are used to define the form, one with a *frm* extension, the other with *frx*. This can be an important consideration when copying form files between folders or e-mailing a form to a friend, for example.

STEP 6: TEST THE FORM

Next, you will test the form.

☐ Click on the form designer or code module to make it *active*.

☐ Click *Run* > *Run Sub/User Form* (or press `F5`) to launch the form. The initialize message box should appear.

☐ Enter only the numbers from the list below of approximate locations into the appropriate text boxes on the form.

City	Degrees	Minutes	Seconds
Paris, France	North 48 East 2	30 12	36 6
Salt Lake City, Utah, USA	North 40 West 111	27 31	1 12
Sydney, Australia	South 33 East 151	32 10	59 12

☐ Click *Apply*.

When you click Apply, the appropriate decimal degree value should display in the Decimal Degrees text box.

☐ Try several of the coordinates.

Because you defined your form as modeless (the form's ShowModal property was set to false), you can move between it and the ArcMap application.

☐ Use the ArcMap *Identify tool* to compare your form's computed decimal degrees with values in the attribute table (values may differ slightly).

Think about other values a user might enter. Some input would not be valid, like -0, -0.04, -33 (northing), South 33, or S33, and entering such values would cause an error. To prevent the form from *bombing out*, your code needs some error checking to ensure valid input. Branching statements will be discussed in the following lesson, which will enable you to check input, then perform different operations depending on the value.

☐ Close the form when finished testing.

Step 7: Work with the Application and ThisDocument variables

In this step, you will work with two predefined variables that are available in ArcMap and ArcCatalog: *Application* and *ThisDocument*. Do not worry if you have not worked with variables before; they will be covered in more detail in the next lesson. For now, all you need to know is that variables in Visual Basic are much like variables you may have used in algebra; they are basically a name that stands for a value.

In this step, you will start by writing code to display the name of the application in a message box.

☐ From the *Project Explorer*, open the map's *ThisDocument* code module.

☐ Add a new public sub procedure called **`AppName`**

☐ Begin a MsgBox statement. For the message, type **`Application`**, then press '**`.`**'. A Code Completion box should display which shows you all properties and methods available for the Application object.

☐ Scroll through the list to answer the following questions as best you can:

Question 1: Which property do you think can be used to access the current map document?____

Question 2: Which method could you use to open another document? __________

Question 3: Which method could be used to keep other users from accessing your code? ____

☐ With the Code Completion list open, type **`N`**. The list should scroll to the Name property.

☐ Press the **`space bar`** to complete the statement.

☐ Bring ArcMap to the front of your display. Choose *Macros* from the *Tools* menu. Locate and run your new macro (AppName) from the dialog. A message box should appear with the simple message *ArcMap*.

Now you will write code that allows the user to change the name displayed on the ArcMap title bar. Remember to assign a new value to an object property; the syntax is: *Object.Property = SomeValue*

☐ Return to the AppName procedure. Comment out the existing MsgBox statement.

☐ Type **`Application`**, then '**.**', then type **`Name`** (or choose it from the Code Completion list).

☐ Type an equal sign (=). Whatever appears on the right of this equal sign will be assigned to the Name property.

☐ Assign the property using an InputBox statement. This will allow the user to type in a new name.

☐ Your finished statement should look similar to the one below:

```
Application.Name = InputBox("New Application Name:")
```

Question 4: Run the procedure. You receive an error. Why? _________________

Some properties are read-only and cannot be changed by assigning a new value. The application name is an example of a read-only property; it will always be ArcMap or ArcCatalog.

You will modify your code to use another property to change the title displayed by the application.

☐ Return to the AppName procedure. *Backspace* over Name. To bring up the Code Completion list again, hold down the Control key (**`Ctrl`**) and press the **`space bar`** simultaneously.

☐ Scroll through the list and locate a property that you think will change the text displayed in the ArcMap title bar.

☐ Bring ArcMap to the front of the display and run *AppName* from the *Macros* dialog. Did it work?

Although you cannot change the application's name, you can change the text that is displayed on the title bar by setting the Caption property.

CHALLENGE: ADD BUTTONS TO OPEN AND CLOSE THE FORM

In this step, you will add a Command button to the form that closes it.

☐ Add a button named **`cmdQuit`** with a caption of *Quit*.

☐ In *cmdQuit*'s click event procedure, type **`UnLoad Me`**

☐ Run the form.

☐ Test the *Quit* button. The form should close when it is clicked.

Next, add a UIButtonControl to the ArcMap interface to launch your form.

☐ Open the *Customize* dialog box.

☐ Add a new *UIButtonControl* and make sure to save it in the current map (DMS_to_DD.mxd).

> NOTE: If you save the control in Normal.mxt, it will not be able to reference a form stored in the current map document.

☐ Open the source code for the new UIButtonControl and navigate to its Click event procedure.

☐ Write a single line of code that opens your DMStoDD form. (Hint: Use one of frmDMStoDD's methods.)

☐ Test the UIButtonControl.

CHALLENGE: USE A MAP EVENT TO LAUNCH THE FORM

In this challenge step, you will write code to open your form using an event associated with the map document.

☐ In the *Visual Basic Editor*, open the map's *ThisDocument* code module.

☐ At the top of the code module, pull down the object list (left) and choose *MxDocument*.

☐ Pull down the procedure list (right). Explore the various events to which the document can respond.

☐ Find the appropriate event procedure to show the form each time the user switches between data and layout view.

Question 5: What is the name of this event? ______________________________

Question 6: What event would you code if you wanted to show the form when the document first opens?____

Question 7: When a new document is created? ____________________________

☐ Write a single line of code using the form's Show method to open the form when the view changes.

☐ *Close* the *Visual Basic Editor*, return to *ArcMap*, and test your code by changing between data and layout view. Does your form appear?

EXERCISE END

ANSWERS TO EXERCISE 3 QUESTIONS

Question 1: Which property do you think can be used to access the current map document?

Answer: Name

Question 2: Which method could you use to open another document?

Answer: NewDocument or OpenDocument

Question 3: Which method could be used to keep other users from accessing your code?

Answer: LockCustomization

Question 4: Run the procedure. You receive an error. Why?

Answer: Name is a read-only property. The name of the application you are currently working with is ArcMap, something you will not be able to change. Read-only properties will be discussed in more detail in Lesson 7.

Question 5: What is the name of this event?

Answer: ActiveViewChanged

Question 6: What event would you code if you wanted to show the form when the document first opens?

Answer: OpenDocument

Question 7: When a new document is created?

Answer: NewDocument

QUIZ

☐ Which of the following is false? Forms and controls:

- Have height and width
- Can change color
- Have a modal setting
- Can be resized
- Have names

☐ All controls have all the same properties (True/False).

☐ By selecting a group of controls, you can globally set one of their shared properties (True/False).

☐ What is the main difference between a text box and a label? When is it appropriate to use each? ____

____

☐ A TextBox control supports which of the following events?

- Click
- Double-click
- Initialize
- Move
- Resize

☐ What is the significance of a control's Name property? How is it different than the Caption property? ____

____

____

☐ Do all controls have a Name property? Yes/No

☐ All properties for any control or form can be set at design time or run time (True/False).

SOLUTIONS

```
'*** frmDMStoDD code module
'Name:
'Date:
'Program purpose:

Private Sub cmdApply_Click()
  txtDD.Text = (txtDegrees.Text) + (txtMinutes.Text / 60) _
               + (txtSeconds.Text / 3600)
End Sub

Private Sub UserForm_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    MsgBox "This program was created on " & Date & " by Thad"
End Sub

'Initialize the DMS text boxes all equal to 0 (zero)
Private Sub UserForm_Initialize()
    MsgBox "This form converts DMS into Decimal Degrees", vbInformation, _
    "Convert DMS to DD"
'**Challenge 1
'ThisDocument code module
'UIButton code to open the form, ThisDocument code module
Private Sub UIButtonControl1_Click()
   frmDMStoDD.Show vbModeless 'Modeless = Can interact with form and ArcMap
End Sub

Private Sub cmdQuit_Click()
   Unload Me
End Sub

'Challenge 2
Private Function MxDocument_ActiveViewChanged() As Boolean
   frmDMStoDD.Show
End Function
```

QUIZ SOLUTION

1. *Which of the following is false? Forms and controls:* Have a modal setting. A form can be modal or modeless. A modal form must be dismissed before a user can interact with any other part of the parent application. Controls do not have a modal property.
2. *All controls have all the same properties (True/False).* False. Many types of controls have some properties in common (the Name and Font properties, for example). Several properties, however, are unique for a type of control.
3. *By selecting a group of controls, you can globally set one of their shared properties (True/False).* True. When a group of controls are selected on the form designer, only their shared properties will appear in the properties dialog. Changing one of these properties will change it for all selected controls.
4. *What is the main difference between a text box and a label? When is it appropriate to use each?* A text box is more appropriate for user input, because it allows new information to be typed in. Labels are best used to display (read-only) information; their caption cannot be changed by the user.
5. *A TextBox control supports which of the following events?* Double-Click is the only event listed here that the TextBox control supports. It supports several other events that are not listed here, such as Change, MouseDown, KeyPress, and Enter.
6. *What is the significance of a control's Name property? How is it different than the Caption property?* A control's name is used to refer to the control at run time (in code). The caption property is the text displayed on the control for the user.
7. *Do all controls have a Name property? Yes/No* Yes. In order to refer to a control at run time, a control must have a name. When a control is initially added to a form, VBA assigns a default name (Label1, for example).
8. *All properties for any control or form can be set at design time or run time (True/False).* False. Some properties can only be set at design time, such as a control's name. Others can only be set at run time, such as a list box's list.

4

Using variables

contents

EXERCISE 4A: WORK WITH VARIABLE SCOPE

In this lesson, you will learn about the scope of variables in Visual Basic programs.

Procedure variables are declared using Dim or Static statements within a procedure. Procedure variables are *recognized* by Visual Basic only inside the procedure where they are declared.

Module variables are declared in a module (i.e., form module) using the Private (same as Dim) statement in the module's General Declarations section. They can then be accessed from any procedure within that module.

Public variables have the widest scope and are declared using the Public statement in the General Declarations section of a module. Public variables are recognized by any form or module in a project. A large part of using variables effectively in Visual Basic is knowing how and when to declare and access variables of these different scopes. Although variables can be declared in the General Declarations section, they must be set within a procedure.

EXERCISE SHORTCUT

1. Run the program named *VariableScope.exe* from ..\IPAO\Forms.
2. Add unique values to the textboxes and run the sub procedure.
3. Assign a unique value to the strAnotherLocal variable. Run the procedure and answer the questions regarding the variable scope of all the dimensioned variables.
4. Determine how public variables from a form module are referenced by other modules, by selecting the appropriate code syntax.
5. Write the code to display the value of a variable that was declared in the General Declarations section of one of the project's standard modules.
6. Examine the code to determine how static variables work, by selecting the statement that declares a variable as static and running the procedure.

STEP 1: RUN A VISUAL BASIC PROGRAM

In this step, you will run the file VariableScope.exe. It is a Visual Basic executable that illustrates variable scope.

☐ Using the *Windows Explorer*, navigate to *...IPAO/Forms* in your student directory.

☐ Double-click on the file *VariableScope.exe.*

The VariableScope program simulates a Visual Basic Code Editor window. The first Code Editor window to display is actually a form named frmVariableScope.

Notice that the first two lines (the General Declarations section) of the module declare a public and private variable.

Notice that Private is used to declare strFormVar. This is a module-level variable.

Notice that Public is used to declare strGlobalVar. This is a public- or project-level variable.

STEP 2: SET VARIABLES OF DIFFERENT SCOPES

In this step, you will set procedure, module, and public variables.

In the first procedure (cmdSetVars_Click), notice the variable strLocalVar is declared with the Dim statement (strLocalVar is a procedure variable).

☐ In the three text boxes, type in your own unique values for each variable: strLocalVar, strFormVar, and strGlobalVar (or keep the existing values).

☐ Click the *Run Sub* button to run this procedure.

Running the procedure sets the three variables above and then reports their values to you in a message box. All three variables should be reported without error.

☐ Click *OK* to close the message box.

☐ Click *Continue* to move to the next piece of code.

STEP 3: CHECK VARIABLE SCOPE

By running the previous procedure, you entered values for the three variables. In this procedure, you will declare and set a new procedure variable and then check the value of all four variables you have worked with so far.

☐ Type in your own unique value for the procedure variable strAnotherLocal.

Review the block of code.

☐ Notice that in this block of code, a message box will pop up to report the value of each of the four variables.

Question 1: Do you think Visual Basic will recognize all four of these variables?

____

☐ Click *Run Sub* to find out.

☐ As each message box appears, review the code where that variable was declared (dimensioned).

☐ Click *OK* to close each of the four message boxes.

The first three values are reported without error.

Question 2: Why does the last line of code in this procedure give you an error?

____

☐ Check to see where strLocalVar is declared.

Question 3: What is the scope of this variable? ___________________________

☐ Click *Continue* to move to the next form.

☐ Do not close either form; you will use both in the following steps.

STEP 4: REFERENCE PUBLIC VARIABLES FROM A FORM MODULE

Now that you have a better idea of how variables of differing scopes are declared, in this step you will examine how public variables from a form module are referenced by other modules.

The next form is called frmScopeContinued and will reference some variables you declared in the previous form (frmVariableScope) and a variable declared in a third standard module (Module1). The first procedure in this *module* is going to display a message box with the value of the public variable declared at the beginning of the last form module (strGlobalVar).

☐ Refer back to the first lines of frmVariableScope. Notice where the variable strGlobalVar is declared.

☐ Read the three MsgBox statements in this first procedure in frmVariableScopeContinued.

☐ Click the one that will properly reference strGlobalVar.

☐ Click the *Run Sub* button. A message box will tell you if your choice was correct. If you did not get it the first time, keep trying.

! When referencing a public variable declared in a form module, you must first reference the form, followed by a dot (.), then the name of the variable.

☐ Click *Continue* to move to the next procedure.

STEP 5: REFERENCE PUBLIC VARIABLES FROM A STANDARD MODULE

This procedure will report the value of a variable that was declared in the General Declarations section of one of the project's standard modules.

☐ Click the *Show Module1* button to view the module.

Module1 declares one public variable, strModGlobal.

☐ Close the Module1 window.

☐ In the cmdViewModuleVar_Click procedure, for the text box, <Module1 Variable>, type in a reference to this variable: **`strModGlobal`**.

☐ In the text box to the right of the equal sign, type in a value for this variable (or keep the default value of *North America*).

☐ Click *Run Sub* to execute the procedure.

A message box will appear, indicating whether you have properly referenced the variable.

NOTE: When referencing public variables that are declared in another form, indicate the name of the form and the variable. In the above case, however, when a public variable is declared in a standard module, only provide the name of the variable (strModGlobal = "North America").

☐ Click *Continue* to move to the final piece of code.

STEP 6: USE STATIC VARIABLES TO INCREMENT A VALUE

When a variable is declared inside a procedure, that variable cannot be referenced anywhere except within that procedure. If a procedure variable is declared with the Dim statement, the value of the variable will last in memory only as long as the procedure is running. When the procedure is run again, these variables are reinitialized.

Sometimes it is necessary to retain a procedure variable's value after a procedure stops running (for example, if you were implementing a counter and wanted to retain the value of a current count until the procedure was called again). After a second, third, and (n)th iteration, you would want to increment the variable and retain the current value between iterations.

The last bit of code will illustrate the use of the Static statement for declaring variables. When a variable is declared as Static, its value is kept in memory even after its procedure has terminated.

☐ In the *cmdIncrement_Click* procedure, click the *Option* button next to the Dim statement.

This Dim statement declares an integer value.

☐ Examine the two remaining lines of code to get an idea of the code's probable intent: to increment the integer value (e.g., counter).

☐ Click *Run Sub* to execute this procedure.

Question 4: What is the result displayed in the adjacent label? ________________

☐ Click *Run Sub* again.

Question 5: Now what is the result displayed? ____________________________

☐ Now click the *Option* button for the Static statement. This declares the same integer variable as static.

☐ Click *Run Sub* and run the procedure three more times.

Question 6: What is the result displayed in the adjacent label? Is this what you would expect? ____

☐ Click the *Option* button for the Dim statement.

☐ Click *Run Sub*.

☐ Click *Finish* when done to exit the application.

EXERCISE END

ANSWERS TO EXERCISE 4A QUESTIONS

Question 1: Do you think Visual Basic will recognize all four of these variables?

Answer: No. strLocalVar is declared in a different procedure, thus it is a procedure-level variable. Procedure-level variables can only be accessed in the procedure in which they are declared.

Question 2: Why does the last line of code in this procedure give you an error?

Answer: The variable strLocalVar is declared in another procedure (see previous answer).

Question 3: What is the scope of this variable?

Answer: Procedure-level.

Question 4: What is the result displayed in the adjacent label?

Answer: 1

Question 5: Now what is the result displayed?

Answer: 1

Question 6: What is the result displayed in the adjacent label? Is this what you would expect?

Answer: 3 Yes, You should expect to see this value because you are now using a Static variable.

EXERCISE 4B: CREATE A GUESSING GAME

In this exercise, you will experiment with variables by creating a guessing game. The application will produce a random number. Use that number to select a feature from the United States layer, and then allow the user to guess the state. If the user cannot guess the state, your application will let them view clues from the feature attribute table.

EXERCISE SHORTCUT

1. Open the ex04.mxd document, found in the Maps directory. Examine the frmGuessState form and code module. Notice how the form has already been designed and some of the code has been written for you.

2. Make sure Option Explicit is set, by examining the Editor tab on the Options dialog.

3. Write code when the UserForm Initializes to select a random number between 0 and 49. Then pass the value to the SelectState procedure that will use it to select a feature from the States layer.

4. Add code to the form's Guess button. The code will ask the user to type in a guess for the country being displayed, then compare the user's guess with the selected feature's Name attribute. If the user is correct, he/she is congratulated and another random state is chosen. If the user is not correct, the number of guesses is incremented and he/she is encouraged to guess again.

5. Code the Hint button to display information in a message box from a randomly selected field in the State layer's attribute table.

STEP 1: OPEN A MAP AND EXISTING FORM

In this step, you will open a map and a form that will eventually allow the user to guess at a selected state in the States layer. The form will randomly choose a number between 0 and 49. Use it to select and zoom into a state, and then allow the user to guess its name. The form will also give hints from the State layer's attribute table.

☐ Start *ArcMap*.

☐ Navigate to *C:\Student\ipao\maps* and open *ex04b.mxd.*

☐ Start the *Visual Basic Editor*.

☐ Navigate to the *Forms* folder under *Project* in the *Project Explorer* window.

☐ Open the form *frmGuessState* by double-clicking on it.

At the moment, the form already has the required controls added to it; however, the controls do not have any corresponding code. You will notice that some code has been written for the form that initializes variables. In the following steps, you will add code for each of the controls to create a functional guessing game.

STEP 2: REQUIRE VARIABLE DECLARATION

Option explicit enforces a variable declaration requirement. With this option enabled, a programmer must declare all variables before they can be used in a program. Without Option Explicit, variables that are not declared default to the Variant data type. Use of variants, though sometimes necessary, is generally poor programming practice and a waste of system memory resources.

☐ In the *Visual Basic Editor*, click *Tools* > *Options*. Then click the *Editor* tab.

☐ Make sure that *Require Variable Declaration* is checked. Click *OK*.

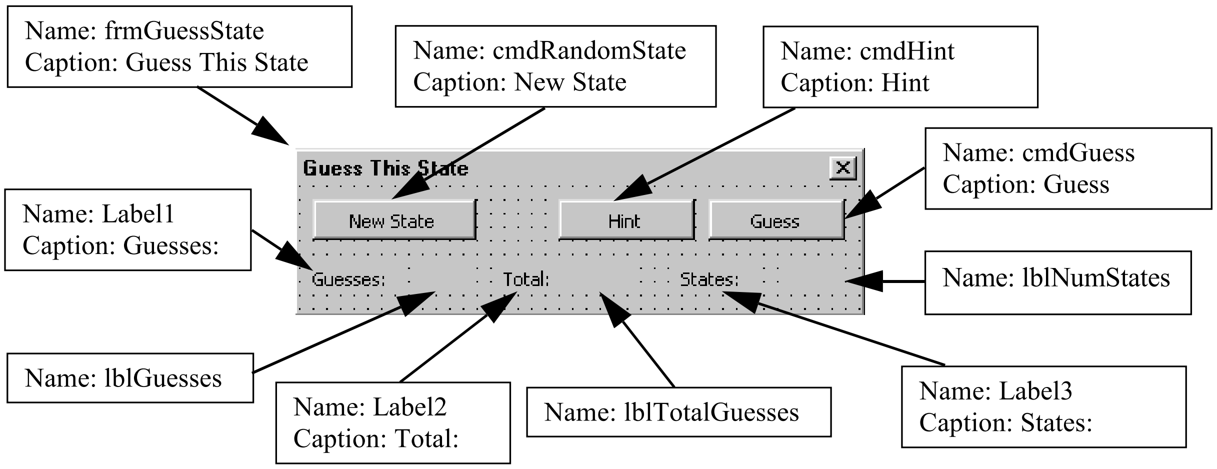
☐ Click *OK*.

STEP 3: WRITE CODE THAT SELECTS A RANDOM FEATURE

Examine controls on the form.

In the form, confirm that the controls and properties are set as shown in the graphic below.

NOTE: Click View > Properties window (or press F4 on your keyboard) to display the properties window..



In this step, you will write the code for the frmGuessState Initialize procedure. When a user first starts your application, this code will execute. The code will select a random number between 0 and 49, then pass the value to a procedure that will use it to select a feature from the States layer.

☐ Double-click *frmGuessState* to view the code module for the form.

☐ Navigate to the *UserForm_Initialize* event procedure.

☐ Notice that the first four lines of this procedure have already been written for you.

This code will simply initialize variables used in the application, such as references to the ArcMap Application and the current document, as well as the States and City layers (the 1st and 2nd layers in the Table of Contents).

At the bottom of this procedure, you will add code to produce a random number.

☐ Declare the following variable to store the random value (using the Dim statement).

```
intRandomNumber As Integer
```

☐ Open the *Visual Basic Help*.

☐ Find and review the topic for the Randomize statement and the Rnd function.

The Randomize statement is used to initialize the random number generator. Each call to Randomize will generate a list of random real numbers inclusively between 0 and 1. The Rnd function is then used to return a random number from this list.

☐ Examine the remarks section of the Rnd function in the online Help. Shown below is a formula using Rnd to create random integers within a given range.

Int((upperbound - lowerbound + 1) * Rnd + lowerbound)

Because the Rnd function can only return a real number between 0 and $0.99\overline{9}$, you will use the above equation to produce a specific range of random values. If Rnd returns a '0', the lowest possible number from Rnd, the equation simplifies to the lowerbound value provided. If '$0.99\overline{9}$' is retrieved, the highest possible number from Rnd, then the expression evaluates to the upperbound provided. All other numbers returned by Rnd will force the expression to evaluate between the upper and lower limits.

☐ Before creating a random number with the Rnd function, add the **`Randomize`** statement to initialize the random number generator.

☐ Add the code below to create a random integer between 0 and 49 and store this value in intRandomNumber (you do not need to add the comment).

```
intRandomNumber = Int((50) * Rnd) ' Int ((49 - 0 + 1) * Rnd + 0)
```

> *NOTE:* Each state is stored as a single record in the layer's attribute table, and each record has a unique index number. The index numbering begins at zero.

The code above has produced a random integer between 0 and 49. To use this number to select a feature from the States layer, you need to pass this integer to a sub procedure that has already been written for you called SelectState. This procedure will first select a state based on the random number, produce a feature definition (subset) using the chosen state, and then zoom to the extent of the feature.

☐ Read through the SelectState procedure to get a feel for what the code is doing.

Do not be discouraged if you do not fully understand this code; it will make much more sense to you once you are more familiar with ArcObjects. Notice that one of the first lines of the procedure sets a variable called m_strStateName; this variable will contain the name of the selected state.

☐ Return to the *Userform_Initialize* event procedure.

☐ Call the sub procedure SelectState, passing the variable intRandomNumber in the parameter list.

STEP 4: WRITE CODE TO COMPARE TWO VALUES

In this step, you will add code to the form's Guess button. The code will ask the user to type in a guess and then compare the user's guess with the selected feature's Name attribute. If the user is correct, he/she is congratulated and another random state is chosen. If the user is not correct, the number of guesses is incremented and he/she is encouraged to guess again.

☐ Double-click on the *cmdGuess* button to reveal the code for its click procedure.

☐ Begin by declaring two static variables: one to keep track of the number of guesses for the current state and another for keeping track of the total number of guesses.

```
Static intNumberOfGuesses As Integer
Static intTotalGuesses As Integer
```

☐ Declare the following variable with the Dim statement to store the user's guess.

```
strGuess As String
```

☐ Increment the number of guesses for this state by setting intNumberOfGuesses equal to intNumberOfGuesses + 1.

☐ Also increment the *total* number of guesses by setting intTotalGuesses equal to intTotalGuesses + 1.

☐ Set lblGuesses' Caption property equal to **`intNumberOfGuesses`**

☐ Set lblTotalGuesses' Caption property equal to **`intTotalGuesses`**

☐ Use an InputBox to prompt the user for a guess, as shown below. Store the guess in the strGuess variable.

```
strGuess = InputBox ("This State Is: ", "Submit Guess")
```

NOTE: To make your game a bit more forgiving, add *Option Compare Text* to the General Declarations of the code module. Option Compare Text ensures that text strings will be considered equal regardless of case ("Utah" = "uTaH", for example).

☐ Write an If Then statement by completing the code below. This statement will check to see if the user guessed the state correctly, then respond with the appropriate action.

```
If ___________ = m_strStateName Then
   _______ "Congratulations! That is correct.", vbExclamation
   intNumberOfGuesses = 0
   Call UserForm_Initialize '<-This will choose another random state!
```

```
______
   MsgBox "Good try, guess again"
End If
```

Question 1: What will be the value of intNumberOfGuesses the first time the user clicks Guess?____ The second? ____

☐ Export frmGuessState as **`frmGuessState.frm`** to your *Student* directory.

☐ Save the ArcMap Document as **`GuessState.mxd`** to your *Student* directory.

☐ Test the form.

When you first start (initialize) the form, your display should zoom to a single selected state. You should be able to click the Guess button and enter a guess for the state's name. Message boxes should appear following each guess informing you whether your guess was correct or not. When you correctly guess the state, you should see a congratulatory message box, and your display should zoom to the extent of another state.

☐ Notice how the number of guesses is updating with every click of the Guess button. The Static variables intNumberOfGuesses and intTotalGuesses retain their value between procedure calls.

☐ Close the form when finished guessing.

STEP 5: WRITE CODE TO DISPLAY ATTRIBUTE INFORMATION

In this step, you will code the Hint button. The Hint button will display information in a message box from a randomly selected field in the State layer's attribute table.

☐ In the form's code window, navigate to the *cmdHint_Click* event procedure.

☐ Use code similar to that in the UserForm_Initialize procedure to produce a random number, this time in the range of 3 to 9 (refer back to the formula). Store the random integer in a variable, intRandField.

☐ Write the code below to display a randomly chosen attribute in a message box (notice that the value appears as the message and the field name as the title of the message box).

```
MsgBox m_pStateFeature.Value(intRandField), vbOKOnly, _
       m_pStateFeature.Fields.Field(intRandField).Name
```

☐ Save and test the form.

☐ Click *Guess* and enter a value into the Input box.

☐ Click *Hint* a few times to get some attribute information about the state.

☐ Close the form.

CHALLENGE: ADD A HINT PENALTY

Currently, the user is not penalized for clicking on the Hint button.

☐ Add code to increase the total number of guesses by five any time the user clicks this button.

CHALLENGE: CODE THE NEW STATE BUTTON

Currently, there is no code behind the New State button. This button should randomly choose and zoom into another state. If the current state is too hard for the user to guess, he/she can simply have the application pick another (hopefully more familiar) state.

☐ Add code to generate a random number in the proper range for selecting a state.

☐ Call the proper procedure to select and zoom into a new state.

☐ Penalize the user by incrementing the total number of guesses by 2.

EXERCISE END

ANSWERS TO EXERCISE 4B QUESTIONS

Question 1: What will be the value of intNumberOfGuesses the first time the user clicks Guess? The second?

Answer: 1, because it is a static variable. The second time the user clicks Guess, it will increment by one to 2.

QUIZ

☐ Option Explicit forces you to declare all variables before they are used in a program (True/False).

☐ Procedure-level variables declared with the *Dim* statement are reinitialized each time the procedure runs, while variables declared as *Static* are held in memory even after a procedure runs (True/False).

☐ Write a line of code to generate a random number between 100 and 135. _______
____

☐ Variables (in Visual Basic) always have to be declared before they are used (True/False).

☐ When you create variables *on the fly* (without declaring them explicitly), what data type is automatically assigned? ________________

SOLUTIONS

```
Option Explicit
Option Compare Text

Private m_pMxApp As IMxApplication
Private m_pMxDoc As IMxDocument
Private m_strStateName As String
Private m_pStateFLayer As IFeatureLayer
Private m_pCityFLayer As IFeatureLayer
Private m_pLayerDef As IFeatureLayerDefinition
Private m_pStateFClass As IFeatureClass
Private m_pStateFeature As IFeature
Private intTotalGuesses '<--For challenge step
Private Sub cmdGuess_Click()
  Static intNumberOfGuesses As Integer
  'Static intTotalGuesses As Integer
  Dim strGuess As String

  intNumberOfGuesses = intNumberOfGuesses + 1
  intTotalGuesses = intTotalGuesses + 1

  lblGuesses.Caption = intNumberOfGuesses
  lblTotalGuesses.Caption = intTotalGuesses

  strGuess = InputBox("This State is: ", "Guess")
  If strGuess = m_strStateName Then
      MsgBox "The state is " & m_strStateName, vbExclamation, "Correct"
      intNumberOfGuesses = 0
      Call UserForm_Initialize
  Else
      MsgBox "Sorry, that is incorrect", vbInformation, "Try Again"
  End If
End Sub

Private Sub UserForm_Initialize()
  Set m_pMxApp = Application
  Set m_pMxDoc = ThisDocument
  Set m_pStateFLayer = pMxDoc.FocusMap.Layer(1)
  Set m_pCityFLayer = pMxDoc.FocusMap.Layer(0)

  Dim intRandomNumber As Integer
  Randomize
  intRandomNumber = Int((50) * Rnd)
  Call SelectState(intRandomNumber)
End Sub

'Challenge 1
Private Sub cmdHint_Click()
  Dim intRandField As Integer
  intRandField = Int((9 - 3 + 1) * Rnd + 3)
  MsgBox m_pStateFeature.Value(intRandField), vbOKOnly, _
         m_pStateFeature.Fields.Field(intRandField).Name
  intTotalGuesses = intTotalGuesses + 5
  lblTotalGuesses.Caption = intTotalGuesses
End Sub
'Challenge 2
Private Sub cmdRandomState_Click()
  intTotalGuesses = intTotalGuesses + 2
  lblTotalGuesses.Caption = intTotalGuesses
  UserForm_Initialize
End Sub

Public Sub SelectState(RecordNumber As Integer)
  Static intNumStates As Integer
  Set m_pStateFClass = m_pStateFLayer.FeatureClass
  Set m_pStateFeature = m_pStateFClass.GetFeature(RecordNumber)
  m_strStateName = m_pStateFeature.Value(2)
  'Select the State
  Set m_pLayerDef = m_pStateFLayer
  m_pLayerDef.DefinitionExpression ="STATE_NAME='" & m_strStateName & "'"
  'Select Cities in the state
  Set m_pLayerDef = m_pCityFLayer
  m_pLayerDef.DefinitionExpression ="STATE_NAME='" & m_strStateName & "'"
  m_pMxDoc.ActiveView.Extent = m_pStateFeature.Extent
  m_pMxDoc.ActiveView.Refresh
  intNumStates = intNumStates + 1
  lblNumStates.Caption = intNumStates
End Sub
```

QUIZ SOLUTION

1. *Option Explicit forces you to declare all variables before they are used in a program (True/False).* True. When option explicit is enforced, all variables must be declared (dimensioned) before they are used in code.
2. *Procedure-level variables declared with the Dim statement are reinitialized each time the procedure runs, while variables declared as Static are held in memory even after a procedure runs (True/False).* True. The Static keyword is used to declare a procedure-level variable that retains its value between calls to the procedure.
3. *Write a line of code to generate a random number between 100 and 135.* Int (Rnd * 36) + 100
4. *Variables (in Visual Basic) always have to be declared before they are used (True/False).* False. If option explicit is not enforced, variables can be created without being explicitly declared.
5. *When you create variables "on the fly" (without declaring them explicitly), what data type is automatically assigned?* If a variable is not explicitly declared as a particular type, Visual Basic assigns the variant data type by default.

5 Programming with class

contents

EXERCISE 5: WORK WITH YOUR OWN OBJECT

This exercise will teach you how to write client side code by instantiating an object and accessing its properties and methods. The object you will create is a Country object, which offers members (properties and methods), such as Population, Name, and GrowthRate for you to use. The server side code, being the blueprint of your Country object, has already been written, inside a class module in ArcMap's VB Editor.

EXERCISE SHORTCUT

1. Open *ex05.mxd* from ..\IPAO\Maps. Use the Object Browser to examine the project's *Country* class and fill in the object model diagram below.

2. Instantiate a Country object in the *frmCountry* UserForm's *initialize* event. Then assign the default Name property to the *txtName* control's text property.

3. Write code for the *cmdSetProperties_Click* event procedure. Set the Name, AreaSquareKM, Population, and GrowthRate properties using the values contained in the txtName, txtArea, txtPop, and cboGrowth controls, respectively.

4. Within the *cmdEstimatePop_Click* procedure, call the *GetProjectedPopulation* method and hold the return value in the dblEstPop variable. Display this value in a message box.

STEP 1: EXPLORE THE COUNTRY CLASS STRUCTURE

Before you instantiate any object, you should be familiar with the properties and methods that belong to that object. One resource available to you is the *Object Browser*. In this step, you will use the *Object Browser* to navigate through the server side code and gather information about the members of the Country object, you are about to create.

☐ Start *ArcMap*.

☐ Navigate to *C:\Student\ipao\maps* and open *ex05.mxd.* This map document contains the server side code that you will use to create your Country object.

☐ Start the *Visual Basic Editor*.

☐ In the Project Explorer, you will see four modules, the *UserForm* modules, named *frmCountry* and *frmCity* and the *Class Modules*, named *Country* and *City*. The *frmCountry UserForm* module is where you will write your client side code. The *Country Class Module* is where the server side code has been written.

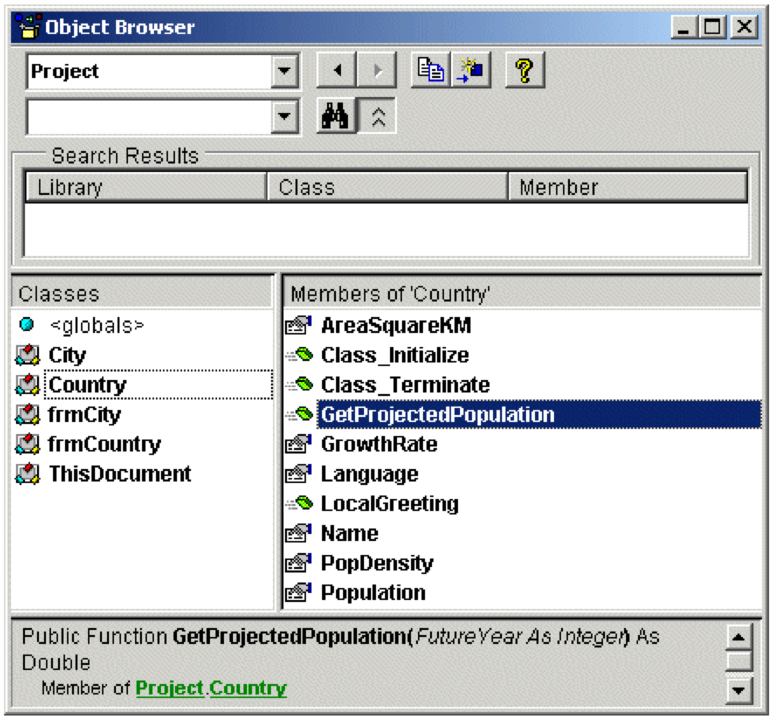
You will now use the Object Browser to learn more about the Country class, and its members.

☐ From the *View* menu, choose *Object Browser* to open the Visual Basic Object Browser dialog.

☐ The pulldown list at the top of the Object Browser contains a list of available libraries. Choose *Project* from this list.

☐ Notice the list of classes displayed in the left side of the browser. Every module contained in your project is listed as a class.

☐ Select (highlight) your *Country* class. On the right side of the browser, you should see a listing of all members (methods and properties) defined for the class.

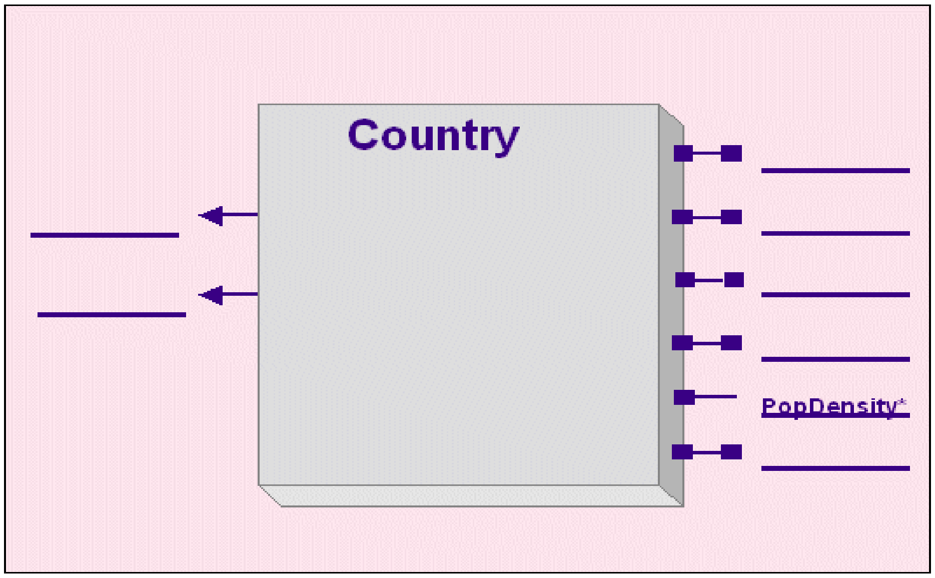


Public Function GetProjectedPopulation(*FutureYear As Integer*) As Double
Member of Project.Country

☐ Select one of the members of the *Country* class. Notice the member definition is displayed at the bottom of the Object Browser dialog.

NOTE: You may find more than members (methods and properties) listed, for example, Class_Initialize and Class_Terminate. To determine if an item listed is an actual property or method, select the member, then examine the bottom panel. If it is a property or method, the scope will be Public.

☐ What properties are available on this class? What are the methods found on the class? Fill in the diagram below with your answers.



NOTE: The Object Browser does not provide information regarding the read/write status of the properties. To locate this information, you may browse through the Country Class Module.

Question 1: Is the PopDensity property a read or write property? _______________

☐ Close the Object Browser when you are finished exploring the Country class.

STEP 2: CREATE A COUNTRY OBJECT AND SET ITS PROPERTIES (WRITE CLIENT CODE)

In this step, you will instantiate a Country object, thereby creating an instance of the Country Class. You will then set and read its properties (Name, Population, etc.). The client side code will be placed in the *frmCountry* form. The Country object will be created in the *frmCountry* form initialize event.

☐ In the *Project Explorer*, open the *frmCountry* form in the *Forms* folder under *Project*. Examine the form layout.

This form has several controls designed for working with a *Country* object you are about to create.

☐ Double-click the form to view the form's code module. Notice that much of the form's code has been written for you.

☐ Navigate to the *UserForm_Initialize* event procedure. Notice that a lot of code has already been provided. Can you tell what this code is doing? __________________

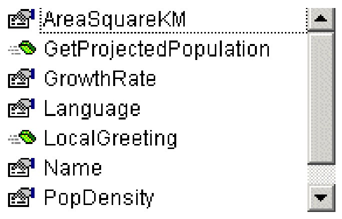
☐ Within the *UserForm_Initialize* procedure, write the two lines of code to dimension and instantiate a new Country object.

```
Dim pCountry as Country
Set pCountry = New Country
```

Question 2: What does the *p* preceding the variable name mean? ______________

☐ On the next line, write the variable **pCountry**, which points to your new Country object. Then place a period after the variable. Do you get a listing of the properties and methods you examined in the Object Browser?__________________________

```
pCountry.
```



NOTE: If you do not see a drop down list of members when you place a period after the object variable, check the spelling of your variable name in your dimension statement.

☐ The Country's Name property has been assigned a default value upon object creation. Display this value in a message box.

```
msgbox pCountry.Name & " is the name of your country."
```

Although a new Country is created when the form is first initialized, most of its properties, except for Name, will not be set until the user clicks the Set Properties button (*cmdSetProperties*).

☐ Run the form.

Question 3: What is the default name of your Country object? _________________

NOTE: Don't worry. You will be able to change the name of your Country object later.

☐ Close the form without entering any information.

☐ Display the form designer portion of your frmCountry form. Examine the controls on the form. Is there a control on the form where the Name property should be displayed?

Question 4: What is the name of the control?______________________________

☐ Examine the Properties Window for that control.

Question 5: What property on the control can you use to display the Country's name? ____

☐ Return to the *UserForm_Initialize()* event and comment out the msgbox statement. Instead, assign the Country's Name property to the *txtName* control's text property.

```
txtName.Text = pCountry.Name
```

☐ Run the form. Does the default name display in the textbox?

STEP 3: SET THE PROPERTIES OF YOUR COUNTRY OBJECT

In this step, you will write the code to set properties to your country object, when the user clicks on the *Set Properties* command button.

☐ Navigate to the *cmdSetProperties_Click* event procedure. Notice the code that has already been written.

Question 6: Can you tell what this code will do? __________________________

☐ Above the existing code, write the variable that points to your Country object, then place a period after the variable name.

Question 7: Do you see the drop down list of members for the Country object? Why not? ____
Hint: Was the variable dimensioned at the procedural level or modular level?

☐ Cut and paste the statement that dimensions the `pCountry` variable from the *UserForm_Initialize* event procedure to the General Declarations section.

NOTE: The *pCountry* variable is now a modular level variable and can be accessed by all the procedures found in the *frmCountry* code module.

☐ Return to the *cmdSetProperties_Click* event procedure. Set the following properties of your Country object. To do this, use the syntax similar to what is shown below.

```
pCountry.<Property> = <Value>
```

Property	***Value***
Name	txtName.Text
AreaSquareKM	txtArea.Text
Population	txtPop.Text
GrowthRate	cboGrowth.Text / 100

☐ Save the map, then run the form.

☐ Test the form by providing values in the controls, then click *Set Properties*.

☐ Test the projected population command button (it has a caption of --??--) and verify that it is 0.

In the next step, you will add code to the projected population button to estimate the population for a given year.

STEP 4: WRITE CODE TO REPORT A PROJECTED POPULATION

Now that you have set all the properties for your country, you can call the *GetProjectedPopulation* method. This method will use the *GrowthRate* and *Population* properties to calculate a projected population for a given year.

☐ In the *frmCountry* code module, navigate to *cmdEstimatePop_Click* procedure.

Notice the code that is already written for you. A variable has been dimensioned (`dblEstPop` As Double) to hold the value returned from the GetProjectedPopulation method; the value of this variable is then displayed as the command button's caption.

☐ Write the line of code below (between the `Dim` statement and the line that sets the button's caption) to pass the date selected on the form to the GetProjectedPopulation method. The return value is stored in `dblEstPop`.

```
dblEstPop = pCountry.GetProjectedPopulation (cboPopYear.Text)
```

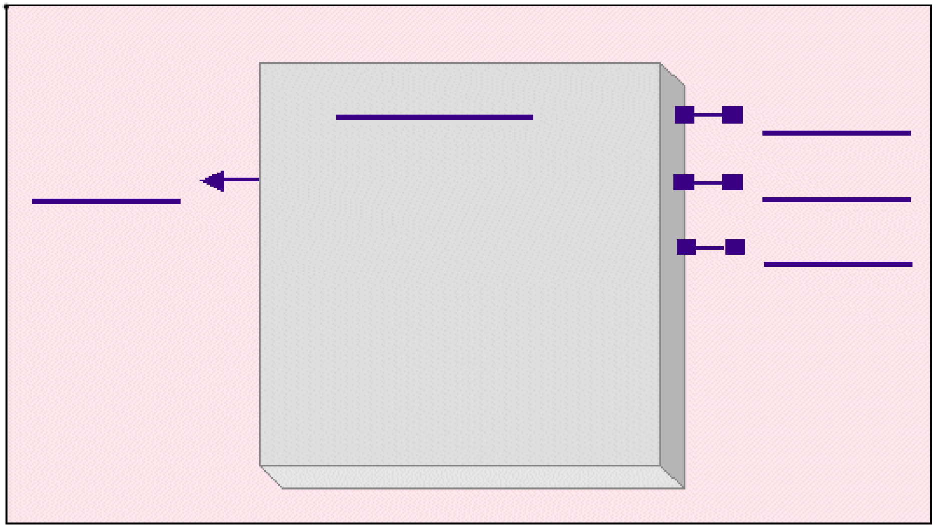
☐ Run the form and test your code. Refer to the end of the exercise for some country statistics or use your own.

Question 8: Based on the provided information, what is Chile's projected population for 2012?____

CHALLENGE: CREATE A CITY OBJECT AND SET ITS PROPERTIES

In this challenge step, you will create an instance of the city class and set its properties. The city object will be created in the *frmCity* form's initialize event and its properties will be set when the user clicks on the *Set Properties* button. But before writing any code, examine the City class using the *Object Browser*.

☐ Using the *Object Browser*, examine the members of the City class and fill in the diagram below.



NOTE: While looking at the Object Browser, you will note that there seem to be 6 properties listed under the Members of 'City' column. There are actually only 3 properties; the other 3 are private module level variables and not available to the client side code. The Object Browser can display this information in the bottom panel. Select a member under the Members of 'City' column, then examine what is displayed in the bottom panel.

☐ Within the *frmCountry* code module, write the code to show the *frmCity* form, when the user clicks on the *Create City* command button.

☐ Run the *frmCountry* form and execute the code to show the *frmCity* form.

As the *frmCity* form is being loaded, the initialize event is being fired. You will now write code to create a City object while the *frmCity* form is being initialized.

☐ Open the *frmCity* code module. Write the code to instantiate a city object when the form is being initialized.

Hint: You will use similar code to create a city object as you did to create the Country object. Make sure to make your object variable available to all procedures in the module.

☐ Within the *cmdSaveCityInfo_Click()* event procedure, set the city's properties to the textbox values found on the form. Use the *With* block to do this.

Hint: For help on how to create a *With* block, view the *With* block used in the *frm-Country* form or use your Visual Basic help (F1).

A *With* block provides a shortcut when writing code. Instead of referencing a variable each time you use a method or property, you can reference it once using the `With` keyword, then use the methods and properties you need, and finish using `End With`. Although it does not save you very much typing, it does make your code more efficient

☐ Call the CityInfo method when the *cmdGetCityInfo* button is clicked.

EXERCISE END

ANSWERS TO EXERCISE 5 QUESTIONS

Question 1: Is the PopDensity property a read or write property?

Answer: The PopDensity property is a Read property only. To view this, examine the code for this property in the Country class

Question 2: What does the p preceding the variable name mean?

Answer: This stands for *pointer*, as object variables do not really contain the object (the way an integer variable contains an integer); instead, they simply point to the referenced object.

Question 3: What is the default name of your Country object?

Answer: The default name of the Country object is Canada. This was initialized in the Country class module, during the Class_Initialize() event.

Question 4: What is the name of the control?

Answer: The name of the TextBox control is txtName.

Question 5: What property on the control can you use to display the Country's name?

Answer: Use the Text property on the txtName control to display the Country's name.

Question 6: Can you tell what this code will do?

Answer: The code will extend the length of the form, revealing additional controls.

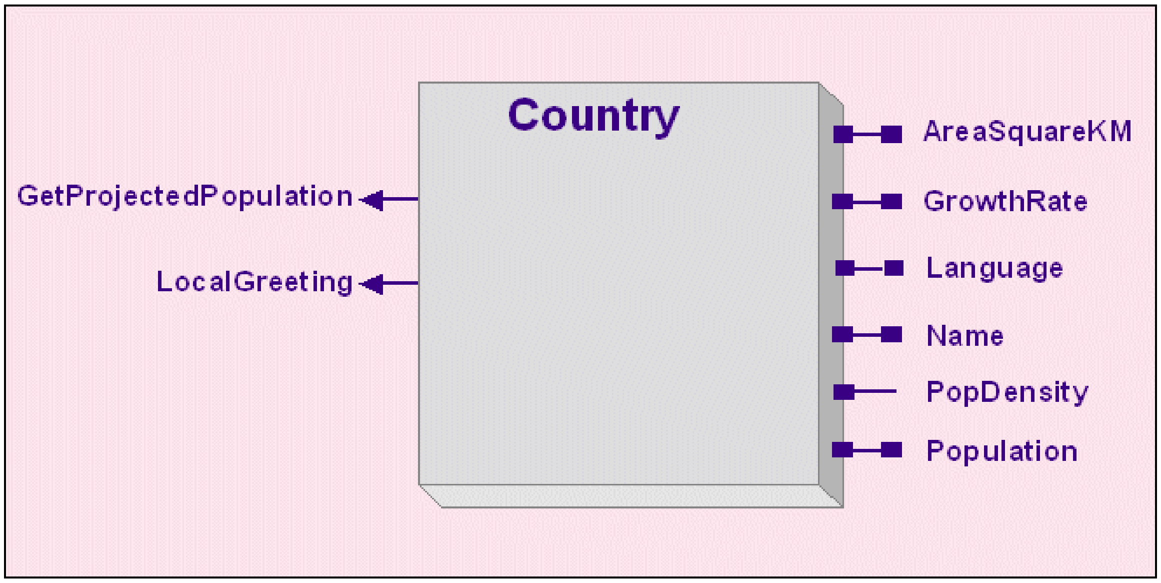
Question 7: Do you see the drop down list of members for the Country object? Why not?

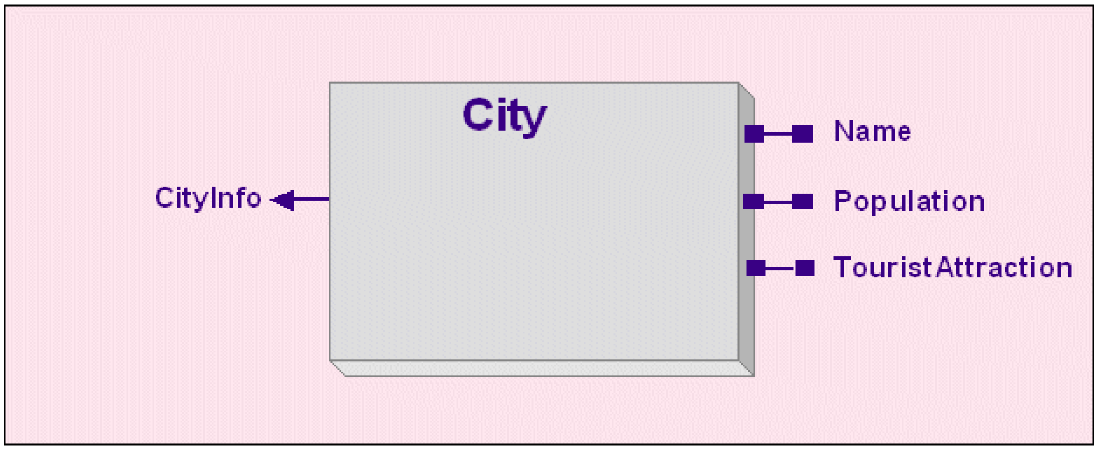
Answer: No, you will not see a drop down list of members for the Country object. The variable referencing the Country object, was dimensioned as a procedural level variable. To have the variable available to all procedures in the module, it must be dimensioned as a module level variable (declared in the General Declarations)

Question 8: Based on the provided information, what is Chile's projected population for 2012?

Answer: Because you are using a variable named intYearsFromNow, which references the current date, the answer will vary depending on when the user clicks the projected population button.

OBJECT MODEL DIAGRAMS





Country statistics

Bangladesh

Population: 129,798,000
Area: 133,910 square km
Growth Rate: 1.8%
Language: Bangla

Canada

Population: 31,006,000
Area: 9,220,970 square km
Growth Rate: 1.1%
Language: English, French

China

Population: 1,246,872,000
Area: 9,326,410 square km
Growth Rate: 0.8%
Language: Chinese (several dialects)

Chile

Population: 14,974,000
Area: 748,800 square km
Growth Rate: 1.3%
Language: Spanish

United States

Population: 270,312,000
Area: 9,158,960 square km
Growth Rate: 0.9%
Language: English

SOLUTIONS

FRMCITY

```
Private pCity As City

Private Sub cmdGetCityInfo_Click()
  pCity.CityInfo
End Sub

Private Sub cmdSaveCityInfo_Click()

  With pCity
    .Name = txtName.Text
    .Population = txtPopulation.Text
    .TouristAttraction = txtTouristAttraction.Text
  End With


  cmdGetCityInfo.Enabled = True
End Sub

Private Sub UserForm_Initialize()

  Set pCity = New City

End Sub

Private Sub txtTouristAttraction_Change()
    Call Enabler
End Sub

Private Sub txtName_Change()
    Call Enabler
End Sub

Private Sub txtPopulation_Change()
    Call Enabler
End Sub

Private Sub Enabler()
    If txtTouristAttraction.Text = "" Or txtName.Text = "" Or Not
IsNumeric(txtPopulation.Text) Then
        cmdSaveCityInfo.Enabled = False
        Exit Sub
    Else
        cmdSaveCityInfo.Enabled = True
    End If
End Sub
```

FRMCOUNTRY

```
Option Explicit

'**Step 3: declare a variable to hold a country object
  Dim pCountry As Country

Private Sub cmdCreateCity_Click()
  Call frmCity.Show
End Sub

Private Sub UserForm_Initialize()
      '**Step 2: Declare and initialize the country variable
      'Dim pCountry As Country
      Set pCountry = New Country
      MsgBox pCountry.Name & " is the name of your country."
      txtName.Text = pCountry.Name


      Me.Height = 147

      Dim intYear As Integer
      For intYear = 2002 To 2100
          cboPopYear.AddItem intYear
      Next intYear

      Dim intCount As Integer
      For intCount = -200 To 1000
          cboGrowth.AddItem intCount / 10
      Next intCount

End Sub

Private Sub cmdSetProperties_Click()
  '**Step 4: set Country properties
    pCountry.Name = txtName.Text
    pCountry.AreaSquareKM = txtArea.Text
    pCountry.Population = txtPop.Text
    pCountry.GrowthRate = cboGrowth.Text / 100


    With frmCountry
        .Caption = pCountry.Name
        .Height = 250
        .Width = 291.75
    End With
    cmdCreateCity.Enabled = True
End Sub

Private Sub cmdEstimatePop_Click()
    Dim dblEstPop As Double
    '**Step 4: display the projected population
    dblEstPop = pCountry.GetProjectedPopulation(cboPopYear.Text)

    cmdEstimatePop.Caption = dblEstPop
End Sub

Private Sub txtArea_Change()
    Call Enabler
```

```
End Sub

Private Sub txtName_Change()
    Call Enabler
End Sub

Private Sub cboGrowth_Click()
    Call Enabler
End Sub

Private Sub txtPop_Change()
    Call Enabler
End Sub

Private Sub Enabler()
    If cboGrowth.Text = "" Or txtName.Text = "" Or Not IsNumeric(txtPop.Text) Or Not
IsNumeric(txtArea.Text) Then
        cmdSetProperties.Enabled = False
        Exit Sub
    Else
        cmdSetProperties.Enabled = True
    End If
End Sub
```

6

COM before the storm

contents

EXERCISE 6: WORK WITH COM CLASSES

This exercise will teach you how to create and use COM classes. You will work with the same Country and City classes you made in the last exercise, this time as a COM class. The first step will be to examine the COM classes and interfaces using the Object Browser, to determine which interfaces you will want to use. Then you will learn how to dimension variables to those interfaces, and instantiate your COM objects.

EXERCISE SHORTCUT

1. Open *ex06.mxd* from ..\IPAO\Maps. Explore the classes and interfaces provided. Complete the two Object Model Diagrams in this exercise for the Country and City classes.

2. In the *ThisDocument* module, create a sub procedure named *CountryTest*. Instantiate a Country object, assign it a few properties, including Name and Language, and call the LocalGreeting method.

3. QueryInterface to the IGovernment interface and call the Vote method.

4. Create a new procedure called *CityGovernment*. Instantiate a City object and call its Vote method. Compare the implementation of the City's Vote method to that of the Country's Vote method.

5. Open the *frmGovern* form and write code in the two appropriate OptionButton event procedures to set m_pGovern to a new City or a new Country, respectively. Within the cmdVote and cmdCollectTax controls, use the Typeof Keyword, to determine if the m_pGovern variable is referencing the Country or City object and call the appropriate message box statement.

STEP 1: .EXPLORE COM CLASS STRUCTURE

In this step, you will explore the basic structure of a COM class. COM classes use interfaces, which are simply definitions of methods and properties. When you use a COM class in your client code, you will always need to communicate with the class through one of its supported interfaces.

☐ Start *ArcMap*.

☐ Navigate to *C:\Student\ipao\maps* and open *ex06.mxd.* This map does not contain any layers but does have a lot of code that has been written for you.

☐ Start the *Visual Basic Editor.*

☐ In the *Project Explorer*, examine the number and names of the classes and interfaces.

Question 1: How many classes are there and what are their name(s)?

____

Question 2: How many interfaces are there and what are their name(s)?

____

Hint: The interfaces are the class modules that are prefixed with an I.

☐ Locate and open the *ICountry* Class Module.

The ICountry class module defines an interface. By convention, class modules that define interfaces are named with a preceding 'I'.

☐ Explore the code in this module. Notice that there is no implementation here, only definitions for properties and methods.

Using interfaces allows the developer to separate the definition of methods and properties (interface design) from the implementation of them (server code). In the last exercise, the first step in creating your Country class was to simply define methods and properties; for a COM class, these would have been written as an interface. After defining members for your Country class, you then proceeded to write code to make them work; this is analogous to implementing an interface for a COM class.

☐ Close the *ICountry* module.

☐ Open the *Country* class module.

☐ Notice the second line of code in the Country module (after `Option Explicit`): *Implements ICountry.* This means the Country class will have code for every method and property defined on the ICountry interface.

☐ Pull down the object list on the *Country* module (upper left) and select *ICountry.*

☐ Pull down the procedure list (upper right). Here you will find all the methods and properties you saw defined in the *ICountry* module.

Notice that, with one exception, there are two listings for each property: one for *Property Get* (read) and one for *Property Let* (write). As an interface designer, you can determine whether properties are read-write, read-only, or write-only. As a class developer, you can control *how* these properties are accessed. As a user of a COM class (writing client code), you will need to know which properties are read or write only.

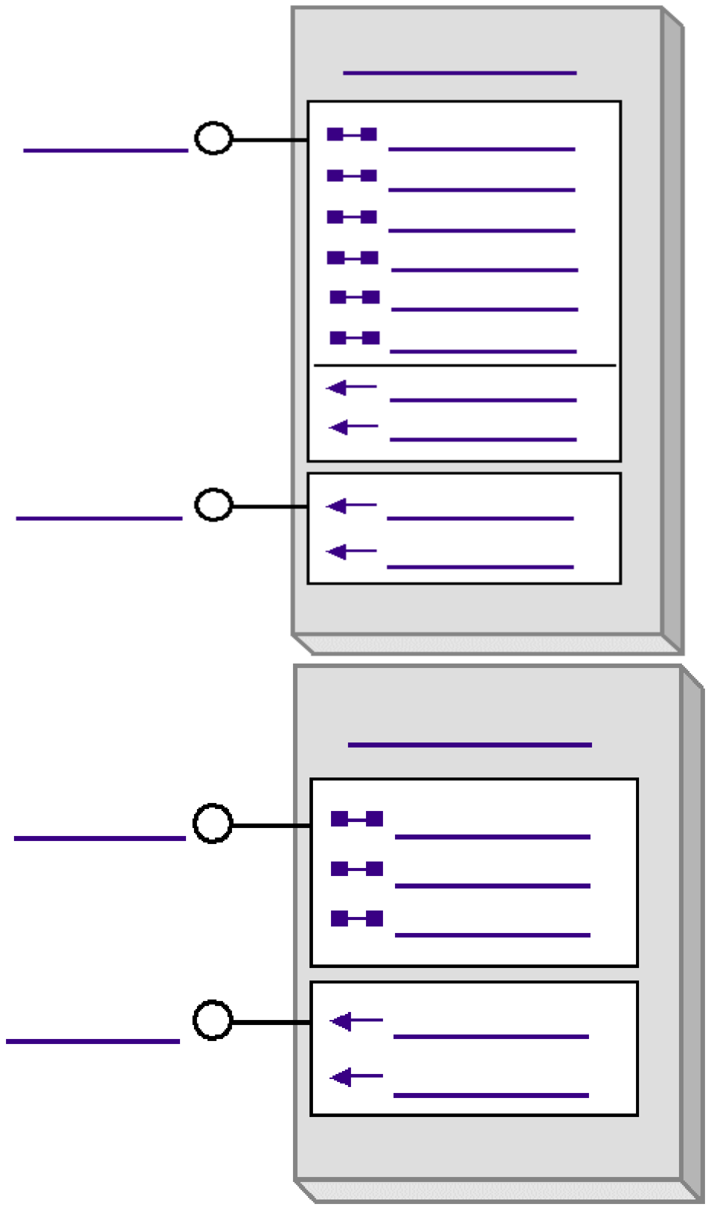
Question 3: What is the only read-only property on the ICountry interface?

____

NOTE: Methods will also only appear once in the dropdown.

☐ Using the Object Browser and the information gathered above, fill in the two diagrams below.

Each rectangular box represents a COM class. Determine which class should be assigned to each box, write the interfaces each class support, and finally, complete the diagram with the properties and methods belonging to each interface.



Step 2: Use your OMD to help you create a Country object

In this step, you will use the object model diagram you completed above to help you create a Country object and assign it a name and language.

☐ Open the project's *ThisDocument* code module.

☐ Insert a new public procedure called `CountryTest`

You will be creating a Country object and assigning it a name and language. The code is similar to the code you wrote in the last exercise. However, in this case, you are using a COM class. To communicate with a COM class, you must *always* declare your variable to one of its interfaces.

Your first step will be to decide what interface you want to use, then dimension a variable to it. To do this, examine your completed Object Model Diagram above.

Question 4: What two interfaces does the Country class implement? ___________

Question 5: Which of these two interfaces provides the Name and Language properties? ____

☐ Inside the *CountryTest* procedure, dimension a variable, called `pCountry`, to the interface that provides the Name and Language properties.

☐ Now instantiate your Country object. Set the `pCountry` variable to a **new** instance of the Country class.

Your code should look similar to the following:

```
Dim pCountry As ICountry
Set pCountry = New Country
```

☐ Type `pCountry` followed by a dot (`.`). Does the Visual Basic code completion dialog display the Name and Language properties?

NOTE: If the code completion does not appear, then take a look at your code. Verify that your code looks like the two lines of code written above.

☐ Assign your new Country object a name and language.

☐ Assign your object any other properties you want.

☐ Call the LocalGreeting method.

Your code should look similar to the following:

```
pCountry.Name = "Canada"
pCountry.Language = "English"
pCountry.Population = 1000000
pCountry.LocalGreeting
```

☐ Run your code. Does a message box appear with the local greeting for the language you specified?

NOTE: If a local greeting does not appear for the language you specified, examine the ICountry_LocalGreeting() method found in the Country class.

STEP 3: QUERYINTERFACE TO THE IGOVERNMENT INTERFACE

You will now add to your existing client side code, calling the Country Object's Vote method.

☐ Within your `CountryTest` procedure, comment out the LocalGreeting method call.

You will now write the code to call the method Vote.

Question 6: On the next line, type **`pCountry`** followed by a dot(.). Does the Visual Basic code completion dialog show a Vote method? Why not?

____

____

Question 7: By examining the Object Model Diagram you filled out in Step 2, what interface provides the method Vote?____________________________________

Remember, when you are dealing with a COM object, you always decide first what interface you want to use, then dimension a variable to it. In this case, you want the IGovernment interface, because it provides the Vote method. This is illustrated on your Object Model Diagram.

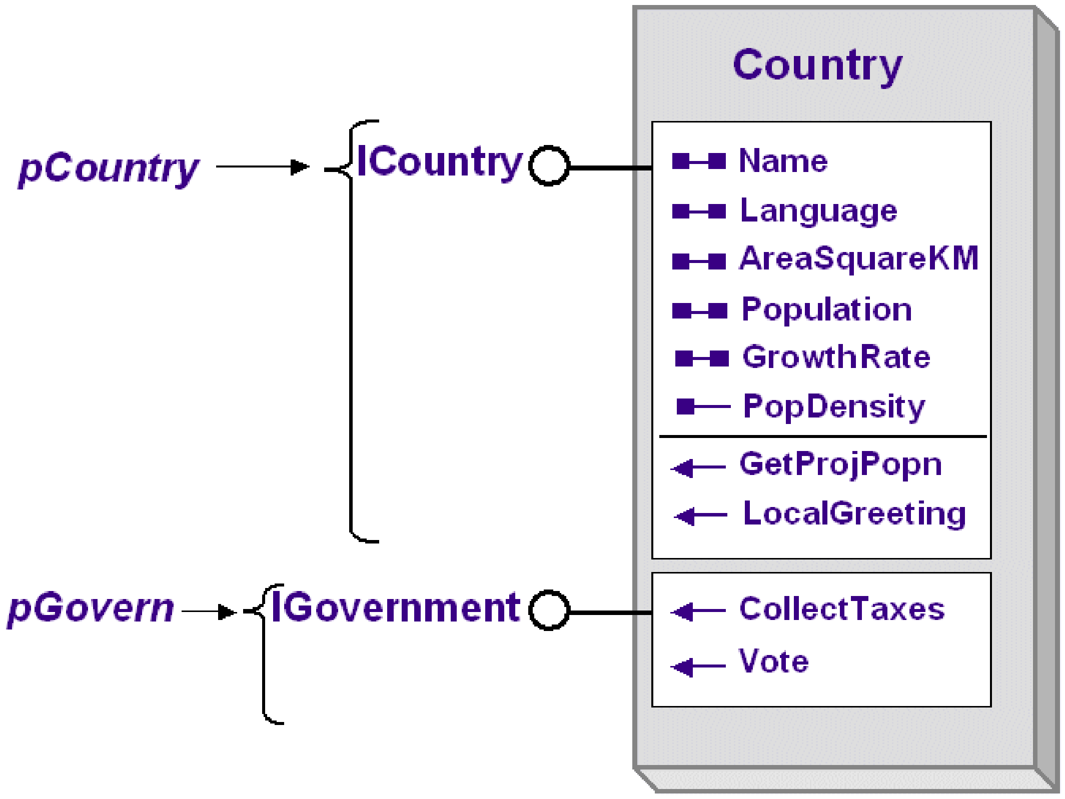
☐ Beneath your existing code, dimension a new variable that communicates with the IGovernment interface, as shown below.

```
Dim pGovern As IGovernment
```

☐ Now write the code to queryinterface to your IGovernment interface. By doing a queryinterface, you are setting the `pGovern` variable to point to the *same* Country object as your `pCountry` variable. (Hint: If you set `pGovern` as a new Country, you will have two *different* Countries.)

```
Set pGovern = __________
```

You now have two variables pointing to the same Country object, through different interfaces. This is illustrated in the diagram below. The `pCountry` variable gives you access to everything on the *ICountry* interface, while `pGovern` allows you to use the two methods you defined on the IGovernment interface.



☐ Complete the code below to work with methods and properties of your Country object.

```
MsgBox "The government of " & ____________.Name & " has voted:" & vbCrLf & _
 _________________.Vote ("Raise Taxes")
```

☐ Run the code.

> *NOTE:* There is a Rnd function found in the implementation of the Vote method. This function returns a random number. If the number generated by the Rnd function is above 50, the proposal is passed. Otherwise the proposal failed.

STEP 4: USE ANOTHER CLASS THAT IMPLEMENTS IGOVERNMENT

One advantage of developing and using COM classes is *polymorphism*. Polymorphism means that several different classes can implement the same interface. The methods and properties defined on an interface cannot change from class to class, but the way in which they are executed (implemented) can.

Question 8: Is there an interface found on your Object Model Diagram that is supported by more than one class? ______________________________________

☐ Open the Country and City class modules and examine the implementation for the IGovernment's methods, Vote and CollectTaxes.

NOTE: Notice that the implementation of these two methods differ between classes.

☐ Open the project's *ThisDocument* code module.

☐ Insert a new public procedure called **`CityGovernment`**

☐ Dimension a variable called **`pCityGovern`** to the interface that provides the Vote method, then instantiate your new City object.

☐ Call the Vote method, passing in a string value for the one argument and display the return value in a message box.

☐ Run your code. Examine the string value returned by the Vote method.

☐ Now run the code found in the `CountryTest` procedure. Did the Vote method found on the Country object return a string value different from the City object?

The City class implements the vote method to explicitly tell you by how many votes the measure passed or failed. The Country class implements the vote method to simply indicate whether the measure passes or fails.

As a client side developer, you do not need to be concerned with how the code is being carried out by the server (City or Country). What you need to know is how to use the methods on IGovernment, and what to expect as a return data type. Each class that supports IGovernment may have different implementation for voting (or calculating tax), but the way you use the interface will always be the same.

CHALLENGE: ASSIGN YOUR CITY OBJECT A NAME

In this challenge, perform a queryinterface to assign your city object a name.

☐ Now that you have instantiated your city object and called the Vote method, queryinterface to the ICity interface and assign your city a name.

Hint: Your first step will be to dimension a variable to the ICity interface. The second step will be to queryinterface.

STEP 5: WORK WITH A FORM THAT ILLUSTRATES POLYMORPHISM

In this step, you will write code for a form that has already been designed for you. Your code will use a single IGovernment variable to illustrate the different implementation of this interface by the Country and City class. You will also use the Visual Basic keyword `TypeOf` to determine if the variable is pointing to the City object or the Country object.

☐ Open the form *frmGovern* in the current project (*ex06.mxd*).

This form will allow you to switch between a Country and City object, then carry out the two methods on IGovernment (Vote, CollectTaxes). You will write the code to create new Country and City objects, and call the Vote and CollectTaxes methods.

☐ Open the form's code module and examine the code that has been written for you.

Notice the single variable that is used by the form `m_pGovern As` IGovernment. Because the IGovernment interface is supported by both the City and the Country class, you can initialize `m_pGovern` as either a City or a Country object.

When the form first opens (`UserForm_Initialize`), `m_pGovern` is set to a new Country.

☐ Write code in the two appropriate *OptionButton* event procedures to set `m_pGovern` to a new City or a new Country, respectively.

Now you will write code for when the user clicks on the Vote and CollectTaxes buttons. The code will bring up a message box, displaying the return value of the Vote method and CollectTaxes method. It will also display whether it is City or Country object being referenced by the `m_pGovern` variable.

☐ Navigate to the `cmdVote_Click()` event procedure.

☐ Write the code to determine if the *type of* object being referenced by the `m_pGovern` variable is a City object.

```
If TypeOf m_pGovern is ICity then
```

☐ On the next line, write a message box that describes the City's vote proposal.

```
MsgBox "The CITY's proposal to " & m_pGovern.Vote(txtProposal.Text)
```

☐ Now finish the If clause with an Else statement for the Country object.

```
Else
    MsgBox "The COUNTRY'S proposal to " & m_pGovern.Vote(txtProposal.Text)
End If
```

☐ Repeat the same steps, using an If/Else statement for the `cmdCollectTax_Click()` event procedure, and incorporate the message boxes that are provided.

☐ Run the form and test your code. Test the option button to verify different objects (City and Country) are being created.

EXERCISE END

ANSWERS TO EXERCISE 6 QUESTIONS

Question 1: How many classes are there and what are their name(s)?

Answer: There are two classes. Their names are Country and City.

Question 2: How many interfaces are there and what are their name(s)?

Answer: There are three interfaces; ICountry, ICity, and IGovernment. The Country class supports the ICountry and IGovernment interfaces. The City class supports the ICity and IGovernment interfaces.

Question 3: What is the only read-only property on the ICountry interface?

Answer: PopDensity

Question 4: What two interfaces does the Country class implement?

Answer: The Country class supports the ICountry and IGovernment interfaces.

Question 5: Which of these two interfaces provides the Name and Language properties?

Answer: The ICountry interface provides the Name and Language properties. This can be seen in the Object Browser.

Question 6: On the next line, type `pCountry` followed by a dot(.). Does the Visual Basic code completion dialog show a Vote method? Why not?

Answer: The Visual Basic code completion will not display the Vote method. The Vote method belongs to the IGovernment interface. The pCountry variable is not communicating with the IGovernment interface, but rather the ICountry interface.

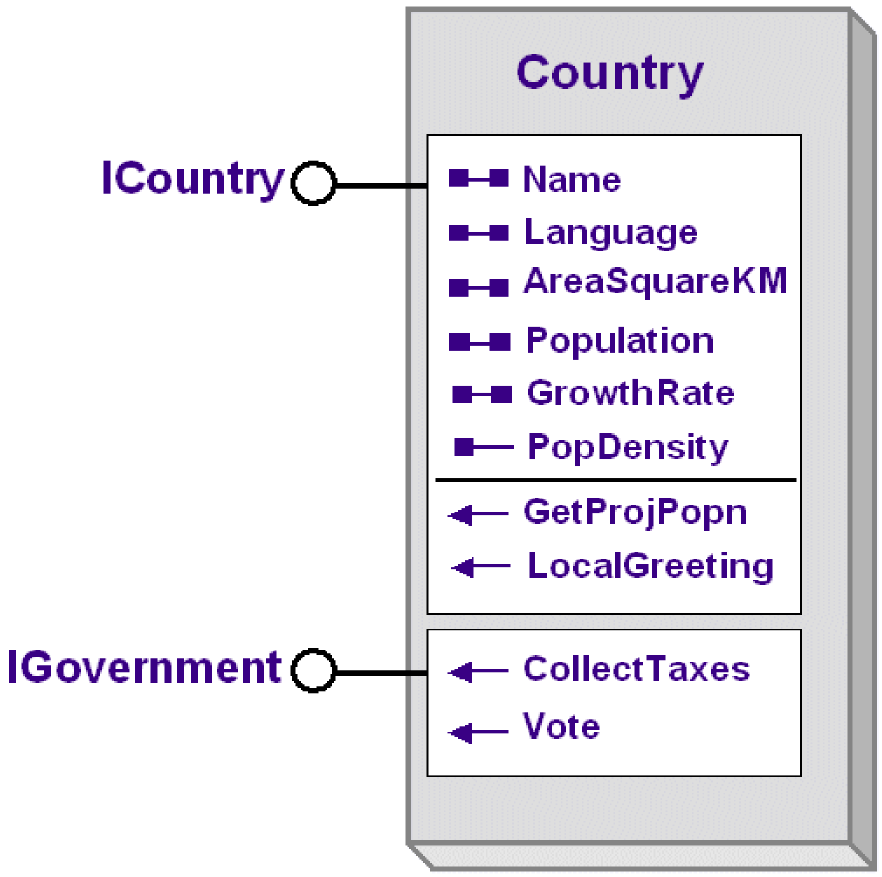
Question 7: By examining the Object Model Diagram you filled out in Step 2, what interface provides the method Vote?

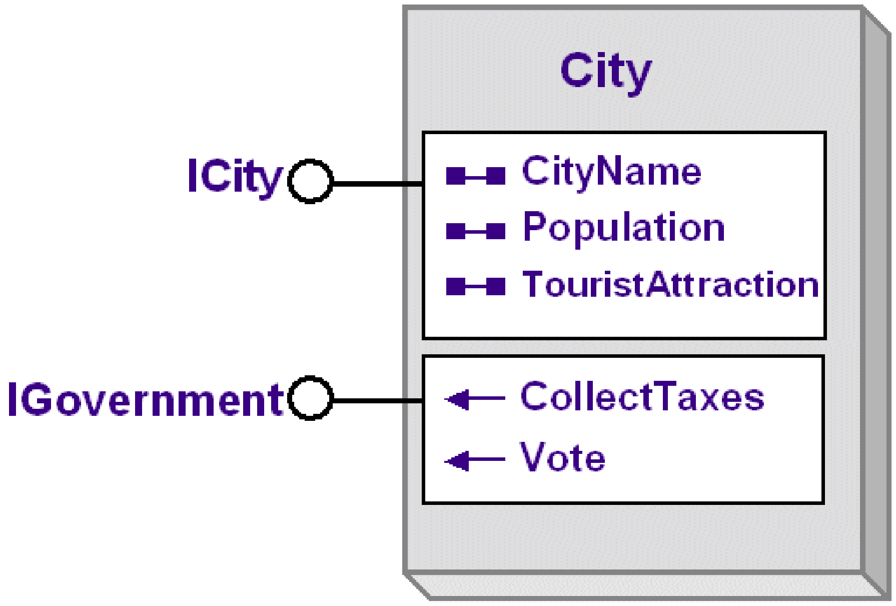
Answer: The Vote method belongs to the IGovernment interface. Therefore to call the Vote method, you will first need to dimension a variable to the IGovernment interface.

Question 8: Is there an interface found on your Object Model Diagram that is supported by more than one class?

Answer: The IGovernment interface is supported by both the Country and City classes. This is an example of Polymorphism, when two or more classes support the same interface.

OBJECT MODEL DIAGRAMS




City
ICity
CityName
Population
TouristAttraction
IGovernment
CollectTaxes
Vote

THISDOCUMENT (EX06.MXD)

```
Public Sub CountryTest()

    Dim pCountry As ICountry
    Set pCountry = New Country

    pCountry.Name = "Sweden"
    pCountry.Population = 12000000
    pCountry.Language = "Swahili"
    'pCountry.LocalGreeting

    Dim pGovern As IGovernment
    Set pGovern = pCountry 'QI

    MsgBox "The government of " & pCountry.Name & " has voted:" & vbCrLf & _
    pGovern.Vote("Raise Taxes")

End Sub

Public Sub CityGovernment()

  Dim pCityGovern As IGovernment
  Set pCityGovern = New City

  MsgBox pCityGovern.Vote("The proposal to raise taxes")

  'Challenge Step
  Dim pCity As ICity
  Set pCity = pCityGovern 'QI

  pCity.CityName = "Dryden"

End Sub
```

FRMGOVERN — FORM MODULE

```
...

Private Sub cmdCollectTax_Click()
    Dim lngTaxes As Long
    lngTaxes = m_pGovern.CollectTaxes(cboTax, txtRevenue)
    If TypeOf m_pGovern Is ICity Then
      MsgBox "Tax collected for the CITY = $" & lngTaxes
    Else
      MsgBox "Tax collected for the COUNTRY = $" & lngTaxes
    End If

End Sub

Private Sub cmdVote_Click()
  If TypeOf m_pGovern Is ICity Then
    MsgBox "The CITY's proposal to " & m_pGovern.Vote(txtProposal.Text)
  Else
    MsgBox "The COUNTRY's proposal to " & m_pGovern.Vote(txtProposal.Text)
  End If
End Sub

Private Sub optCity_Click()
    Set m_pGovern = New City
End Sub

Private Sub optCountry_Click()
    Set m_pGovern = New Country
End Sub
...
```

7

Understanding object model diagrams

contents

Exercise 7A: Navigating the object model diagrams

In this lesson, you will learn how to read object model diagrams. You will begin by practicing with an object model diagram representing country, province, and municipality objects.

Step 1: Explore a fictitious object model diagram

The Country object model diagram included with your course materials has been designed to illustrate the relationship between country, province, and municipality objects. In this step, you will need to decipher the diagram in order to answer questions about the individual classes and the relationships they share.

Question 1: How many interfaces does the Province class support? What are they?

____

Question 2: How many interfaces does the City class support? What are they?

____

Question 3: Which classes can be created with the *New* keyword? ____________

____

Question 4: How can a new Province be created? ___________________________

____

Question 5: Can an instance of the Municipality class be created? _____________

☐ Look at the relationship between Province and Municipality. Based on this relationship...

Question 6: If a Municipality is deleted, its associated Province is also (deleted/lives on independently). If a Province is deleted, its associated Municipalities (are also deleted/live on independently).

Question 7: Which class has a wormhole associated with it? _________________
Where does it go? ____

Question 8: According to the relationships illustrated in the diagram, which of the following are true?

- Country *is a type of* Province.
- Country *is composed of* Municipalities.
- Town *is a type of* Municipality.

- Country *creates a* Province.
- Province *is composed of a single* Municipality.

Question 9: Are there any examples of polymorphism in this object model? If so, name one. ____

Question 10: According to the interfaces described in the diagram, which of the following are true?

- The IGovernment interface on Municipality might have different methods and properties than those listed on Country's IGovernment interface.
- The RulingParty property on IGovernment is read-only.
- The Provinces property on ICountry returns a pointer to the IProvince interface.
- The Capital property on IProvince is set with a reference to an object, not a copy of it.

Step 2: Complete some code based on the diagram

In this step, you will use the Country object model diagram to complete some code.

For answering the questions below, assume the following variables have been declared:

```
pProvince As IProvince
pCountry As ICountry
pCapitalCity As IMunicipality
```

Question 11: Which line of code will correctly assign a new Province capital?

```
pProvince.Capital = pCapitalCity
Set pProvince.Capital = pCapitalCity
```

Question 12: Which line of code below is correct for assigning a new name to the Province?

```
pProvince.Name = "Montagreta"
Set pProvince.Name = "Montagreta"
```

Question 13: Complete the code below to report the number of provinces in pCountry.

```
Dim pAllProvinces As __________________________
Set pAllProvinces = pCountry.__________________
MsgBox "There are " & pAllProvinces.______________ & " provinces in " &
____________.Name
```

Question 14: Complete the code below to determine if pProvince has a capital.

```
Set pCapitalCity = pProvince.Capital
If pCapitalCity Is ______________ Then
```

```
  MsgBox "The province does not have a capital!"
  _______ Sub
End If
```

Question 15: Complete the code below to report whether pProvince's capital is a city, town, or village.

```
Set pCapitalCity = pProvince.Capital
If ____________ pCapitalCity Is ICity Then
  MsgBox "The capital is a City"
ElseIf _____________ pCapitalCity Is ___________________ Then
  MsgBox "The capital is a Town"
Else
  MsgBox "The capital is a Village"
End If
```

Step 3: Explore some ArcObjects object model diagrams

In this step, you will apply your OMD reading skills to the printed ArcObjects object model diagrams that are provided to you in class. You will decipher the ArcObjects diagrams to answer questions about specific ArcObjects classes and the relationships they exhibit. In the next step, you will write code in ArcMap based on information gleaned from the diagrams.

Answer the following questions using the printed Carto Layer and Geodatabase object model diagrams provided to you in class.

Question 16: Look at Layer class and its subclasses on the Carto Layer OMD. Based on the diagram, which of the following are true?

- The FeatureClass property on IFeatureLayer2 is set with a reference to an object, not with a copy of it.
- All types of layers have a MaximumScale property.

Hint: Look at the Layer Abstract class.

- The CompositeGraphicsLayer class supports the IGeoDataset interface.

Answer the following questions using the Geodatabase object model diagram.

☐ Find the WorkspaceFactory class (upper left).

Question 17: WorkspaceFactory is a(n) (Abstract/Instantiable/Creatable) class.

Question 18: A WorkspaceFactory (is a type of/creates a/has a/is composed of) Workspace object.

Question 19: Which interface on the Workspace class has methods for creating new data sources (files on disk)? ____

Question 20: There is a wormhole connecting the WorkspaceFactory class. How many different Object Model Diagrams does this wormhole connect to?

____

EXERCISE END

Answers to Exercise 7A Questions

Question 1: How many interfaces does the Province class support? What are they?

Answer: Province supports two interfaces: IProvince and IGovernment.

Question 2: How many interfaces does the City class support? What are they?

Answer: City supports three interfaces: ICity, IMunicipality (inherited), and IGovernment (inherited).

Question 3: Which classes can be created with the New keyword?

Answer: Country, City, Village, and Town can be created with *New*.

Question 4: How can a new Province be created?

Answer: A new Province object can be created from a Country object (CreateProvince method).

Question 5: Can an instance of the Municipality class be created?

Answer: An instance of the Municipality cannot be created; it is an Abstract class.

Question 6: If a Municipality is deleted, its associated Province is also (deleted/lives on independently). If a Province is deleted, its associated Municipalities (are also deleted/live on independently).

Answer: If a municipality is deleted, the associated Province lives on independently. If a Province is deleted, all associated Municipalities go with it (a Province *is composed* of Municipalities).

Question 7: Which class has a wormhole associated with it? Where does it go?

Answer: The Country class has a wormhole that goes to the Politics OMD.

Question 8: According to the relationships illustrated in the diagram, which of the following are true?

Answer: Town *is a type of* Municipality. Country *creates a* Province.

Question 9: Are there any examples of polymorphism in this object model? If so, name one.

Answer: There are two examples of polymorphism (the same interface used on several classes): IGovernment is supported by all classes on the diagram, and IMunicipality is supported on the City, Town, and Village classes (inherited).

Question 10: According to the interfaces described in the diagram, which of the following are true?

Answer: The *RulingParty* property on IGovernment is read-only. The *Capital* property on IProvince is set by reference.

Question 11: Which line of code will correctly assign a new Province capital?

Answer: On the diagram, this property is listed with a hollow (white) box, which means it is set with a reference to an object: use Set.

```
Set pProvince.Capital = pCapitalCity
```

Question 12: Which line of code below is correct for assigning a new name to the Province?

Answer: On the diagram, this property is listed with a solid (black) box, which means it is set with a value or a copy of an object: no Set.

```
pProvince.Name = "Montagreta"
```

Question 13: Complete the code below to report the number of provinces in pCountry.

Answer:

```
Dim pAllProvinces As IProvinces
Set pAllProvinces = pCountry.Provinces
MsgBox "There are " & pAllProvinces.ProvinceCount & " provinces in " & pCountry.Name
```

Question 14: Complete the code below to determine if pProvince has a capital.

Answer:

```
Set pCapitalCity = pProvince.Capital
If pCapitalCity Is Nothing Then
  MsgBox "The province does not have a capital"
  Exit Sub
End If
```

Question 15: Complete the code below to report whether pProvince’s capital is a city, town, or village.

Answer:

```
Set pCapitalCity = pProvince.Capital
If TypeOf pCapitalCity Is ICity Then
```

```
    Msgbox "The capital is a City"
  ElseIf TypeOf pCapitalCity Is ITown Then
    MsgBox "The capital is a Town"
  Else
    MsgBox "The capital is a Village"
  End If
```

Question 16: Look at Layer class and its subclasses on the Carto Layer OMD. Based on the diagram, which of the following are true?

Answer: The FeatureClass property on IFeatureLayer is set by reference. All types of layers have a MaximumScale property. The CompositeGraphicsLayer class supports IGeoDataset.

Question 17: WorkspaceFactory is a(n) (Abstract/Instantiable/Creatable) class.

Answer: WorkspaceFactory is an Instantiable class.

Question 18: A WorkspaceFactory (is a type of/creates a/has a/is composed of) Workspace object.

Answer: A WorkspaceFactory creates a Workspace.

Question 19: Which interface on the Workspace class has methods for creating new data sources (files on disk)?

Answer: IFeatureWorkspace on the Workspace class has methods for creating new data sources (CreateTable, CreateFeatureDataset, etc.).

Question 20: There is a wormhole connecting the WorkspaceFactory class. How many different Object Model Diagrams does this wormhole connect to?

Answer: There are 7 different OMD's (GISClient, DataSourcesFile, DataSourcesGDB, DataSourcesOLEDB, DataSourcesRaster, TrackingAnalyst and Geoprocessing) that connect to the WorkspaceFactory class.

EXERCISE 7B: USING THE OBJECT MODEL DIAGRAMS

In this exercise, you will write your first code using ArcObjects. You will understand how to navigate and use the Help system in the following steps. You may not be extremely familiar with some of the ArcObjects you work with, but the key is to examine relationships in the Object Models. Many of the ArcObjects you use here will be reviewed in the next lecture and exercise.

EXERCISE SHORTCUT

1. Create a UIEditBox Control that will change the Application's caption.
2. Create a sub procedure that will report the name of the current document, the active data frame name, and the number of layers in the active data frame in a Message Box.

STEP 1: WRITE CODE TO CHANGE THE APPLICATION TITLE

In this step, you will open an ArcMap document and write some code using the ArcMap object model diagram.

☐ Start *ArcMap* and open *ex07.mxd* from your training directory. This map has two data frames: one showing South American countries, cities, and rivers and the other showing ecoregions for Brazil.

☐ Open the *Customize* dialog box.

☐ On the *Commands* tab, highlight the *UIControls* category.

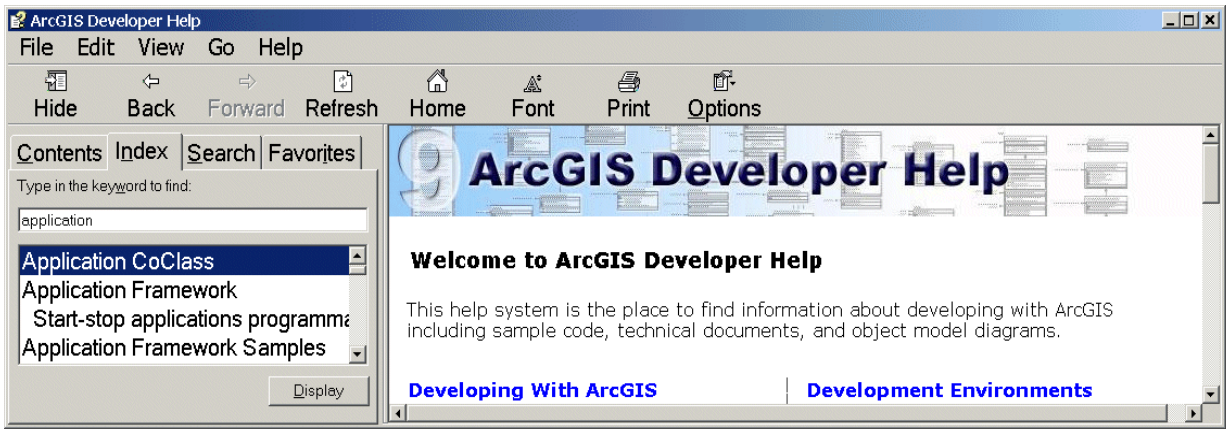
☐ Make sure that *Save in* is set to ex07.mxd. Create a new *UIEditBoxControl* and add it to the interface.

☐ Open the *Visual Basic Editor*.

☐ Navigate to *UIEditBoxControl1's Change* event procedure. The code you write here will execute whenever the contents of the Edit box change.

☐ Open the Developer Help by clicking *Start* > *Programs* > *ArcGIS* > *Developer Help* > *VB6 Help*.

☐ Click the *Index* tab and type **`Application`**


ArcGIS Developer Help
File Edit View Go Help
Hide Back Forward Refresh Home Font Print Options
Contents Index Search Favorites
Type in the keyword to find:
application
Application CoClass
Application Framework
Start-stop applications programma
Application Framework Samples
Display
ArcGIS Developer Help
Developing With ArcGIS
Development Environments

Welcome to ArcGIS Developer Help

This help system is the place to find information about developing with ArcGIS including sample code, technical documents, and object model diagrams.

☐ Double-click *Application CoClass*.

☐ In the *Topics Found* dialog, choose *Application CoClass* with the location of *esriarcmap* and press the *Display* button.

The Application class can represent many different application objects. In this case, you are examining the application object that represents ArcMap. There are also application objects for ArcCatalog, ArcGlobe, and ArcScene. The help panel shows a listing of all interfaces the Application class implements.

☐ In the Help panel for *Application CoClass*, click the *esriArcMap* hyperlink.

☐ Click the Contents tab. Notice the contents shows an overview of the esriArcMap library.

☐ In the *Contents* tab, under *esriArcMa*p, click *ArcMap Library Object Model Diagram* and open the .pdf.

☐ Locate the Application class on your ArcMap object model diagram. The Application variable points to this class.

The Application class is the only class on the ArcMap object model diagram. Most other object model diagrams have many more classes.

Application is a preset variable that is always available in ArcMap or ArcCatalog. This variable is your entry point into the object model, and once you are *in*, you can find a way to navigate to other objects.

NOTE: Most applications that use the VBA development environment will have a preset variable called Application that points to the application in which you are developing. Microsoft products such as Word, PowerPoint, and Outlook all have an Application variable.

Question 1: How many interfaces are supported by the Application class?_______
Hint: You can also view the interfaces on the help page for the Application class.

☐ Return to VBA.

☐ Type `Application` and then a 'dot' (period) to bring up the Visual Basic code completion dialog.

Question 2: Examine the methods and properties displayed in the code completion list. Using the ArcMap OMD, can you determine which interface is the Application variable pointing to? ____
____

You will write some simple code here to change the text that appears on the ArcMap title bar. To change this text, you will set a property of Application.

Question 3: Use your OMD to examine all the properties on the IApplication interface. Which property would you use to change the title bar text? __________
How many properties are read-only? ____________________________________
How many are you able to set (write)?___________________________________

The *Name* property, at first glance, might seem the logical property for changing the title bar text. This is a read-only property, however, and the application name will always be ArcMap (in this case). Look at the properties you can change.

Question 4: Application has a *writable* property for changing the title bar text. What is it? ___________________________ What data type is used to set this property?
____

☐ In the *UIEditBoxControl Change* event procedure, write the code below to set the ArcMap caption with the text in the Edit box control.

```
Application.Caption = UIEditBoxControl1.Text
```

☐ Close the *Visual Basic Editor* and return to *ArcMap*.

☐ Test your code by typing into the Edit box.

The text on the ArcMap title bar should immediately change as you type.

STEP 2: WRITE CODE TO WORK WITH THE DOCUMENT'S ACTIVE DATA FRAME

In this step, you will use the ArcMap and Map Layer OMD's to navigate to objects in the current map. You will use the diagrams to find your way from the Application to the MxDocument, then to the Map class.

☐ Return to the *Visual Basic Editor* and open the *ThisDocument* module under the project (ex07.mxd).

☐ Define the following subprocedure:

```
Public Sub ReportMapInfo ()
```

Question 5: Before writing any code, examine the ArcMap object model diagram. Look at the Application class. Is there a method or property that returns the current document? What is it?_______ What interface pointer is returned by this read-only property?____

☐ Write the code below to dimension a variable to hold the returned interface:

```
Dim pDocument As IDocument
```

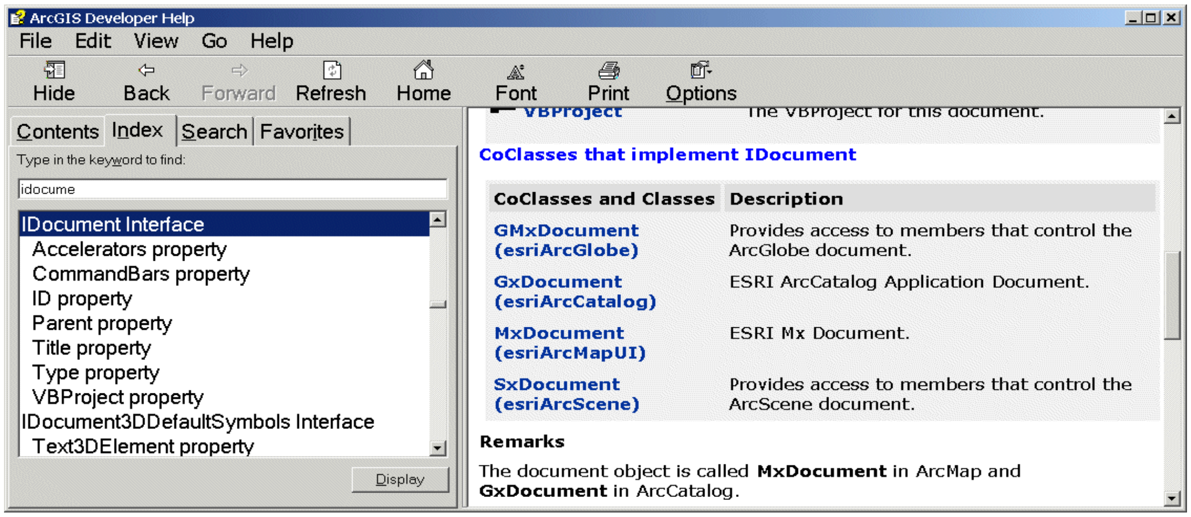
☐ Initialize the variable by writing the following code:

```
Set pDocument = Application.Document
```

Now that the pDocument variable is referencing the Document property, you need to find out exactly what the document represents.

☐ Open the *ArcGIS Developer Help*, click the *Index* tab, and type `IDocument`

☐ On the *IDocument* help page, scroll down to *CoClasses that Implement IDocument*.



CoClasses that implement IDocument

CoClasses and Classes	Description
GMxDocument (esriArcGlobe)	Provides access to members that control the ArcGlobe document.
GxDocument (esriArcCatalog)	ESRI ArcCatalog Application Document.
MxDocument (esriArcMapUI)	ESRI Mx Document.
SxDocument (esriArcScene)	Provides access to members that control the ArcScene document.

Remarks

The document object is called **MxDocument** in ArcMap and **GxDocument** in ArcCatalog.

This is an example of Polymorphism. There are many classes that implement the same interface (IDocument). In this example, the MxDocument (.mxd) and GMxDocument (.3dd for ArcGlobe) both implement IDocument. Because you are working with ArcMap, you will want to use MxDocument.

☐ Click the *MxDocument (esriArcMapUI)* hyperlink.

☐ Read the overview of the MxDocument CoClass and click the *esriArcMapUI* link (upper right corner).

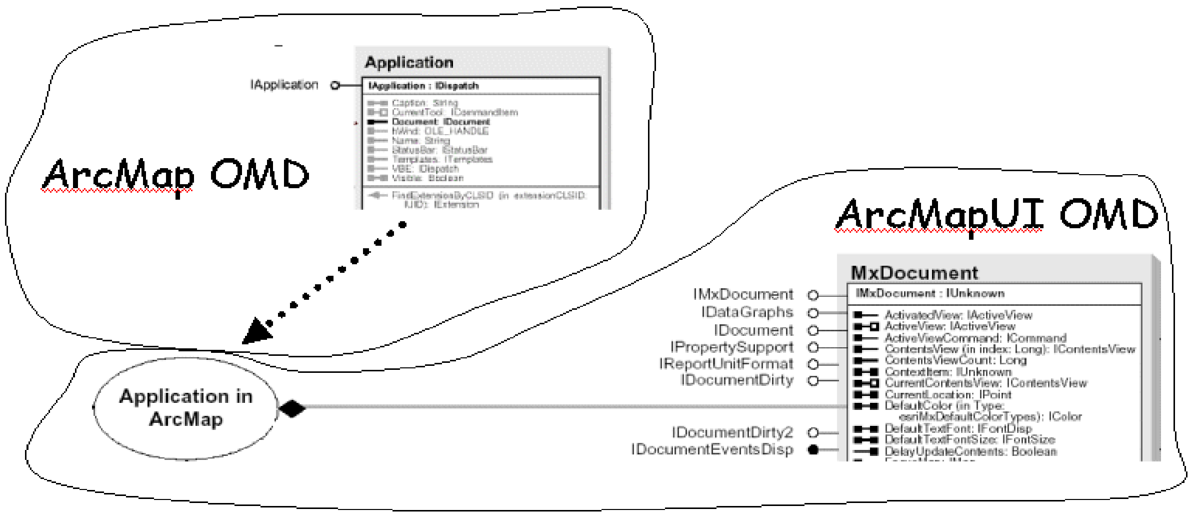
The esriArcMapUI is another library which contains access to another set of ArcObjects.

☐ Click the *Contents* Tab.

☐ Under *esriArcMapUI*, click *ArcMapUI Library Object Model Diagram* to open the .pdf.

☐ Find the MxDocument class on this diagram.

You will notice there is a wormhole connecting the MxDocument class with the ArcMap OMD you first opened in this exercise. So far you have looked at two different Object Model Diagrams, which are connected together by the wormhole as indicated below. The diagram below shows that the Application class is composed of exactly one MxDocument. The wormhole indicates that if you would like to find more information about the Application class, you should look in the ArcMap OMD.



You will notice the IDocument interface on MxDocument does not show all the properties and methods on the OMD. In this case, you can to go back to the help to find out additional information about IDocument.

Question 6: Return to the help and find IDocument. How many properties are listed on the IDocument interface?__ Use the same technique to examine IMxDocument What methods or properties does IMxDocument have for returning the document's map(s) (data frames)? ____
____

☐ Write the code below to create a variable that points to the IMxDocument interface.

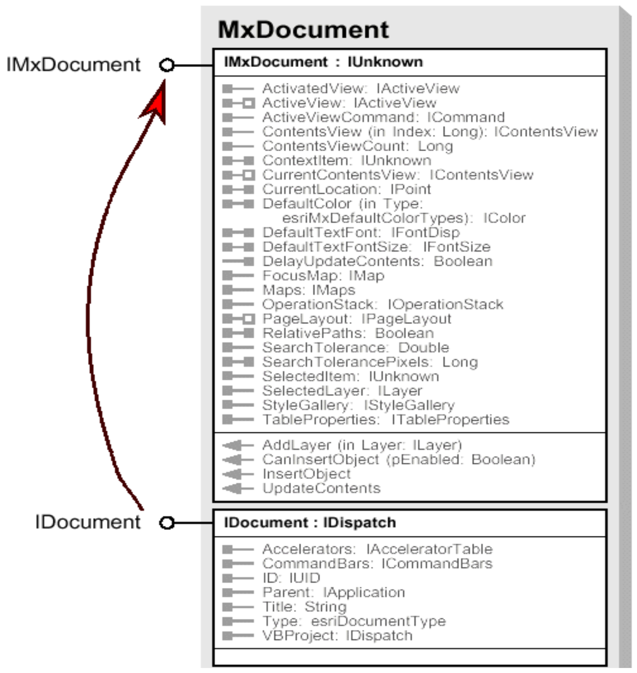
```
Dim pMxDocument As IMxDocument
```

☐ Initialize this new variable with the following code:

```
Set pMxDocument = pDocument
```

You now have *two* variables that point to the *same* object, the current document (MxDocument). You can now access all methods and properties on IDocument by using the pDocument variable and all those on IMxDocument by using pMxDocument. In either case, you are working with the same document.

Question 7: What is the term for referencing another interface on the same object?____


MxDocument
IMxDocument
IMxDocument : IUnknown
ActivatedView: IActiveView
ActiveView: IActiveView
ActiveViewCommand: ICommand
ContentsView (in Index: Long): IContentsView
ContentsViewCount: Long
ContextItem: IUnknown
CurrentContentsView: IContentsView
CurrentLocation: IPoint
DefaultColor (in Type: esriMxDefaultColorTypes): IColor
DefaultTextFont: IFontDisp
DefaultTextFontSize: IFontSize
DelayUpdateContents: Boolean
FocusMap: IMap
Maps: IMaps
OperationStack: IOperationStack
PageLayout: IPageLayout
RelativePaths: Boolean
SearchTolerance: Double
SearchTolerancePixels: Long
SelectedItem: IUnknown
SelectedLayer: ILayer
StyleGallery: IStyleGallery
TableProperties: ITableProperties
AddLayer (in Layer: ILayer)
CanInsertObject (pEnabled: Boolean)
InsertObject
UpdateContents
IDocument
IDocument : IDispatch
Accelerators: IAcceleratorTable
CommandBars: ICommandBars
ID: IUID
Parent: IApplication
Title: String
Type: esriDocumentType
VBProject: IDispatch

NOTE: The diagram you look at will have many more interfaces on the MxDocument class and the IDocument interface will not actually list all of its properties on the OMD. The above illustration has been scaled down to fit on one page.

The IMxDocument interface has two properties that can be used to access the document's maps: Maps and FocusMap. FocusMap will return the current map (active data frame); Maps will return a collection of all the document's maps. Both of these properties will be further reviewed in Lesson 8.

☐ Note that the FocusMap property returns an object. The relationship below shows that the FocusMap property returns an object that supports the IMap interface.

FocusMap: IMap

Any variable that references the FocusMap property will then reference the Active Data Frame in the ArcMap Table of Contents.

☐ Use the *ArcGIS Developer Help* to find the *IMap* Interface as you did with Application and IDocument.

Question 8: What is the only CoClass that implements IMap? ________________
Hint: Scroll to the bottom of the Help page for IMap.

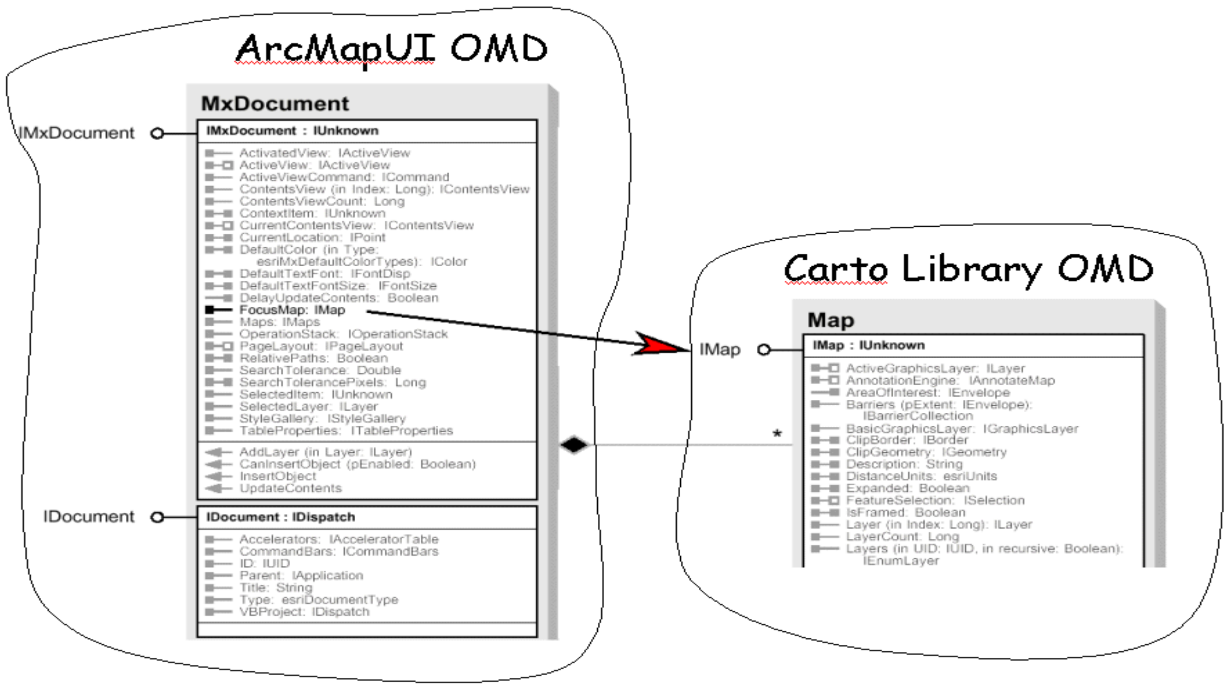
Question 9: Which OMD contains the IMap interface? ______________________
Hint: Click on the library name (esriCarto) and then navigate back to the Contents tab.

☐ Return to the *Visual Basic Editor.*

☐ Dimension a variable to hold the focused map. Dimension the variable as the return type of the property.

```
Dim pMap As IMap

Set pMap = pMxDocument.FocusMap
```


ArcMapUI OMD
MxDocument
IMxDocument
IMxDocument : IUnknown
ActivatedView: IActiveView
ActiveView: IActiveView
ActiveViewCommand: ICommand
ContentsView (in Index: Long): IContentsView
ContentsViewCount: Long
ContextItem: IUnknown
CurrentContentsView: IContentsView
CurrentLocation: IPoint
DefaultColor (in Type: esriMxDefaultColorTypes): IColor
DefaultTextFont: IFontDisp
DefaultTextFontSize: IFontSize
DelayUpdateContents: Boolean
FocusMap: IMap
Maps: IMaps
OperationStack: IOperationStack
PageLayout: IPageLayout
RelativePaths: Boolean
SearchTolerance: Double
SearchTolerancePixels: Long
SelectedItem: IUnknown
SelectedLayer: ILayer
StyleGallery: IStyleGallery
TableProperties: ITableProperties
AddLayer (in Layer: ILayer)
CanInsertObject (pEnabled: Boolean)
InsertObject
UpdateContents
IDocument
IDocument : IDispatch
Accelerators: IAcceleratorTable
CommandBars: ICommandBars
ID: IUID
Parent: IApplication
Title: String
Type: esriDocumentType
VBProject: IDispatch
Carto Library OMD
Map
IMap
IMap : IUnknown
ActiveGraphicsLayer: ILayer
AnnotationEngine: IAnnotateMap
AreaOfInterest: IEnvelope
Barriers (pExtent: IEnvelope): IBarrierCollection
BasicGraphicsLayer: IGraphicsLayer
ClipBorder: IBorder
ClipGeometry: IGeometry
Description: String
DistanceUnits: esriUnits
Expanded: Boolean
FeatureSelection: ISelection
IsFramed: Boolean
Layer (in Index: Long): ILayer
LayerCount: Long
Layers (in UID: IUID, in recursive: Boolean): IEnumLayer
*

Question 10: If you wanted to use the LayerCount property from the Map class, what is the return type?____________________________________
Hint: After the LayerCount property you will see a colon (LayerCount: XXXXX). You can find the return type by also examining the Help.

☐ Finally, you will use the variables you have created so far to get information about the current document and its maps. Dimension the following variables to store properties of the MxDocument and Map classes.

```
Dim strDocName As String
Dim strFocusMapName As String
Dim intLayerCount As Long
```

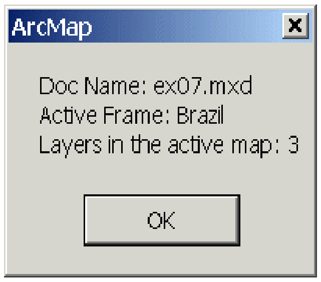
☐ Complete the code below to initialize your variables:

```
strDocName = pDocument.____________
strFocusMapName = ____________.____________
intLayerCount = ______________.______________
```

☐ Report the information in a message box as shown below.

```
MsgBox "Doc Name: " & strDocName & vbCrLf & _
  "Active Frame: " & strFocusMapName & vbCrLf & _
  "Layers in the active map: " & intLayerCount
```

NOTE: vbCrLf (carriage return line feed) is a Visual Basic constant for a line break. The underscore character (_) is used to continue a single line of VB code on several lines in the Editor.


ArcMap

Doc Name: ex07.mxd
Active Frame: Brazil
Layers in the active map: 3

☐ Run your code. A message box should report the document information.

☐ With ArcMap open, hold down *Alt* on your keyboard and click the *South America* data frame to activate it.

☐ Re-execute the *ReportMapInfo* sub procedure to see the message box return new information.

☐ Save changes to ex07.mxd.

EXERCISE END

Answers to Exercise 7B questions

Question 1: How many interfaces are supported by the Application class?

Answer: 11

Question 2: Examine the methods and properties displayed in the code completion list. Using the ArcMap OMD, can you determine which interface is the Application variable pointing to?

Answer: IApplication.

Question 3: Use your OMD to examine all the properties on the IApplication interface. Which property would you use to change the title bar text? How many properties are read-only? How many are you able to set (write)? The Name property, at first glance, might seem the logical property for changing the title bar text. This is a read-only property, however, and the application name will always be ArcMap (in this case). Look at the properties you can change.

Answer: You would use the Caption property to change the title bar text. There are 6 properties that are read-only. You can set 3 (Caption, CurrentTool, Visible).

Question 4: Application has a writable property for changing the title bar text. What is it? What data type is used to set this property?

Answer: The property is Caption, which uses a string data type.

Question 5: Before writing any code, examine the ArcMap object model diagram. Look at the Application class. Is there a method or property that returns the current document? What is it? What interface pointer is returned by this read-only property?

Answer: The Document property on the IApplication interface returns the current document. IDocument is returned by the document property.

Question 6: Return to the help and find IDocument. How many properties are listed on the IDocument interface? Use the same technique to examine IMxDocument What methods or properties does IMxDocument have for returning the document's map(s) (data frames)?

Answer: 7 properties are found on the IDocument interface. The "maps" and "focusmap" properties can return the document's data frames.

Question 7: What is the term for referencing another interface on the same object?

Answer: QueryInterface

Question 8: What is the only CoClass that implements IMap?

Answer: The only CoClass that implements IMap is the Map Class.

Question 9: Which OMD contains the IMap interface?

Answer: By switching to the Contents Tab and finding the OMD, the IMap interface and Map class is contained on the Carto Library Object Model Diagram.

Question 10: If you wanted to use the LayerCount property from the Map class, what is the return type?

Answer: The return type for the LayerCount property is Long. The OMD indicates this by stating "LayerCount: Long". When you declare a variable to hold the LayerCount value, it should be of type Long.

SOLUTIONS

STEP 1: WRITE CODE TO CHANGE THE APPLICATION TITLE

```
'**Step 4
Private Sub UIEditBoxControl1_Change()
    Application.Caption = UIEditBoxControl1.Text
End Sub
```

STEP 2: WRITE CODE TO WORK WITH THE DOCUMENT'S MAPS

```
'**Step 5
Public Sub ReportMapInfo()
    Dim pDocument As IDocument
    Set pDocument = Application.Document

    Dim pMxDocument As IMxDocument
    Set pMxDocument = pDocument

    Dim pMap As IMap
    Set pMap = pMxDocument.FocusMap

    Dim strDocName As String
    Dim strFocusMap As String
    Dim intLayerCount As Long

    strDocName = pDocument.Title
    strFocusMap = pMap.Name
    intLayerCount = pMap.LayerCount

    MsgBox "Doc Name: " & strDocName & vbCrLf & "Active Frame: " & strFocusMap &
vbCrLf & "Layers in the active map: " & intLayerCount
End Sub
```

8

Maps and layers

contents

EXERCISE 8: WORKING WITH MAPS AND LAYERS

This exercise has been broken into five steps. Steps 3, 4, and 5 are optional. You may do one (or more) of the optional steps along with Steps 1 and 2.

In Steps 1 and 2, you will write code that accesses all the maps (data frames) in the current document. The names of all the maps found in the document will be added to a combobox. You will then loop through a selected map to access all of its layers and add those to a combobox.

EXERCISE SHORTCUT

1. Open *ex08.mxd* and launch the *Visual Basic Editor*. Populate the *cboMaps* control with the name of the data frames in the document.
2. When a Map is selected in the *cboMaps* control, populate the *lboMapLayers* control with the names of the layers contained within the selected Map.
3. (Optional) When a layer is selected from the *lboMapLayers*, populate the *lboFields* control with the names of the fields found in the selected layers.
4. (Optional) Create a sub procedure named *ReportLayerType* in the *ThisDocument* module. Write code that determines if a layer is selected, and if selected, determine what type of layer it is.
5. (Optional) Create a sub procedure named *AddLayer* in the *ThisDocument* module. Write code that adds a new featurelayer to the focus map. Copy a datasource (featureclass) from another layer contained in the Document, to the new featurelayer.

STEP 1: ACCESS THE DOCUMENT'S MAPS

As an ArcObject programmer, there are a couple of ways in which you can access maps in a document. Remember that what you call a *map* in ArcObjects, a user from the interface would call a *data frame*. Maps are nothing more than a collection of layers. Each map in the same document can have a different set of layers, can represent various extents, and can even have different spatial references (projections, map units, etc.).

☐ Start *ArcMap*. Open *ex08.mxd* from your student directory.

☐ Launch the *Visual Basic Editor*.

The map contains a form that has been created for you: *frmDocInfo*. This form will display information about the current document such as the number of maps in the document, the number of layers in each map, and the number of fields in each layer table.

☐ Locate and open the form *frmDocInfo* from the *Project Explorer* window.

The design of the form is complete; only a minimal amount of code has been written for you.

Examine the objects found on the *frmDocInfo* form. There is a combobox and a listbox found on the form. The combo box will be used to display the names of the data frames (maps) found in the MxDocument. The listbox is used to display the list of layers within a data frame that is selected in the combo box.

☐ Double-click the form to access its code module.

Your first step will be to access the collection of maps found in the MxDocument, when the form initializes and populate the combo box.

Question 1: What interface provides the Maps (being the collection of maps) property? ____
Hint: Use the *ArcObjects Developer Help*. Click the *Index* tab and find the property *Maps*. You will then see an *Applies To* hyperlink.

☐ At the top of the module (General Declarations), dimension a variable to the interface that provides the Maps property.

```
Private m_pMxDoc As IMxDocument
```

☐ Return to the Developer Help or use the ArcMapUI Object Model diagram and determine the return type of the Maps property.

☐ Dimension a module level variable to the return type of the Maps property.

```
Private m_pMaps As IMaps 'IMaps is a collection of maps!
```

Question 2: What is the scope of the variables above? (procedural/modular/global)

____

☐ Navigate to the `UserForm_Initialize` event procedure. This is where you will initialize the variables you declared above.

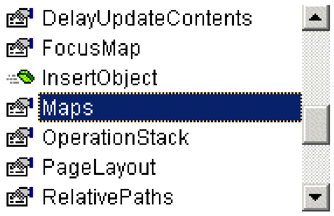
☐ Set the `m_pMxDoc` variable equal to the current document.

```
Set m_pMxDoc = ThisDocument 'QI
```

> *NOTE:* You just performed a QueryInterface and now have two variables (`ThisDocument` and `m_pMxDoc`) pointing to the same object (MxDocument) through different interfaces.

☐ Set the `m_pMaps` variable equal to a property on `m_pMxDoc` that returns the collection of maps in the MxDocument.

```
Set m_pMaps = m_pMxDoc.
```



☐ You will loop through the collection of maps stored in `m_pMaps` by completing the code below. Inside the loop, you will write the name of each map to a ComboBox control on the form (*cboMaps*).

```
Dim i As Integer
For i = 0 To m_pMaps._________ - 1
  cboMaps.AddItem m_pMaps._________(i).Name
Next ___
```

☐ Finally, you will write the name of the active data frame to a label on the form by completing the following code:

```
lblFocusMap.Caption = "Active Data Frame: " & m_pMxDoc._____________.Name
```

☐ Test your code by running the form.

When your form launches, you should see the name of the active data frame displayed in the label at the top. You should be able to pull down the ComboBox on the form to view the names of all three maps (data frames). In the next step, you will write similar code to list all the layers for a map selected in the ComboBox.

STEP 2: ACCESS THE LAYERS IN A MAP

In this step, you will write code to find a map selected in the ComboBox (*cboMaps*) and write the names of each layer in the selected map to a ListBox on the form (*lboMapLayers*).

☐ Declare the module-level variables below in the form module's General Declaration section to contain the selected map, an enum of all the map's layers, and an individual layer.

```
Private m_pMap As IMap 'Pointer to a map in the maps collection
Private m_pEnumLayers As IEnumLayer 'Enumeration of layers in a map
Private m_pLayer As ILayer 'Pointer to a layer in a map
```

Question 3: Using the Developer Help, what property found on the IMaps interface returns a pointer to an object that supports IMap? ___________________________
Hint: In the Developer Help, type IMap under the *Index* tab and click on the links for the members to examine the return types for properties belonging to IMap.

Question 4: Using the Developer Help, what property on the IMap interface returns a reference to an object that supports IEnumLayer?__________________________

Question 5: Using the Developer Help, what property on the IEnumLayer interface returns a reference to an object that supports ILayer? ________________________

☐ Navigate to the *Change* event procedure for *cboMaps* (`cboMaps_Change`).

When the value in this ComboBox changes, the names of the selected map's layers will be updated in the ListBox.

When accessing a map from the Maps collection, there is no way to ask for a map by name. The only way to return a map from the collection is to use its index number (the order of the map in the collection). If you want to return a particular map according to its name, you can use an *If Then* statement inside of a looping structure.

☐ Begin a For Next loop identical to the one you wrote in the `UserForm_Initialize` procedure (Hint: Do not forget to declare a loop index variable).

You will loop through all maps in the document until you find the one whose name matches the one selected in the ComboBox.

☐ Inside the loop, provide the logic to find the chosen map by completing the code below.

```
Set m_pMap = m_pMaps.Item(__) 'Pull a map from the collection
If m_pMap._______ = cboMaps.Text Then 'Is this the chosen map?
  Exit For 'Exit For will exit the loop once the map is found
End If
Next ____
```

Now that you have found the specified map with the loop above, you will write another loop to write the name of each layer in the map to a ListBox.

☐ Initialize the `m_pEnumLayers` variable with all the layers in the chosen map by completing the code below.

```
Set m_pEnumLayers = m_pMap.________________'Not just one layer!
```

☐ Complete the code below to initialize the `m_pLayer` variable by pulling the first layer out of the `m_pEnumLayers` enum.

```
Set m_pLayer = m_pEnumLayers.________________
```

Remember that there are two kinds of looping structures. The For Next loops you wrote above are used when you know how many times to loop; the loop you write below will *not* be a For Next loop, but will loop according to a condition instead.

☐ Complete the loop below to write the name of each layer to the *lboMapLayers* ListBox.

```
Do _______ m_pLayer Is Nothing
  lboMapLayers.AddItem m_pLayer.Name
  Set m_pLayer = ____________.Next
Loop
```

☐ Run the form to test your code.

Question 6: Choose a map from the ComboBox. Does the layer list display the chosen map's layers? ____

Question 7: Choose another map from the ComboBox. What happens in the ListBox? ____
____

☐ Add the code below to the top of the `cboMaps_Change` event procedure to remove any existing text items from the *lboMapLayers* ListBox before adding new ones.

```
lboMapLayers.Clear
```

☐ Test your code again. Choose several maps to make sure the ListBox is cleared properly.

You now know how to loop through each map in the document, as well as through each layer in a map.

You now have three optional steps you may choose from. Optional Step 3 has you write code to access a selected layer's attribute table and all of its fields. Optional Step four uses the Visual Basic keywords; `TypeOf` and `Nothing` to test for object reference and optional Step 5 shows you how to add a new featurelayer to a map.

STEP 3: ACCESS THE ATTRIBUTES OF A FEATURE LAYER (OPTIONAL)

In this optional step, you will write code to fill the form's *lboFields* ListBox with the names of all fields in the selected layer's attribute table.

☐ Navigate to the *Click* event procedure for the *lboMapLayers* ListBox (*lboMapLayers_Click*).

☐ First, write code to clear the contents of the *lboFields* ListBox, as shown below.

```
lboFields.Clear
```

Similar to the code you wrote earlier, you will write a looping routine to find the layer chosen in the *lboMapLayers* ListBox.

☐ Complete the loop below to locate the selected layer. Remember: because most variables used earlier were declared as module-level, you can use them here.

```
m_pEnumLayers._________ 'Move the enum pointer above the first layer!
Set m_pLayer = m_pEnumLayers.________ 'Get the first layer from the enum

___ _________ m_pLayer ___ Nothing 'Begin the (conditional) loop
  If m_pLayer.Name = lboMapLayers.Text Then 'Check to see if this is the layer
    Exit ___ 'If we have the layer, exit the loop (not the sub)!
  End If
  Set m_pLayer = ____________._______ 'Get another layer
Loop 'end the loop
```

After the loop above has finished, `m_pLayer` will be set to the layer selected in the ListBox. The next step will be to see if it has an attribute table (only FeatureLayers have an attribute table, called a FeatureClass in ArcObjects).

☐ Complete the code below to see if the selected layer (`m_pLayer`) is a FeatureLayer. If it is not a FeatureLayer, you will write the text 'Not a Feature Layer' in the *lboFields* ListBox.

```
If Not __________ m_pLayer Is IFeatureLayer ________ 'Does it support this interface?
  lboFields._____________ "Not a Feature Layer"
  Exit Sub 'Stop execution of the procedure
End If
```

Because the loop above exits the sub if the chosen layer does not support the IFeatureLayer interface, the rest of your code can assume that a FeatureLayer was chosen in the ListBox.

☐ Declare the variable below to point to the IFeatureLayer interface on the layer (because you now know that it supports it).

```
Dim pFLayer As IFeatureLayer
```

☐ Initialize the `pFLayer` variable by using QueryInterface on `m_pLayer`, as shown below.

```
Set pFLayer = m_pLayer
```

☐ Declare the variables below—one to contain a FeatureClass and one to contain a collection of Fields.

```
Dim pClass As IFeatureClass 'Analogous to the attribute table
Dim pFields As IFields 'A collection of fields (don't forget the "s"!)
```

Question 8: FeatureClass is a property on the IFeatureLayer interface. Which variable will you use to access the layer's FeatureClass? ____________________

☐ Initialize some of the new variables by completing the code below.

```
Set pFClass = ________________.FeatureClass
Set pFields = pFClass.________________ 'Not just one!
```

Because Fields is a collection, you will access individual items by using an index number (the position within the collection).

☐ Complete the loop below to write each field's name to the *lboFields* ListBox.

```
Dim i As ______________
For i = ___ To ___________.___________ - 1
  lboFields.Additem pFields.________ (__).Name
_____ i
```

Question 9: Run the form to test your code. Select a map, then a layer. Are the layer's fields displayed? Select the Air Photo layer in the Maplewood map. What is displayed in the Fields ListBox?____

STEP 4: WRITE CODE TO TEST AN OBJECT REFERENCE (OPTIONAL)

In this optional step, you will write code to see what type of layer is selected in the document's Table of Contents. You will use the *TypeOf* structure to see which interfaces an object variable supports.

☐ Return to the *Visual Basic Editor* and reopen the project's *ThisDocument* code module.

☐ Define a public subprocedure called **`ReportLayerType`**

☐ Dimension the following variables, one to hold a reference to the MxDocument's IMxDocument interface and one to point to the ILayer interface.

```
Dim pMxDocument As IMxDocument
Dim pLayer As ILayer
```

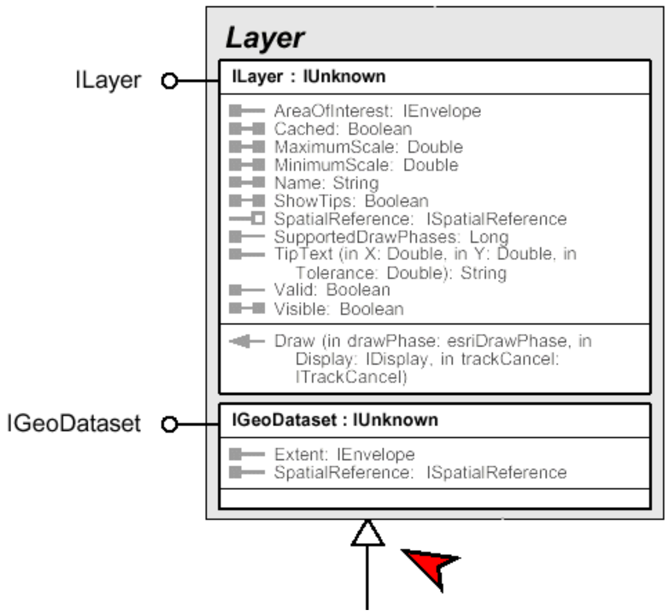
☐ Set the `pMxDocument` variable using one of the lines of code below (either will work).

```
Set pMxDocument = Application.Document
'--OR--
Set pMxDocument = ThisDocument
```

`ThisDocument` is a reserved variable used by ArcMap that points to the IDocument interface of the current document. This variable is available as soon as you start ArcMap, as is the `Application` variable.

☐ Initialize your `pLayer` variable. Set it equal to a property on IMxDocument that returns the currently selected layer (ILayer).

Although there are several types of layers a user could potentially have selected in his or her map, all ArcMap layers will support the ILayer interface. Therefore, the SelectedLayer property on IMxDocument returns a pointer to the ILayer interface of whichever type of layer is selected.


Layer
ILayer
ILayer : IUnknown
AreaOfInterest: IEnvelope
Cached: Boolean
MaximumScale: Double
MinimumScale: Double
Name: String
ShowTips: Boolean
SpatialReference: ISpatialReference
SupportedDrawPhases: Long
TipText (in X: Double, in Y: Double, in Tolerance: Double): String
Valid: Boolean
Visible: Boolean
Draw (in drawPhase: esriDrawPhase, in Display: IDisplay, in trackCancel: ITrackCancel)
IGeoDataset
IGeoDataset : IUnknown
Extent: IEnvelope
SpatialReference: ISpatialReference

☐ Using your OMD, identify the interfaces unique to each subtype of layer, then complete the code below to determine what kind of layer is currently selected in the document.

```
If TypeOf pLayer Is IRasterLayer Then
  MsgBox "The selected layer is a __________ Layer"
ElseIf TypeOf pLayer Is ____________________ Then
  MsgBox "The selected layer is a Feature Layer"
ElseIf TypeOf pLayer Is ________________ Then
  MsgBox "The selected layer is a Coverage Annotation Layer"
ElseIf TypeOf pLayer Is ________________ Then
  MsgBox "The selected layer is a Group Layer"
End If
```

☐ Close the *Visual Basic Editor* and return to *ArcMap*. Select a layer in the Table of Contents, then test your procedure.

Question 10: What happens if you run the procedure without a selected layer?

____

How could you prevent this error? ____

____

STEP 5: ADD A NEW LAYER TO THE MAP (OPTIONAL)

In this optional step, you will write a simple procedure that adds a new layer to the focused map. Accessing data on disk will be covered in a later lesson, so for now you will simply add a layer that is based on the same data source as an existing layer in the map.

☐ Close the *frmDocInfo* form designer and code module.

☐ Open the *ThisDocument* code module under the project (*ex08.mxd*).

☐ Define a new sub procedure, as shown below.

```
Public Sub AddLayer( )
```

☐ Begin your procedure by declaring some variables that you will need.

```
Dim pMxDoc As IMxDocument
Dim pNewFLayer As IFeatureLayer
```

☐ Initialize the `pMxDoc` variable by setting it equal to the current document.

☐ Initialize the `pNewFLayer` variable by setting it equal to a new FeatureLayer.

☐ Assign the text **`New Layer`** as pNewFLayer's Name property.

☐ Complete the chain of code below to add the layer to the map.

```
pMxDoc.FocusMap.________________ pNewFLayer
```

Question 11: Bring ArcMap to the front of your display and run the *AddLayer* procedure from the *Tools* > *Macros* > *Macros* menu choice. Do you see your new layer added to the map? ____

Adding a layer to a map is quite simple. Specifying a data source for the new layer is the tricky part. Again, accessing data on disk will be covered later in this course. For now, you will simply use the data source of an existing layer in the map (essentially making a copy of the layer).

☐ Return to the *AddLayer* procedure.

☐ Dimension the following variable immediately below the line that initializes `pNewFLayer` equal to a new FeatureLayer.

```
Dim pCopyFLayer As IFeatureLayer
```

☐ Complete the code below to (1) test if there is a selected layer in the document and (2) test if the selected layer is a FeatureLayer.

```
If ________.SelectedLayer Is ____________ Then
  MsgBox "No layer selected"
  Exit Sub
ElseIf Not TypeOf pMxDoc.______________ Is ________________ Then
  MsgBox "Selected layer is not a feature layer"
  Exit Sub
End If
```

After the code above has executed, you can safely assume that the selected layer is a FeatureLayer.

☐ Set `pCopyFLayer` equal to the document's selected layer.

☐ Use your object model diagram to choose one of the lines below. Complete the appropriate line to set the new layer's (`pNewFLayer`) data source equal to that of the selected layer's (`pCopyFLayer`).

```
pNewFLayer.FeatureClass = pCopyFLayer.FeatureClass
'*** OR ****
Set pNewFLayer.FeatureClass = pCopyFLayer.FeatureClass
```

Question 12: What is the difference between these two lines? _________________

Question 13: Why did you choose the one you did? _________________________

☐ Modify the existing line of code that sets the name of `pNewFLayer` to match the one below.

```
pNewFLayer.Name = "Copy of " & pCopyFLayer.Name
```

Question 14: Bring ArcMap to the front of your display. Make sure a FeatureLayer is selected, then run the `AddLayer` macro once again. Is a copy of the selected layer added to the map? ______________________________________ What is its name?
____

Question 15: Finally, test your code by running it without a selected layer, then again with a non-FeatureLayer layer selected. Does it handle these situations gracefully? ____

EXERCISE END

ANSWERS TO EXERCISE 8 QUESTIONS

Question 1: What interface provides the Maps (being the collection of maps) property?

Answer: The IMxDocument interfaces provides the Maps property.

Question 2: What is the scope of the variables above? (procedural/modular/global)

Answer: Modular, the Private keyword helps you distinguish that a variable is modular-level.

Question 3: Using the Developer Help, what property found on the IMaps interface returns a pointer to an object that supports IMap?

Answer: The property Item will return a reference to an object that implements IMap. Therefore to access the first map in the maps collection, the code may look similar to the following:

```
Dim pMap as IMap
Set pMap = pMaps.Item(0)
```

Question 4: Using the Developer Help, what property on the IMap interface returns a reference to an object that supports IEnumLayer?

Answer: The property Layers will return a reference to an object that supports IEnumLayers. Therefore to access the enumeration of layer in a dataframe, the code may look similar to the following:

```
Dim pLayers as IEnumLayer
Set pLayers = pMap.Layers
```

Question 5: Using the Developer Help, what property on the IEnumLayer interface returns a reference to an object that supports ILayer?

Answer: The property Next will return a pointer to an object that implements ILayer. Therefore to access the first layer in an enumeration, the code may look similar to the following:

```
Dim pLayer as ILayer
Set pLayer = pLayers.Next
```

Question 6: Choose a map from the ComboBox. Does the layer list display the chosen map's layers?

Answer: Yes. If it does not, check the code you wrote for errors.

Question 7: Choose another map from the ComboBox. What happens in the ListBox?

Answer: It adds the data frames layers, however, it also keeps the information from the layers of the previous data frames.

Question 8: FeatureClass is a property on the IFeatureLayer interface. Which variable will you use to access the layer's FeatureClass?

Answer: pFClass

Question 9: Run the form to test your code. Select a map, then a layer. Are the layer's fields displayed? Select the Air Photo layer in the Maplewood map. What is displayed in the Fields ListBox?

Answer: Yes, the fields are displayed. If the Air Photo is selected, you should see the text "Not a feature layer" appear in the listbox.

Question 10: What happens if you run the procedure without a selected layer? How could you prevent this error?

Answer: You will receive an error stating that the "object variable or with block variable is not set". You could prevent this error by checking to see if the selected layer is nothing, then popup a message box instructing the user to select a layer first. You could also explicitly search for a particular layer by name rather than rely on the user to select a layer.

Question 11: Bring ArcMap to the front of your display and run the AddLayer procedure from the Tools > Macros > Macros menu choice. Do you see your new layer added to the map?

Answer: A new layer is added to the Table of Contents, but the data source is not set.

Question 12: What is the difference between these two lines?

Answer: The use of the keyword set is the difference in the two lines. Set refers to a property put by reference and will hold a dynamic reference to an object.

Question 13: Why did you choose the one you did?

Answer: You should have chosen the use of Set. If you look at the object model diagram, you will notice a hollow box referencing this property. Whenever you see the hollow property box, you should use the "Set" keyword.

Question 14: Bring ArcMap to the front of your display. Make sure a FeatureLayer is selected, then run the `AddLayer` macro once again. Is a copy of the selected layer added to the map? What is its name?

Answer: Yes, a copy is added. The name will be "Copy of", followed by the name of the layer you currently have selected in the Table of Contents.

Question 15: Finally, test your code by running it without a selected layer, then again with a non-FeatureLayer layer selected. Does it handle these situations gracefully?

Answer: Yes, a message box should appear telling you that no layer has been selected.

SOLUTIONS

```
'frmDocInfo
Private m_pMxDoc As IMxDocument
Private m_pMaps As IMaps
Private m_pMap As IMap
Private m_pEnumLayers As IEnumLayer
Private m_pLayer As ILayer

Private Sub cboMaps_Change()
    cmdFrameInfo.Enabled = cboMaps.Text <> ""
    lboMapLayers.Clear

    Dim i As Integer
    For i = 0 To m_pMaps.Count - 1
        Set m_pMap = m_pMaps.Item(i)
        If m_pMap.Name = cboMaps.Text Then
            Exit For
        End If
    Next i
    Set m_pEnumLayers = m_pMap.Layers
    Set m_pLayer = m_pEnumLayers.Next
    Do Until m_pLayer Is Nothing
        lboMapLayers.AddItem m_pLayer.Name
        Set m_pLayer = m_pEnumLayers.Next
    Loop
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub UserForm_Initialize()
    Set m_pMxDoc = ThisDocument
    Set m_pMaps = m_pMxDoc.Maps
    Dim i As Integer
    For i = 0 To m_pMaps.Count - 1
        cboMaps.AddItem m_pMaps.Item(i).Name
    Next i
    lblFocusMap.Caption = "Active Data Frame: " & m_pMxDoc.FocusMap.Name
End Sub
```

'**OPTIONAL STEP THREE**

```
Private Sub lboMapLayers_Click()

    lboFields.Clear
    cmdLayerInfo.Enabled = lboMapLayers.Text <> ""
    m_pEnumLayers.Reset
    Set m_pLayer = m_pEnumLayers.Next

    Do Until m_pLayer Is Nothing
        If m_pLayer.Name = lboMapLayers.Text Then
            Exit Do
        End If
        Set m_pLayer = m_pEnumLayers.Next
    Loop
    If Not TypeOf m_pLayer Is IFeatureLayer Then
        lboFields.AddItem "Not a Feature Layer"
        Exit Sub
    End If
    Dim pFLayer As IFeatureLayer
    Set pFLayer = m_pLayer

    Dim pFClass As IFeatureClass
    Dim pFields As IFields

    Set pFClass = pFLayer.FeatureClass
    Set pFields = pFClass.Fields

    Dim i As Integer
    For i = 0 To pFields.FieldCount - 1
        lboFields.AddItem pFields.Field(i).Name
    Next i

End Sub
'************************
```

OPTIONAL STEP FOUR (THISDOCUMENT MODULE - EX08.MXD)

```
'**Optional Step Four**
Public Sub ReportLayerType()
    Dim pMxDocument As IMxDocument
    Dim pLayer As ILayer

    Set pMxDocument = ThisDocument
    'Set pMxDocument = Application.Document

    Set pLayer = pMxDocument.SelectedLayer
    If TypeOf pLayer Is IRasterLayer Then
        MsgBox "The selected layer is a raster layer"
    ElseIf TypeOf pLayer Is IFeatureLayer Then
        MsgBox "The selected layer is a feature layer"
    ElseIf TypeOf pLayer Is ICoverageAnnotationLayer Then
        MsgBox "The selected layer is a Coverage Annotation Layer"
    ElseIf TypeOf pLayer Is IGroupLayer Then
        MsgBox "The selected layer is a Group layer"
    End If
End Sub
```

OPTIONAL STEP FIVE (THISDOCUMENT MODULE - EX08.MXD)

```
'**Optional Step Five**
Public Sub AddLayer()
    Dim pMxDoc As IMxDocument
    Dim pNewFLayer As IFeatureLayer

    Set pMxDoc = ThisDocument
    Set pNewFLayer = New FeatureLayer

    Dim pCopyFLayer As IFeatureLayer
    If pMxDoc.SelectedLayer Is Nothing Then
        MsgBox "No layer selected"
        Exit Sub
    ElseIf Not TypeOf pMxDoc.SelectedLayer Is IFeatureLayer Then
        MsgBox "Selected layer is not a feature layer"
        Exit Sub
    End If
    Set pCopyFLayer = pMxDoc.SelectedLayer
    Set pNewFLayer.FeatureClass = pCopyFLayer.FeatureClass
    pNewFLayer.Name = "Copy of " & pCopyFLayer.Name
    pMxDoc.FocusMap.AddLayer pNewFLayer

End Sub
```

9

Data creation

contents

EXERCISE 9A: CREATE A LAYER FROM A SHAPEFILE

In this exercise, you will add a new FeatureLayer to an ArcMap document. You will access a shapefile data source on disk, then associate this data with the new layer.

EXERCISE SHORTCUT

1. Gather information from the Developer Help and Object Model diagrams to determine how to create a new featurelayer and add the new featurelayer to your focus map
2. Write the code to reference the active data frame.
3. Obtain a reference to the *...\IPAO\Data\World* workspace to access the shapefiles contained in this workspace.
4. Obtain a reference to the *Faults.shp* shapefile.
5. Assign the *Faults.shp* featureclass to the FeatureLayer. Assign a name to the layer and set the *ShowTips* property to True.
6. Assign the FeatureLayer to the active data frame.

STEP 1: GATHER INFORMATION ON HOW TO CREATE A NEW LAYER

In this step, you will gather information from the Developer Help and Object Model diagrams to determine how to:

- Create a new FeatureLayer.
- Add the FeatureLayer to your focus map.

☐ To create a new FeatureLayer, use the Developer Help, the Carto Layer Object Model diagram, or both, and examine the FeatureLayer class.

Question 1: The FeatureLayer is what type of class? CoClass/Class/Abstract.

Question 2: What Visual Basic keyword is used when creating a new instance of a CoClass? ____

Question 3: What interface on the FeatureLayer class will let you assign a name to your FeatureLayer object? ________________________________

Question 4: What are the two lines of code to dimension and instantiate a Featurelayer object?

____

____

Hint: There are three basic steps to instantiating an COM object. Step One: decide what interface you want to use (you want the interface that provides the Name property). Step Two: dimension a variable to that interface. Step Three: instantiate the object, setting the variable equal to a new instance of the class.

You have now determined how to create a new FeatureLayer. The next step is to understand how to add the FeatureLayer to your focus map.

☐ Using the Developer Help, examine the Map Class for a method that lets you add a layer to the map.

Question 5: What interface supported by the Map class provides a method to add a layer to the map?____

☐ Examine the relationship between the MxDocument and the Map class, using the Carto Object Model diagram.

Question 6: What is the relationship between the MxDocument and the Map class?

____

By examining the Object Model diagram, you can see that the MxDocument is composed of multiple Maps. Therefore, to reference a map, in order to add a layer, your first step is to reference the MxDocument, then through the MxDocument, you may reference the desired map, such as the focus map.

Question 7: Using the Developer Help, what interface found on the MxDocument class provides the FocusMap property? ________________________________

Question 8: Write the two lines of code to do a QueryInterface from the IDocument interface to the IMxDocument interface.

____

____

Hint: You already have a variable the communicates with the IDocument interface.

STEP 2: WRITE CODE TO REFERENCE THE ACTIVE DATA FRAME

Your code will add a new shapefile FeatureLayer to the focus map. In this step, you will open a new map document, launch the Visual Basic Editor, and add a new procedure in a new standard code module.

☐ Start *ArcMap*.

☐ Open a new (empty) map document.

☐ Launch the ArcMap *Visual Basic Editor*.

☐ In the Project, insert a new standard *module*.

☐ Create a new Public Sub procedure in the module called *AddShape*.

You will begin your code by referencing the active data frame (focused map).

☐ Declare a procedural level variable, **`pMxDoc`**, to the interface that provides the FocusMap property.

☐ QueryInterface using the `ThisDocument` variable as your pointer to the MxDocument object.

☐ Declare a procedural level variable, **`pMap`**, to the return type of the FocusMap property.

Hint: To determine the return type of the FocusMap property, examine the Focus-Map property in the Developer Help.

☐ Set the `pMap` variable to the MxDocument's focus map.

Your code should look similar to the following:

```
Dim pMxDoc as IMxDocument
Set pMxDoc = ThisDocument 'QI

Dim pMap as IMap
Set pMap = pMxDoc.FocusMap
```

In the following steps, you will write code to reference a shapefile to display in a new layer to the active data frame.

STEP 3: GET A SHAPEFILE WORKSPACE

In ArcGIS, a Workspace refers to *any* folder or database on disk that contains data. Your starting point for accessing a Workspace through code is to create a WorkspaceFactory. A WorkspaceFactory has to be created as a specific type (Shapefile, ArcInfo, ArcSDE, etc.), depending on the type of data you want to access or create. If you had a folder that contains both shapefiles and coverages, for example, you would need to create a ShapeFileWorkspace and an ArcInfoWorkspace to access both types of data from the same location on disk.

☐ Using the Developer Help, examine the ShapefileWorkspaceFactory CoClass and the interfaces supported by the class.

Question 9: Is there an interface supported by the ShapefileWorkspaceFactory that has a method that returns a Workspace from file? ___________________________

☐ Declare a procedural level variable, **`pWSFactory`**, to an interface that supports the OpenFromFile method.

☐ Instantiate the **`pWSFactory`** variable to a new ShapefileWorkspaceFactory Object.

☐ Declare a variable, **`pWorkspace`**, to the return type of the OpenFromFile method.

Your code should look similar to the following:

```
Dim pWSFactory as IWorkspaceFactory
Set pWSFactory = New ShapefileWorkspaceFactory

Dim pWorkspace as IWorkspace
```

☐ Set the `pWorkspace` variable by writing the code below.

```
Set pWorkspace = pWSFactory.OpenFromFile _
("C:\Student\IPAO\Data\World", Application.hWnd)
```

NOTE: The second parameter for OpenFromFile is a Window Handle, or hWnd object (pronounced *Aych-Wend*). An hWnd is an integer that references an application's window. Use the hWnd to *tie* other dialogs or data to a particular application.

STEP 4: GET A FEATURECLASS FROM THE WORKSPACE

Now that you have successfully referenced a folder on disk that contains shapefile data, you will access a specific shapefile to add as a layer to the map.

☐ Locate the Workspace class on the Geodatabase object model diagram.

Question 10: Your variable (`pWorkspace`) is pointing to the IWorkspace interface. Do you see any methods or properties on this interface for accessing or creating data?

____

Question 11: Scan methods and properties listed on other interfaces supported by the Workspace class. Which interface can be used to create or access data?

____

The IFeatureWorkspace interface contains methods for accessing existing data in a Workspace such as OpenFeatureDataset, OpenFeatureClass, and OpenTable. It also contains methods for creating new datasets such as CreateFeatureDataset, CreateFeatureClass, and CreateTable.

☐ Declare a new variable called **`pFWorkspace`** as a pointer to the IFeatureWorkspace interface.

☐ Set `pFWorkspace` equal to `pWorkspace` (this is QueryInterface).

☐ Declare the following variable to reference a shapefile FeatureClass.

```
Dim pFClass As IFeatureClass
```

☐ Write the code below to access the worldwide geologic faults dataset and store it in the pFClass variable.

```
Set pFClass = pFWorkspace.OpenFeatureClass("Faults.shp")
```

At this point, you have referenced the desired dataset (shapefile) but have yet to associate it with the new FeatureLayer. In the next step, you will associate the FeatureClass with the FeatureLayer, then add it to the map.

STEP 5: DEFINE THE FEATURELAYER'S DATA SOURCE, SET SOME OF ITS PROPERTIES

Remember that a layer is nothing more than a pointer to a data source on disk. Layers also have properties to control how the data is displayed (symbology, labels, map tips, etc.). In this step, you will set your FeatureLayer's FeatureClass property (which defines the data source), as well as some of its basic properties.

☐ Look at the IFeatureLayer interface on the Carto object model diagram (on the FeatureBaseLayer class).

☐ Declare a variable called **`pFLayer`** as a pointer to the IFeatureLayer interface.

You may notice that there is also an IFeatureLayer2 interface on the OMD. You could use IFeatureLayer2 instead if you wanted. Sometimes these interfaces with numeric endings represent newer versions with additional methods or properties. Sometimes they are identical to the interface without a numerical ending, but are provided to allow you to create custom components (you own custom feature layer) as discussed in the Extending ArcGIS Desktop Applications course. As a COM rule, an interface will never die, so you can always rely on it to work in future versions.

☐ Set `pFLayer` equal to a New FeatureLayer.

☐ Based on what you see in the Developer Help for IFeatureLayer, choose the correct line of code below to assign the faults shapefile to the layer.

```
' Write ONE of the lines of code below ...
pFLayer.FeatureClass = pFClass
'**- OR -**
Set pFLayer.FeatureClass = pFClass
```

Question 12: Why did you choose the one you did? ________________________
____

☐ Also assign values for the following layer properties (pFLayer):

```
Name = "Geologic Faults"
ShowTips = True
```

STEP 6: ADD THE FAULTS LAYER TO THE ACTIVE DATA FRAME

Finally, you will add the geologic faults layer to the document's active data frame (focused map).

You will use the AddLayer method on IMap to add a new FeatureLayer. The one parameter for AddLayer is (of course) a layer to add. Notice that the object model diagram shows Layer as an abstract class; only the CoClass subtypes of Layer can be created with the `New` keyword. You will create a new FeatureLayer.

☐ Use the AddLayer method on the focus map to add the layer.

```
pMap.AddLayer pFLayer
```

☐ In *ArcMap*, from the *Tools* menu, run your procedure as a *macro*.

You should see the new layer added to your map. The layer should be called Geologic Faults. If you hover your mouse over features in the layer, map tips should be displayed. You will learn to control layer symbology and classification in a later lesson.

EXERCISE END

ANSWERS TO EXERCISE 9A QUESTIONS

Question 1: The Featurelayer is what type of class? CoClass/Class/Abstract.

Answer: The Featurelayer is a CoClass. It states this in the Developer Help and is shown on the Carto Layer Object Model diagram as 3D and shaded. All CoClasses on the OMDs will be 3D and shaded.

Question 2: What Visual Basic keyword is used when creating a new instance of a CoClass?

Answer: NEW is the keyword that lets Visual Basic know you want to create a new instance of a COM CoClass.

Question 3: What interface on the FeatureLayer class will let you assign a name to your FeatureLayer object?

Answer: Both the ILayer and IFeatureLayer interfaces provide a Name property. As you will see in later lessons, the Name property belongs to the ILayer interface. However you have access to the Name property from the IFeatureLayer interface because it inherits the members of ILayer.

Question 4: What are the two lines of code to dimension and instantiate a Featurelayer object?

Answer:

```
Dim pFlayer as ILayer (or IFeatureLayer)
Set pFLayer = NEW FeatureLayer
```

Question 5: What interface supported by the Map class provides a method to add a layer to the map?

Answer: The IMap interface provides the AddLayer method, this method will add a layer to the map.

Question 6: What is the relationship between the MxDocument and the Map class?

Answer: As shown by the object model diagram, the MxDocument is composed of multiple maps. Therefore in order to reference a map, you must go through the MxDocument object.

Question 7: Using the Developer Help, what interface found on the MxDocument class provides the *FocusMap* property?

Answer: IMxDocument provides the FocusMap property. The FocusMap property will return a reference to the Map object that is in focus.

Question 8: Write the two lines of code to do a QueryInterface from the *IDocument* interface to the *IMxDocument* interface.

Answer:

```
Dim pMxDoc as IMxDocument
Set pMxDoc = ThisDocument 'QI
```

Question 9: Is there an interface supported by the ShapefileWorkspaceFactory that has a method that returns a Workspace from file?

Answer: The IWorkspaceFactory and IWorkspaceFactory both offer the OpenFromFile method. This method will return a reference to a workspace on file.

Question 10: Your variable (`pWorkspace`) is pointing to the IWorkspace interface. Do you see any methods or properties on this interface for accessing or creating data?

Answer: No. There are no methods for creating or accessing data on IWorkspace. However, there are plenty of methods on the workspace class for accessing and creating data on different interfaces.

Question 11: Scan methods and properties listed on other interfaces supported by the Workspace class. Which interface can be used to create or access data?

Answer: The IFeatureWorkspace interface has methods for creating or accessing data. Some of the more common methods are: CreateFeatureClass, CreateFeatureDataset, OpenFeatureClass, and OpenFeatureDataset.

Question 12: Why did you choose the one you did?

Answer: The line of code with the Set keyword should have been chosen. In the Developer Help, the FeatureClass property has a hollow box. This tells you that you are setting a property by reference and should always use the Set keyword. The Object Model diagram also portrays this symbology for IFeatureLayer2.

SOLUTIONS

```
'**** AddShapefile ***
Public Sub AddShape()
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument 'QI

    Dim pMap As IMap
    Set pMap = pMxDoc.FocusMap


    Dim pWSFactory As IWorkspaceFactory
    Set pWSFactory = New ShapefileWorkspaceFactory

    Dim pWorkspace As IWorkspace
    Set pWorkspace = pWSFactory.OpenFromFile("C:\Student\IPAO\Data\World", _
    Application.hWnd)

    Dim pFWorkspace As IFeatureWorkspace
    Set pFWorkspace = pWorkspace 'QI

    Dim pFClass As IFeatureClass
    Set pFClass = pFWorkspace.OpenFeatureClass("Faults.shp")

    Dim pFLayer As IFeatureLayer
    Set pFLayer = New FeatureLayer
    Set pFLayer.FeatureClass = pFClass

    pFLayer.Name = "Geologic Faults"
    pFLayer.ShowTips = True

    pMap.AddLayer pFLayer
End Sub
```

EXERCISE 9B: CREATING DATA

In this exercise, you will learn to create new workspaces and datasets. You will begin by creating a new personal geodatabase (Access.mdb). You will then use the Workspace object in your code to create a new table in the geodatabase. Your code will create a collection of fields for the table, add records (rows), and then provide values for each row. Although your code will work with a personal geodatabase, the basic procedure will be the same for any type of data you want to create.

EXERCISE SHORTCUT

1. In a new standard module in ArcCatalog, create a function creates a GeoDatabase. The function should pass in the parent folder and GeoDatabase name, and return a pointer to the workspace.
2. In the ThisDocument module, write a procedure that calls the function to create the workspace.
3. Create a table in the GeoDatabase with three fields: OID, Store, and Sales.
4. Create rows in your table through the CreateRow method, and assign values to the Store and Sales fields.

STEP 1: CREATE A WORKSPACEFACTORY OBJECT

Your starting point will be identical to the one you used to access an existing shapefile earlier: the WorkspaceFactory class. Remember that WorkspaceFactory is an abstract class, but it has several createable subclasses. In this step, you will create a new AccessWorkspaceFactory in order to create a new personal geodatabase (*.mdb).

☐ Start *ArcCatalog*. Open the ArcCatalog *Visual Basic Editor*.

☐ Insert a new standard *module*. Name it `CreateData`

☐ Define a new function procedure, as shown below.

```
Public Function CreateAccessGDB (ParentFolder As String, GDBName As String) As
IWorkspace
```

☐ Dimension the following variables inside the new procedure (using the Dim keyword).

```
pAccessWSF As IWorkspaceFactory
```

```
pWorkName As IWorkspaceName
pName As IName
pWorkspace As IWorkspace
```

☐ Set pAccessWSF equal to a new AccessWorkspaceFactory.

☐ Locate the IWorkspaceFactory interface on your Geodatabase object model diagram or in the object browser. Earlier, you used the OpenFromFile method to access an existing workspace.

Question 1: Which method or property can be used to make a *new* workspace?

____

NOTE: The *Create* method on IWorkspaceFactory does not explicitly return a Workspace, but rather a *WorkspaceName*. Name objects are lightweight versions of the things they represent and can be opened to produce the desired object.

You will use the Create method on IWorkspaceFactory to make a WorkspaceName object. You will then use the Open method on the Name to open the new Workspace.

☐ Write the code below to create the new WorkspaceName.

```
Set pWorkName = pAccessWSF.Create (ParentFolder, GDBName, Nothing, Application.hWnd)
```

The required arguments for the Create method shown above are the folder in which to create the new workspace, the name of the new workspace, connection properties (e.g., if connecting to an ArcSDE database), and the application window handle. The folder pathname and the database name are passed into the procedure. You will resume below. Nothing is specified in order to skip the connection properties argument, and the preset *Application* variable is used to retrieve the ArcCatalog window handle.

☐ Locate WorkspaceName on the object model diagram. Notice the IWorkspaceName interface provides access to only the most basic workspace properties (e.g., Category, Pathname, and Type).

Question 2: In order to retrieve the actual Workspace object, you will use the *Open* method supported by the WorkspaceName class. Using the Developer Help examining WorkspaceName, can you locate the interface that contains the Open method?____
Hint: If using the Object Model Diagrams, inheritance will lead you to another object model, thus using the Help is easier to examine this relationship.

Question 3: Locate the Name object on the object model diagram. What is the relationship of WorkspaceName to Name?_________________________
What does the Open method on IName return? ______________________

Because there are several subtypes of the Name class, there are a variety of objects that could be returned when the Open method is called. If you open a FeatureDatasetName, for example, a FeatureDataset would be returned. In your case, opening a WorkspaceName will return a Workspace. Because IName has no way of anticipating which class the Open method will be used on, it lists *IUnknown* as the return value. This simply means that you will get the default interface of the type of Name you opened (e.g., FeatureClassName, TableName, WorkspaceName).

> NOTE: IUnknown is an interface implemented on all COM classes, and although (as a Visual Basic programmer) you will never need to work *directly* with this interface, it is the interface that controls the life of the object and allows you to use QueryInterface.

☐ Set pName equal to pWorkName. This will give you access to the IName interface on your WorkspaceName object (QueryInterface).

```
Set pName = pWorkName
```

☐ Write the code below to Open the WorkspaceName and store the returned Workspace object in the pWorkspace variable.

```
Set pWorkspace = pName.Open
```

☐ Finally, return the new workspace from the CreateAccessGDB function, as shown below.

```
Set CreateAccessGDB = pWorkspace
```

You now have a function that can be used to create a personal geodatabase by simply specifying the output location and name for the new workspace. You will test this code later by calling it to make a new access geodatabase in your student directory.

STEP 2: WRITE CODE THAT CALLS THE CREATEACCESSGDB PROCEDURE

Now that you have written a procedure that can be used to create a new personal geodatabase, you will write a procedure that calls it. Your new procedure will pass in a pathname and a name for the database in order to return a pointer to the IWorkspace interface on the new workspace (review the sub procedure definition).

☐ Open the *ThisDocument* module in *ArcCatalog*.

☐ Define a new sub procedure as shown below.

```
Public Sub MakeSalesGDB ( )
```

☐ Begin the new procedure by declaring the following variable (with the Dim keyword).

```
pWorkspace As IWorkspace
```

☐ Set the pWorkspace variable by calling your CreateAccessGDB procedure, as shown below.

```
Set pWorkspace = CreateAccessGDB ("C:\Student", "AnnualSales")
```

☐ Run the *MakeSalesGDB* sub procedure.

☐ Bring *ArcCatalog* to the front of your display. Navigate to your student directory to verify that your code created a new personal geodatabase called *AnnualSales* (you may have to refresh the catalog before it appears).

☐ Delete the *AnnualSales* database from ArcCatalog (I know it was a lot of work. Don't worry—you will make another one.)

Now that you know the procedure to make a new geodatabase works, you will complete the MakeSalesGDB to create a table inside the new database.

STEP 3: CREATE A NEW TABLE

The Workspace class has methods for accessing and creating data. In the steps above, you dimensioned your pWorkspace variable as a pointer to the IWorkspace interface. This interface does *not* contain methods for creating new data.

☐ Return to the *MakeSalesGDB* sub procedure. You will continue adding code to this sub procedure to add a table and populate attribute values.

Question 4: Scan the Workspace class and find the interface with methods for creating new tables (among other things). Which one is it?___________________

☐ Write the code below in order to get the required interface on your new Workspace (QueryInterface).

```
Dim pFeatureWS As ________________________
Set pFeatureWS = pWorkspace
```

Notice that one of the arguments for the CreateTable method (on the IFeatureWorkspace interface) is a collection of fields (IFields). Before you can call CreateTable, you will need to write some code that makes a Fields collection.

☐ Declare the variables below (with the Dim keyword).

```
pFieldsEdit As IFieldsEdit  '<--This is IFieldsEdit, with an "s"!!

pOIDFld As IFieldEdit       '<--These are IFieldEdit, without an "s"!!
pStoreNameFld As IFieldEdit '<--
pSalesFld As IFieldEdit     '<--
```

! Field and Fields objects have an edit interface. If you are making new Field or Fields objects, you need to point to their editing interface (IFieldEdit, IFieldsEdit).

☐ Set the IFieldsEdit variable above (pFieldsEdit) equal to a new Fields (collection).

☐ Set each of the IFieldEdit variables above (pOIDFld, pStoreNameFld, and pSalesFld) equal to new Field objects.

☐ Assign values to the three field's properties, as shown in the table below.

Field	Name	Type	Length
pOIDFld	"OID"	esriFieldTypeOID	8
pStoreNameFld	"Store"	esriFieldTypeString	16
pSalesFld	"Sales"	esriFieldTypeInteger	8

☐ Add each field to the Fields collection (pFieldsEdit), as shown below.

! The AddField method is a hidden member. This means that it will not appear in the Visual Basic object browser nor in the code completion dropdown list. To show this (and all hidden members), open the Visual Basic object browser, right-click in the window, and choose Show Hidden Members from the context menu.

```
pFieldsEdit.AddField pOIDFld
pFieldsEdit.AddField pStoreNameFld
pFieldsEdit.AddField pSalesFld
```

☐ You can now call the CreateTable method (on IFeatureWorkspace) using the Fields collection as one of the required arguments, as shown below.

```
Dim pTable As ITable
Set pTable = pFeatureWS.CreateTable ("Sales2000", pFieldsEdit, Nothing, Nothing, "")
```

The CreateTable method has five required arguments: the name of the new table, the collection of fields, an identifier for a Dynamic Link Library (DLL) to provide custom behavior, an identifier for a class extension DLL (also providing some custom behavior), and an ArcSDE configuration keyword. Because you are not basing your table on custom objects, you satisfied the required DLL arguments with *Nothing*. Because you are not making a Spatial Database Engine (SDE) table, you used an empty string for the configuration keyword.

NOTE: *Nothing* is often used to satisfy an argument that requires an object. For a required String argument, use an empty string ("").

At this point, you have written code to produce a new Access geodatabase that contains a single table. In the next (and final) step, you will write code that adds data to your table.

STEP 4: ADD ROWS TO THE TABLE, ADD VALUES TO THE ROWS

In this step, you will add several new records (rows) to the Sales2000 table. You will also write code to populate these rows with some values. Although the code you write here will be technically correct, the process of adding records would not likely be *hard coded* into your application in this fashion (it would be more likely to use a form with text boxes to input data, for example).

☐ Declare a new variable to store a Row object, as shown below (use the Dim keyword).

```
pRow As IRow
```

☐ Set the pRow variable by calling the CreateRow method on ITable (your pTable variable).

☐ Assign values to the Store and Sales fields by writing the code below.

```
pRow.Value(1) = "Kwik-E Mart"
pRow.Value(2) = 300000
```

NOTE: The *Value* property takes a single required argument, an integer that corresponds to a field's index position in the table (counting from left to right, beginning with 0). The OID field (basically a record number) is always the first field (value of 0) and is maintained automatically by the software.

☐ Call the Store method on IRow (the pRow variable) to commit the new information to the table.

☐ Repeat the preceding three steps above (CreateRow, Value, Store) to create four additional rows with the following values:

Store	Sales
"Moe's Munch-n-Go"	437993
"The White Owl"	761402
"Tiltonelli's"	335078
"24-7"	711365

☐ Run the *MakeSalesGDB* procedure.

> **! Make sure you have deleted the AnnualSales geodatabase before running your code. Remember your code creates the AnnualSales geodatabase, therefore if the geodatabase already exists, your code will generate an error. To verify that the geodatabase has been deleted, bring ArcCatalog to the front of your display, navigate to your student directory and refresh the directory (press F5).**

☐ Bring *ArcCatalog* to the front of your display. Navigate to your student directory and locate the *AnnualSales* geodatabase.

☐ Open the *AnnualSales* geodatabase and verify that your table (Sales2000) was created.

> *NOTE:* You may need to refresh the AnnualSales.mdb (right-click and choose refresh) in order to see the changes.

☐ Highlight the table in ArcCatalog, then click the *Preview* tab. You should see a preview of your table, with the three fields and five records that were defined above.

Congratulations. You have successfully written code to produce a new personal geodatabase and a table. Because you wrote a standalone procedure to create the Access geodatabase (CreateAccessGDB), you can reuse this code in any project that needs it.

Export your CreateData module to your student directory (there is no document to save in ArcCatalog; all the code you wrote is stored in the Normal.gxt template).

EXERCISE END

Answers to Exercise 9B questions

Question 1: Which method or property can be used to make a new workspace?

Answer: Create

Question 2: In order to retrieve the actual Workspace object, you will use the Open method supported by the WorkspaceName class. Using the Developer Help examining WorkspaceName, can you locate the interface that contains the Open method?

Answer: The Open method is found on the IName interface. If using the Object Model Diagrams, IName is not listed on the WorkspaceName class because it is located on another object model diagram (System Object Model) and listed under "Interfaces" (not listed on a particular class).

Question 3: Locate the Name object on the object model diagram. What is the relationship of WorkspaceName to Name? What does the Open method on IName return?

Answer: WorkspaceName is a "type of" Name, indicating inheritance. The open method returns a pointer to IUnknown. The Open method can be used to open a variety of objects. Because the Object Model Diagram has no way of anticipating which object you will be opening, IUnknown is used.

Question 4: Scan the Workspace class and find the interface with methods for creating new tables (among other things). Which one is it?

Answer: IFeatureWorkspace has a createtable method which can be used to create new tables.

SOLUTIONS

```
Public Function CreateAccessGDB(ParentFolder As String, GDBName As String) _
As IWorkspace
  Dim pAccessWSF As IWorkspaceFactory
  Dim pWorkName As IWorkspaceName
  Dim pName As IName
  Dim pWorkspace As IWorkspace
  Set pAccessWSF = New AccessWorkspaceFactory
  Set pWorkName = pAccessWSF.Create(ParentFolder, GDBName, Nothing, Application.hWnd)
  Set pName = pWorkName
  Set pWorkspace = pName.Open
  Set CreateAccessGDB = pWorkspace
End Function

Public Sub MakeSalesGDB()
  Dim pWorkspace As IWorkspace
  Set pWorkspace = CreateAccessGDB("C:\Student", "AnnualSales")
  Dim pFeatureWS As IFeatureWorkspace
  Set pFeatureWS = pWorkspace

  Dim pFieldsEdit As IFieldsEdit
  Dim pOIDFld As IFieldEdit
  Dim pStoreNameFld As IFieldEdit
  Dim pSalesFld As IFieldEdit

  Set pOIDFld = New Field
  Set pStoreNameFld = New Field
  Set pSalesFld = New Field

  Set pFieldsEdit = New Fields

  pOIDFld.Name = "OID"
  pOIDFld.Type = esriFieldTypeOID
  pOIDFld.Length = 8

  pStoreNameFld.Name = "Store"
  pStoreNameFld.Type = esriFieldTypeString
  pStoreNameFld.Length = 16

  pSalesFld.Name = "Sales"
  pSalesFld.Type = esriFieldTypeInteger
  pSalesFld.Length = 8

  pFieldsEdit.AddField pOIDFld
  pFieldsEdit.AddField pStoreNameFld
  pFieldsEdit.AddField pSalesFld

  Dim pTable As ITable
  Set pTable = pFeatureWS.CreateTable _
       ("Sales2000", pFieldsEdit, Nothing, Nothing, "")
  Dim pRow As IRow
  Set pRow = pTable.CreateRow
  pRow.Value(1) = "Kwik-E Mart"
  pRow.Value(2) = 300000
  pRow.Store
  Set pRow = pTable.CreateRow
  pRow.Value(1) = "Moe’s Munch-n-Go"
  pRow.Value(2) = 437993
  pRow.Store

  Set pRow = pTable.CreateRow
  pRow.Value(1) = "The White Owl"
  pRow.Value(2) = 761402
  pRow.Store

  Set pRow = pTable.CreateRow
  pRow.Value(1) = "Tiltonelli’s"
  pRow.Value(2) = 335078
  pRow.Store

  Set pRow = pTable.CreateRow
  pRow.Value(1) = "24-7"
  pRow.Value(2) = 711365
  pRow.Store
End Sub
```

10

Geometry and geoprocessing

contents

EXERCISE 10: USE COORDINATE INPUT TO DRAW FEATURES

In this exercise, you will learn to use some of the methods and properties related to point, line, and polygon geometry. You will write an application that helps input information to a database of point features containing soil characteristics. The input of point location and attributes into the GIS must be done manually by entering coordinates and other information read from a standard form.

EXERCISE SHORTCUT

1. Open *ex10.mxd* from *..\IPAO\Maps*. In the *frmMakePoints* form, navigate to the *cmdMakePoint_Click* procedure. Code this procedure to create a new point in the Soil Samples feature class based on the *X* and *Y* text boxes.

2. In the *cmdMakePoint_Click* procedure, use *Description*, *Depth*, and *pH* text boxes to populate the appropriate fields.

3. Code the *frmMakePolygon* form to capture a series of points the user inputs (see coordinate value on page 10-6). Store these points in an IPointCollection object. Use these points to create a new polygon and draw it on the display when the user clicks the *Draw Polygon* button.

STEP 1: CREATE FEATURES FROM USER INPUT

In this step, you will open an existing map that contains a form with all the necessary controls for capturing a single pair of geographic (x,y) coordinates and several soil attributes from the user. You will add the code required to produce a new point feature, as well as the attributes for the corresponding record in the table. You will first need to start ArcMap and open the map document.

☐ Start *ArcMap*.

☐ Navigate to *C:\Student\ipao\maps* and open *ex10.mxd.*

When the map opens, you will see a map showing a few locations that have been surveyed for soil chemistry. The surveys were done years ago and currently all the information resides in hardcopy format. As you can see, there are only a handful of survey locations in the database, but new data forms are coming in daily; you need a better way to input the data so that it is up to date.

☐ Start the *Visual Basic Editor.*

☐ In the *Project Explorer* window, open the form *frmMakePoints* in the *Project > Forms folder.*

☐ Examine the controls on the form *frmMakePoint*.

You will add code to the form to make sure there is a selected layer in the map document, create a reference to the selected layer, then create a new point feature and set its geometry equal to the coordinates entered in the text boxes.

☐ Double-click on *frmMakePoint* to open its *Code* window.

☐ In the *General Declarations* section, declare the following module-level variables:

```
m_pFLayer As IFeatureLayer
m_pFClass As IFeatureClass
m_pFeature As IFeature
m_pPoint As IPoint
```

Question 1: Which Library and Object Model Diagram contains the IPoint interface?____

____

Next, you will add code to the Make Point button.

☐ In the Code window, navigate to the *cmdMakePoint_Click* procedure.

You will add new code below this existing statement:

```
Set m_pMxDoc = ThisDocument
```

☐ Create an If/Then statement to test If the current document's ActiveView is not a map (supports the IMap interface) by using the TypeOf keyword. If the current document is not a map, display a message box notifying the user that a map must be active, then Exit the Sub.

Question 2: What would cause the ActiveView not to support the IMap interface?

____

☐ Write the following code to set `m_pFLayer` to the appropriate layer in the Table of Contents. The following code would be useful if you have many layers and are looking for one by name. This is a more efficient way of finding a layer than getting the selected layer.

```
Dim iLoop As Integer 'keep track of which layer you are on
Dim pCheckforLayer As ILayer 'a temporary variable used to check all layers
For iLoop = 0 To m_pMxDoc.______.LayerCount - 1
    If TypeOf m_pMxDoc.FocusMap.Layer(iLoop) Is ________ Then
```

```
        Set pCheckforLayer = m_pMxDoc.FocusMap.Layer(_____)
        If pCheckforLayer.Name = "Soil Samples" Then
           Set m_pFLayer = pCheckforLayer
        End If
     End If
Next iLoop
```

☐ Initialize the following variables (using Set).

```
m_pFClass = m_pFLayer.FeatureClass 'the soils FC
m_pFeature = m_pFClass.CreateFeature
m_pPoint = New Point
```

The code above has added a new record (feature) to the Soil Samples feature class (a blank row in the table) as well as a new point that will eventually define the geometry of the new feature. Next, you will take the information provided by the user and set the geometry of the new point.

Question 3: Using the *ArcGIS Developer Help* or *Geometry OMD*, do you see any members that allow you to set the X and Y values for the point you have created?

____

☐ Assign the X property of `m_pPoint` equal to the Text in txtXCoord.

☐ Do the same for the point's Y property, using the Text in txtYCoord.

☐ Set the Shape property of `m_pFeature` equal to `m_pPoint`.

Question 4: Using the ArcGIS Developer Help, what is the return type for the Shape property on `m_pFeature`? ____

____

Hint: You can place your cursor on the Shape Property and press *F1*.

Question 5: If the return type for the Shape Property is IGeometry, why can you set the value to `m_pPoint`, which is declared as IPoint? ________________________

____

Hint: Use the Geometry OMD to find the relationship between IGeometry and the Point class.

☐ Use m_pFeature's Store method to commit the new geometry to the feature class.

☐ Refresh the map by sending the Refresh method to m_pMxDoc's ActiveView.

☐ Launch the form to add new points.

☐ Provide an x- and y-coordinate that is within the appropriate range for the study area. Try `-117.0199, 45.8444`. Do not worry about the other text boxes at the moment.

☐ Click the *Make Point* button.

You should see a new point added to the layer. If you are unable to find your new point on the display, open the soil points attribute table. You should find a new row (with only a shape value and no attributes) at the bottom of the table.

Step 2: Populate feature attributes

You have successfully added a new point to the layer, but you still need to define the attributes of each new sample point. In this step, you will write code to take the descriptive information entered on the form and add it to the corresponding feature attributes.

☐ Return to the *Visual Basic Editor*.

☐ Open the *Code window* for the form (*frmMakePoints*).

☐ Navigate to the *cmdMakePoint_Click* procedure.

☐ Open a new line immediately above the line that stores the feature and write the following code to add the Description attribute.

```
Dim intDescPosition As Integer
intDescPosition = m_pFClass.FindField("Desc")
m_pFeature.Value(intDescPosition) = txtDesc.Text
```

The parameter intDescPosition specifies the index position of the Description field, which was returned by the FindField method of the Feature Class.

☐ Now write similar code to assign the pH (PH) and Depth (Depth) attributes from the respective text boxes.

☐ Test your form once again by entering appropriate x- and y-values and some attribute information.

☐ When finished, open the table for the soil points layer and verify that the attributes were written to the new features.

Step 3: Create a polygon from a point collection

As your field season progressed, you received a threatening letter from Farmer Jack, an irate landowner in your study area. He warned you to keep your survey crew off of his property. At the bottom of his letter were scrawled the GPS positions for each fence post surrounding his property. You need to input these points immediately and produce a polygon so you can warn your graduate students to stay away from Farmer Jack's land.

☐ Return to the *Visual Basic Editor* and open the form *frmMakePolygon* in the *Project > Forms folder.*

☐ Examine the controls you will use to create a polygon from a collection of points.

Each point will be input individually by typing the X and Y position and clicking *Add Point*. After each point is stored in the PointCollection, the user will be able to click *Make Polygon* to draw the polygon defined by the points. As a challenge step, the label at the top of the form will keep track of how many points have been added to the current collection.

☐ Open the *Code window* for the *form*.

☐ Navigate to the *General Declarations* section.

☐ Notice the variables that have already been declared for you. The last two objects, SimpleFillSymbol and RgbColor, will be discussed in more detail in a later lesson.

☐ Add the following variable declarations using the Private statement.

```
m_pPointColl As IPointCollection
m_pPoint As IPoint
```

☐ Navigate to the *UserForm_Initialize* event procedure.

The existing code in this procedure simply initializes some of the form level variables when the form is first opened (initialized).

☐ Below the existing code, add the following line to initialize `m_pPointColl`.

```
Set m_pPointColl = New Polygon
```

☐ Navigate to the *Click* procedure for *cmdAddPoint*.

You will add code here to first create a new point, then set the point's X and Y properties according to values in the text boxes, and finally add the point to the point collection.

☐ Start by setting the form-level variable `m_pPoint` equal to a New Point.

☐ Set m_pPoint's X property equal to the text in txtPointX.

☐ Set m_pPoint's Y property equal to the text in txtPointY.

☐ Use m_pPointColl's AddPoint method to put m_pPoint in the collection.

Only the few lines of code you wrote above are required to produce a point collection from user input. If the user inputs at least three points, you can display the polygon.

☐ Navigate to the *Click* procedure for *cmdMakePolygon*.

Notice that some code has already been written for you (you will learn more about working with display in a later lesson).

☐ Below the line that sets the symbol (with .SetSymbol) in the 'With m_pSDisplay' block, add this line of code to draw the polygon (PointCollection).

```
.DrawPolygon m_pPointColl
```

☐ Test your form.

☐ Repeatedly enter the x- and y-coordinates from the list below. After entering each point, click the *Add Point* button.

☐ After adding all four points, click the *Draw Polygon* button.

Below are the GPS positions for Farmer Jack's property boundary. You'd better hurry—your students will be heading to the field soon.

Point No.	X -Coordinate	Y-Coordinate
1	-117.02	45.85
2	-117.01	45.85
3	-117.01	45.83
4	-117.02	45.84

Does the polygon draw on the display? ________ Notice that as soon as the display is refreshed, the polygon disappears. The polygon was not added as a graphic to the map (which will be discussed in a later lesson) but was simply painted temporarily to the screen.

If you have time, proceed to the challenge steps. Otherwise, save your map to the student directory.

CHALLENGE: DISPLAY THE POINT COLLECTION COUNT

In this challenge step, you will add code to display the number of points currently stored in frmMakePolygon's PointCollection (`m_pPointColl`).

☐ Display the number of points that have been added to the point collection in *lblPointNum*. (Hint: There is more than one way to do this.)

CHALLENGE: WORK WITH SPATIAL REFERENCE AND CALCULATE AREA

In this challenge step, you will add code to find the area of the polygon you just created. However, there is not a spatial reference assigned to this data, so in order to obtain meaningful area values, you must first apply a spatial reference to the polygon.

☐ Add a new command to the form and name it **`cmdGetArea`**. You may need to enlarge your form a little bit. Give it an appropriate caption.

☐ Declare a variable to represent the polygon in the General Declarations.

```
Private m_pPoly as IPolygon
```

☐ Set this variable equal to the point collection in the click event for *cmdMakePolygon*. This is an example of QueryInterface.

```
Set m_pPoly = m_pPointColl
```

If you look at your previous code, you see that you used IPointCollection to create a new Polygon. This is fine, but in order to work with the spatial reference you need to work with the IPolygon interface. The above line of code is simply letting you query interfaces.

☐ Your end goal is to get the area property, so you must declare a variable as IArea and query interface yet again, this time in click event for cmdGetArea.

```
Dim pArea As IArea
```

```
Set pArea = m_pPoly
MsgBox pArea.Area
```

☐ Run the make polygon entry again. Move the form to the side, so you can see the display. Enter in the first 3 points from page 10-6. Add the coordinates for one more point that is the same as your first point to enclose the polygon. Draw the polygon and click your new command button.

Notice that the area returned does not make much sense at this point. The area was calculated, but in angular units and is being returned in Decimal Degrees. In order to make accurate area calculations, you should have your data in a projected coordinate system. You must apply some type of projection to your polygon to correct this.

☐ Remove the message box reporting the area; you will add a new one with the appropriate area calculation.

☐ Your first step is to create a new Spatial Reference Environment to create your projection information in the *cmdGetArea* click event.

```
Dim pSpRFc As ISpatialReferenceFactory
Set pSpRFc = New SpatialReferenceEnvironment
```

☐ Add the following lines of code to define the current Geographic Coordinate system being used to enter the points.

```
Dim pGCS As IGeographicCoordinateSystem
Set pGCS = pSpRFc.CreateGeographicCoordinateSystem(esriSRGeoCS_NAD1983)
Dim pSpRef1 As ISpatialReference
Set pSpRef1 = pGCS
```

ESRI has many predefined geographic and projected coordinate systems that can be used. In this case, you are defining your points with the pre-defined constant esriSRGeoCS_NAD1983. You are defining the polygon with the coordinates that were used, in this case a Geographic Coordinate system, using the North American Datum of 1983.

☐ Now using the same approach, define a new Spatial Reference that refers to a projected coordinate system to apply to the data.

```
Dim pPCS As IProjectedCoordinateSystem
Set pPCS = pSpRFc.CreateProjectedCoordinateSystem(esriSRProjCS_NAD1983SPCS_ORNorthFT)
Dim pSpRef2 As ISpatialReference
Set pSpRef2 = pPCS
```

You want to assign a projection that will minimize your distortion. A State Plane or UTM projection are appropriate projections for minimizing distortion. The above code is defining a new Projected Coordinate System as a State Plane Projection in the North Oregon zone, based on the 1983 North American Datum. You assigned this projection to a spatial reference. The linear unit for this particular projection is in feet. You can now apply the spatial reference to your polygon.

☐ Finish the code by applying the spatial reference and returning the area.

```
Set m_pPoly.SpatialReference = pSpRef1
m_pPoly.Project pSpRef2
MsgBox "Farmer Jack owns " & pArea.Area & " Square Feet of land"
```

☐ Run the form again. Move it out of the way of the display area and add in all the points from page 10-5 as you did previously, add one last point that replicates the very first point to enclose the polygon. Draw the polygon and test your Area calculator.

EXERCISE END

ANSWERS TO EXERCISE 10 QUESTIONS

Question 1: Which Library and Object Model Diagram contains the IPoint interface?

Answer: The IPoint interface is found in the Geometry Library (esriGeometry) and the Geometry Object Model Diagram. You can use the Index tab in the ArcGIS Developer Help and type in IPoint to find this information.

Question 2: What would cause the ActiveView not to support the IMap interface?

Answer: The Active View is the display area in ArcMap. This display area can either be in Data View (the default) or Layout View. When viewing the Data View, you are essentially viewing the Map class. When you are viewing the Layout View, you are viewing the Page Layout class. The Page Layout class does not support IMap, therefore if you were viewing the Page Layout, the MsgBox would appear.

Question 3: Using the ArcGIS Developer Help or Geometry OMD, do you see any members that allow you to set the X and Y values for the point you have created?

Answer: On the IPoint interface, there is an *X* and a *Y* property that allow you to define the coordinates. There is also a *PutCoords* method that allows you to set both the X and Y value on one line of code.

Question 4: Using the ArcGIS Developer Help, what is the return type for the Shape property on `m_pFeature`?

Answer: The return type for the Shape property is IGeometry.

Question 5: If the return type for the Shape Property is IGeometry, why can you set the value to `m_pPoint`, which is declared as IPoint?

Answer: Because m_pPoint is referencing the Point class, you can determine by the TypeOf relationship that the Point class also inherits the IGeometry interface from the Geometry class.

SOLUTIONS

frmMakePoints

```
Option Explicit
Private m_pMxDoc As IMxDocument

Dim m_pFLayer As IFeatureLayer
Dim m_pFClass As IFeatureClass
Dim m_pFeature As IFeature
Dim m_pPoint As IPoint

Private Sub cmdMakePoint_Click()
  Set m_pMxDoc = ThisDocument

  If Not TypeOf m_pMxDoc.ActiveView Is IMap Then
      MsgBox "A Map Must be Active!"
      Exit Sub
  End If

  Dim iLoop As Integer 'keep track of which layer you are on
  Dim pCheckforLayer As ILayer 'a temporary variable used to check all layers
  For iLoop = 0 To m_pMxDoc.FocusMap.LayerCount - 1
     If TypeOf m_pMxDoc.FocusMap.Layer(iLoop) Is IFeatureLayer Then
        Set pCheckforLayer = m_pMxDoc.FocusMap.Layer(iLoop)
        If pCheckforLayer.Name = "Soil Samples" Then
            Set m_pFLayer = pCheckforLayer
         End If
     End If
  Next iLoop

  Set m_pFClass = m_pFLayer.FeatureClass
  Set m_pFeature = m_pFClass.CreateFeature
  Set m_pPoint = New Point

  m_pPoint.X = txtXCoord.Text
  m_pPoint.Y = txtYCoord.Text
  Set m_pFeature.Shape = m_pPoint

  Dim intDescPosition As Integer
  intDescPosition = m_pFClass.FindField("Desc")
  m_pFeature.Value(intDescPosition) = txtDesc.Text

  Dim intPHPosition As Integer
  intPHPosition = m_pFClass.FindField("PH")
  m_pFeature.Value(intPHPosition) = txtPH.Text

  Dim intDepthPosition As Integer
  intDepthPosition = m_pFClass.FindField("Depth")
  m_pFeature.Value(intDepthPosition) = txtDepth.Text

  m_pFeature.Store
  m_pMxDoc.ActiveView.Refresh

End Sub
Private Sub cmdQuit_Click()
  Unload Me
End Sub
```

frmMakePolygons

```
Option Explicit
Private pMxApp As IMxApplication
Private m_pMxDoc As IMxDocument
Private m_pSDisplay As IScreenDisplay
Private pSym As ISimpleFillSymbol
Private pColor As IRgbColor
Private m_pPointColl As IPointCollection
Private m_pPoint As IPoint
Private m_pPoly as IPolygon

Private Sub cmdAddPoint_Click()
  Set m_pPoint = New Point
  m_pPoint.X = txtPointX.Text
  m_pPoint.Y = txtPointY.Text
  m_pPointColl.AddPoint m_pPoint
'**Challenge 1
  lblPointNum.Caption = "Points: " & m_pPointColl.PointCount
'**
End Sub

Private Sub cmdMakePolygon_Click()
  With m_pSDisplay
      .StartDrawing m_pSDisplay.hDC, 0
      .SetSymbol pSym
      .DrawPolygon m_pPointColl
      .FinishDrawing
  End With
End Sub

Private Sub cmdQuit_Click()
  Unload Me
End Sub

Private Sub UserForm_Initialize()
  Set pMxApp = Application
  Set m_pMxDoc = ThisDocument
  Set m_pSDisplay = pMxApp.Display
  Set pColor = New RgbColor
  pColor.RGB = vbRed
  Set pSym = New SimpleFillSymbol
  pSym.Style = esriSFSCross
  pSym.Color = pColor
  m_pSDisplay.SetSymbol pSym
  Set m_pPointColl = New Polygon
End Sub

‘**Challenge 2
Private Sub cmdGetArea_Click()
    Call Project
End Sub

Public Sub Project()
    Dim pArea As IArea
    Set pArea = m_pPoly
    MsgBox pArea.Area 'This is the area in the native data units (data frame)

      Dim pSpRFc As SpatialReferenceFactory
```

```
   Set pSpRFc = New SpatialReferenceEnvironment
   Dim pGCS As IGeographicCoordinateSystem
   Set pGCS = pSpRFc.CreateGeographicCoordinateSystem(esriSRGeoCS_NAD1983)
   Dim pSpRef1 As ISpatialReference
   Set pSpRef1 = pGCS

   Dim pPCS As IProjectedCoordinateSystem
   Dim pSpRef2 As ISpatialReference
   Set pPCS = _
pSpRFc.CreateProjectedCoordinateSystem(esriSRProjCS_NAD1983SPCS_ORNorthFT)
   Set pSpRef2 = pPCS

   Set m_pPoly.SpatialReference = pSpRef1
   m_pPoly.Project pSpRef2
       MsgBox "Farmer Jack owns " & pArea.Area & " Square Feet of land"
End Sub
```

11

Working with subsets and selections

contents

EXERCISE 11: WORKING WITH SUBSETS AND SELECTIONS

In this exercise, you will work with some of the ArcObjects related to making selections and producing subsets of data based on attribute criteria, spatial criteria, or both. You will learn how to make a query definition for a feature layer to easily display a subset of features. Finally, you will produce spatial and attribute queries to report information about a subset of features.

EXERCISE SHORTCUT

1. Open *ex11.mxd* from ../IPAO/Maps. Create a definition expression for the selected layer. Set the Country layer to the selected layer. Using the Population Field, have the user define the expression for the minimum population to be displayed through an Input Box.

2. Use a QueryFilter to query landlocked countries. Create a Search Cursor applying the Query Filter to the Country Feature Class. Display information for the Population and Area attribute fields, as well as the total number of countries that meet the criteria in a Message Box.

3. Create a UIButtonControl that will summarize the number of cities with a population > 2 million for the selected country. Use a SpatialFilter object to determine the relationship between the selected country and the City layer.

STEP 1: APPLY A DEFINITION QUERY TO A LAYER IN THE MAP

In this step, you will write code to restrict the features displayed in the selected layer. You will define a subset of features by using a definition query, which uses attribute values to limit the features displayed in a map layer (and corresponding attribute table).

☐ Start *ArcMap*.

☐ Open *ex11.mxd* from your *Student* directory.

☐ Use the *Customize* dialog box to create a new UIButtonControl (make sure to save it in the current document: ex11.mxd).

☐ Add the new control as a button to the interface.

☐ Right-click the new control and choose *View source*. The *Visual Basic Editor* appears.

☐ Move to the top of the *ThisDocument* code module and declare the following module-level variable (using the Private keyword in *General Declarations*).

```
m_pMxDoc As IMxDocument
```

☐ Navigate to UIButtonControl1's *Click* event procedure (*UIButtonControl1_Click*) and declare the following variables (using the Dim keyword).

```
pLayer As ILayer
pFLayerDefinition As IFeatureLayerDefinition
```

☐ Set the required variables, as shown below.

```
Set m_pMxDoc = ThisDocument
Set pLayer = m_pMxDoc.SelectedLayer
```

☐ Complete the branching statement below to verify that there is a selected layer in the document and that it is a FeatureLayer. If there is no selected layer, or if it is not a FeatureLayer, you will terminate execution of the procedure.

```
If _______________ Is _______________ Then
  MsgBox "No layer is selected"
  ________ Sub
______ Not TypeOf pLayer Is ____________________ Then
  MsgBox "Selected layer is not a feature layer"
  Exit _____
End If
```

After the above branching statement, the rest of your code can safely assume that the selected layer is a feature layer.

☐ Query interface to reference the IFeatureLayerDefinition interface on the selected layer, as shown below.

```
Set pFLayerDefinition = pLayer
```

☐ Assign the string below to pFLayerDefinition's DefinitionExpression property.

```
"POPULATION > " & InputBox ("Enter a population")
```

☐ Refresh the map to see the definition applied, as shown below.

```
m_pMxDoc.ActiveView.Refresh
```

☐ Bring *ArcMap* to the front of your display. Select the cities layer in the Table of Contents.

Question 1: Test your button. Type a value of `5000000` (5 million) in the input box. You should see the definition expression applied to the cities layer. How many cities have a population greater than 5 million? ________________________________
Hint: Open the attribute table.

Question 2: Select the country layer in the Table of Contents and test your button. Show countries with a population greater than `100000000` (100 million). How many are there? ____ What happens if you run your code on a layer that does not have a POPULATION attribute (Lakes or Rivers, for example)? ____________________

To clear a layer's definition expression (from the interface): Open the *Layer Properties* dialog, click the *Definition Query* tab, delete the expression in the dialog, then press *OK*.

> NOTE: You can programmatically clear a definition expression in code by setting the DefinitionExpression property equal to an empty string (""). As a challenge, you might want to write code that clears the definition when a user clicks the Cancel button on the input box.

STEP 2: APPLY A QUERYFILTER, LOOP THROUGH A SUBSET OF RECORDS

In this step, you will use a QueryFilter object to produce a subset of country features that are landlocked. You will access each of these records from a FeatureCursor in order to calculate basic statistics for records matching the query filter's criteria (landlocked countries).

☐ Insert a new sub in the Project's *ThisDocument* code module, as shown below.

```
Public Sub CalcLandlockStats ()
```

☐ Begin by declaring the variables below (using the Dim keyword).

```
pFLayer As IFeatureLayer
pFClass As IFeatureClass
```

☐ Set the following variable:

```
Set m_pMxDoc = ThisDocument  'This is a module-level variable!
```

☐ Set pFLayer equal to the fourth layer in the document's active data frame (focus map). This should be the Country layer.

Hint: Remember the 4th layer is Index #3.

☐ Set pFClass equal to the pFLayer's FeatureClass property.

☐ Dimension the following variables (using Dim):

```
pQFilter As IQueryFilter
pFCursor As IFeatureCursor
```

☐ Set pQFilter by creating a New QueryFilter.

☐ Assign the following string to pQFilter's WhereClause property.

```
"LANDLOCKED = 'Y'"
```

The Country feature class has an attribute called *Landlocked*. It has values of 'Y' for countries that do not have a coastline and values of 'N' for countries that do (yes/no).

☐ Write the code below to search the Country feature class for all records with a value of 'Y' for the landlocked attribute. The records matching this criterion will be stored in the FeatureCursor (pFCursor).

```
Set pFCursor = pFClass.Search (pQFilter, True)
```

NOTE: The required parameters for the Search method are a query filter and a Boolean to specify whether or not memory should be recycled. For read-only operations, recycling memory is more efficient. Do not recycle memory when creating an edit cursor.

The feature cursor resulting from the code above (pFCursor) stores a subset of landlocked country features. Next, you will need to write a looping structure to access each of the records in the cursor and to calculate basic statistics.

☐ Declare the variables below (use the Dim keyword).

```
pFeature As IFeature
intPop As Long
intArea As Long
intCountries As Integer
```

☐ Set pFeature by pulling the first record out of the cursor by writing the code below.

```
Set pFeature = pFCursor.NextFeature
```

NOTE: Remember that a cursor is initialized with the pointer above the first item. The first time NextFeature (or NextRow) is called, the first item in the cursor is returned.

☐ Begin a Do Until loop by completing the code below.

```
Do Until pFeature Is _________________
```

☐ Write the following code inside the loop to keep a total population, area, and count for the countries in the cursor (you do not need to include the comments).

```
intPop = intPop + pFeature.Value(9) 'Population is the 10th field
intArea = intArea + pFeature.Value(10) 'Area is the 11th field
intCountries = intCountries + 1  'Keep count of the total number of countries
```

☐ Return the next feature from the cursor by completing the code below.

```
Set pFeature = pFCursor.______________________
```

! **If you do not include the line of code above inside the Do Until loop, your code will get stuck in an infinite loop.**

☐ End the Do Until loop with the *Loop* keyword.

☐ Finally, display information about the landlocked countries, such as the total number, average population, area, and population density, by writing the code below.

```
Dim dblPopDensity As Double
dblPopDensity = intPop / intArea
```

```
MsgBox "Information on landlocked countries: " & vbCrLf & _
        "Number: " & intCountries & vbCrLf & _
        "Average Area: " & (intArea / intCountries) & vbCrLf & _
        "Population Density: " & dblPopDensity
```

NOTE: The area values are expressed in square kilometers.

Question 3: Run the CalcLandlockStats procedure. How many landlocked countries are there? ____
What is the population density for these countries? ___________________________

STEP 3: CREATE A SUBSET BASED ON BOTH SPATIAL AND ATTRIBUTE CRITERIA

In this step, you will make a button that lets the user summarize a subset of features based on both spatial and attribute criteria at the same time. Your code will report the number of cities that have a population greater than two million within whichever country is currently selected in the map.

☐ Open the *Customize* dialog box.

☐ Select *ex11.mxd* from the *Save in* pulldown list.

☐ Create a new UIButtonControl and add it to one of the ArcMap toolbars.

☐ Right-click the control on the interface and choose *View Source* to bring up the *Visual Basic Editor*.

Your code will produce a feature cursor of cities that fall within the selected country in the Countries layer. To accomplish this, you will need to make a spatial filter that uses the selected country's shape as the geometry property. The code you write below will reference the selected country by getting the layer's selection and storing it in a cursor.

☐ Declare the following variables (using the Dim keyword):

```
pCountryLayer As IFeatureSelection
pCountrySelection As ISelectionSet
pCountryCursor As IFeatureCursor
```

☐ Set the required object variables as shown below (you do not need to include the comments).

```
Set m_pMxDoc = ThisDocument '<-- Remember that this is a module-level variable!
Set pCountryLayer = m_pMxDoc.FocusMap.Layer(3) '<-- Countries is the 4th layer
Set pCountrySelection = pCountryLayer.SelectionSet '<-- Return the selection
```

Because pCountryLayer was dimensioned with the IFeatureSelection interface, you can access selected features by using the SelectionSet property. The next code you write will transfer the selection set to a feature cursor so you can easily retrieve the selected country.

☐ Move all features in the selection set (pCountrySelection) to a feature cursor (pCountryCursor) by writing the code below. Remember that *Nothing* may be specified as the query filter parameter to return all features.

```
pCountrySelection.Search Nothing, True, pCountryCursor
```

> NOTE: The syntax for the Search method works slightly different on the ISelectionSet interface. Instead of storing the returned cursor using an assignment statement (e.g., Set myCursor = myTable.Search), the cursor variable is passed in as one of the required parameters.

☐ Declare a variable to contain the selected country (or the first county if there was more than one selected), then return it from the cursor, as shown below.

```
Dim pCountry As IFeature
Set pCountry = pCountryCursor.NextFeature
```

☐ Complete the code below to make sure a country was selected. If no country is selected, report an error to the user, then stop execution of the procedure.

```
If pCountry Is ___________ Then
    MsgBox "Please select a country"
    _______ Sub
End If
```

Now that you have referenced the selected country, you will create a new spatial filter using the country's shape for the geometry property. You will declare two variables to work with the same SpatialFilter object: one will point to the ISpatialFilter interface, and the other will point to the IQueryFilter interface.

☐ Declare the variables below, using the Dim keyword.

```
pSpatialFilter As ISpatialFilter
pQueryFilter As IQueryFilter
```

☐ Now write code to create a new SpatialFilter object.

```
Set pSpatialFilter = New SpatialFilter
```

☐ Use QueryInterface to point your IQueryFilter variable (pQueryFilter) to the same SpatialFilter, as shown below.

```
Set pQueryFilter = pSpatialFilter
```

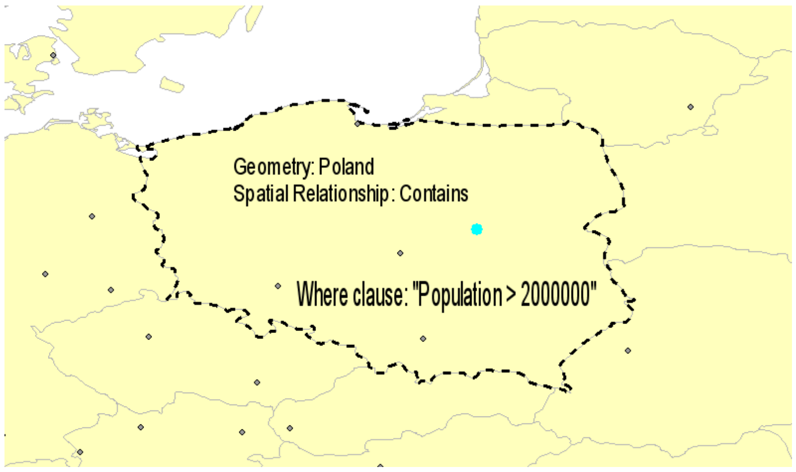
NOTE: Because SpatialFilter *is a type of* QueryFilter, it inherits all interfaces defined on QueryFilter. By using both the ISpatialFilter interface and the IQueryFilter interface on the same object, it is possible to define an attribute expression (WhereClause property) and a spatial criteria (Geometry and SpatialRel properties) for a single filter.

☐ Set all of the filter's selection criteria using the two variables that point to it, by completing the code below. Because you want to locate cities inside the selected country, the spatial relationship will be *Contains*.

```
______________.WhereClause = "Population > 2000000"
Set ______________.Geometry = pCountry.Shape
pSpatialFilter.______________ = esriSpatialRelContains
```

NOTE: ISpatialFilter has a SearchOrder property that controls whether the attribute expression or the spatial criteria are evaluated first in an ArcSDE geodatabase. By default, the spatial criteria are evaluated first, followed by the attribute. This property only affects ArcSDE geodatabase data.

Now that you have produced the spatial filter and defined the spatial and attribute criteria required, you will apply the filter to the city feature class.


Geometry: Poland
Spatial Relationship: Contains
Where clause: "Population > 2000000"

☐ Declare the following variables (using the Dim keyword).

```
pCityLayer As IFeatureLayer
pCityFClass As IFeatureClass
```

☐ Set the variables as shown below (you do not need to include the comments).

```
Set pCityLayer = m_pMxDoc.FocusMap.Layer(0) '<-- Cities is the 1st layer
Set pCityFClass = pCityLayer.FeatureClass '<-- FeatureClass stores the features
```

☐ Declare the variables below, using the Dim keyword.

```
pCityCursor As IFeatureCursor
pCity As IFeature
intCount As Integer
```

☐ Apply the spatial filter to the city feature class using the Search method. Store the returned feature cursor in the pCityCursor variable. (Refer to the IFeatureClass interface in the object browser or in the online Help if you need help with the Search method syntax.)

☐ Complete the code below to loop through and count each feature (city) in the cursor. (Refer to the example in the lecture, or the code you wrote earlier if you need help)

```
Set pCity = pCityCursor.________________
Do ________ pCity Is _________________
    intCount = intCount + ___
    Set pCity = ___________.________________
______
```

☐ Report the number of cities that met the search criteria by displaying a message box.

☐ Bring *ArcMap* to the front of your display. Make sure you have a country selected, then test your button. It should tell you how many cities in the selected country have a population greater than two million.

☐ Save your map to your student directory.

EXERCISE END

ANSWERS TO EXERCISE 11 QUESTIONS

Question 1: Test your button. Type a value of `5000000` (5 million) in the input box. You should see the definition expression applied to the cities layer. How many cities have a population greater than 5 million?

Answer: 30

Question 2: Select the country layer in the Table of Contents and test your button. Show countries with a population greater than `100000000` (100 million). How many are there? What happens if you run your code on a layer that does not have a POPULATION attribute (Lakes or Rivers, for example)?

Answer: 9 countries have a population greater than 100 million. If you run the code on a layer without a POPULATION attribute, you get an error stating "One or more layers failed to draw: General function failure [Rivers]".

Question 3: Run the CalcLandlockStats procedure. How many landlocked countries are there? What is the population density for these countries?

Answer: There are 43 landlocked countries. The population density for these countries is 20.68758 people per square kilometer.

SOLUTIONS

```
Private m_pMxDoc As IMxDocument

Private Sub UIButtonControl1_Click()
  Dim pLayer As ILayer
  Dim pFLayerDefinition As IFeatureLayerDefinition
  Set m_pMxDoc = ThisDocument
  Set pLayer = m_pMxDoc.SelectedLayer

  If pLayer Is Nothing Then
      MsgBox "No layer is selected"
      Exit Sub
  ElseIf Not TypeOf pLayer Is IFeatureLayer Then
      MsgBox "Selected layer is not a feature layer"
      Exit Sub
  End If
  Set pFLayerDefinition = pLayer
'Next line is a challenge from Note at the end of Step 1
  pFLayerDefinition.DefinitionExpression = ""
  pFLayerDefinition.DefinitionExpression = "POPULATION > " & _
    InputBox("Enter a population")
  m_pMxDoc.ActiveView.Refresh
End Sub

Public Sub CalcLandlockStats()
  Dim pFLayer As IFeatureLayer
  Dim pFClass As IFeatureClass
  Set m_pMxDoc = ThisDocument  'This is a module-level variable!
  Set pFLayer = m_pMxDoc.FocusMap.Layer(3)
  Set pFClass = pFLayer.FeatureClass
  Dim pQFilter As IQueryFilter
  Dim pFCursor As IFeatureCursor
  Set pQFilter = New QueryFilter
  pQFilter.WhereClause = "LANDLOCKED = 'Y'"
  Set pFCursor = pFClass.Search(pQFilter, True)
  Dim pFeature As IFeature
  Dim intPop As Long
  Dim intArea As Long
  Dim intCountries As Integer
  Set pFeature = pFCursor.NextFeature
  Do Until pFeature Is Nothing
      intPop = intPop + pFeature.Value(9)
      intArea = intArea + pFeature.Value(10)
      intCountries = intCountries + 1
      Set pFeature = pFCursor.NextFeature
  Loop
  Dim dblPopDensity As Double
  dblPopDensity = intPop / intArea

 MsgBox "Information on landlocked countries: " & vbCrLf & _
         "Number: " & intCountries & vbCrLf & _
         "Average Area: " & (intArea/intCountries) & vbCrLf & _
         "Population Density: " & dblPopDensity
End Sub

Private Sub UIButtonControl2_Click()
  Dim pCountryLayer As IFeatureSelection
  Dim pCountrySelection As ISelectionSet
  Dim pCountryCursor As IFeatureCursor
  Set m_pMxDoc = ThisDocument
  Set pCountryLayer = m_pMxDoc.FocusMap.Layer(3)
  Set pCountrySelection = pCountryLayer.SelectionSet
  pCountrySelection.Search Nothing, True, pCountryCursor

  Dim pCountry As IFeature
  Set pCountry = pCountryCursor.NextFeature
  If pCountry Is Nothing Then
      MsgBox "Please select a country"
      Exit Sub
  End If

  Dim pSpatialFilter As ISpatialFilter
  Dim pQueryFilter As IQueryFilter
  Set pSpatialFilter = New SpatialFilter
  Set pQueryFilter = pSpatialFilter
  pQueryFilter.WhereClause = "Population > 2000000"
  Set pSpatialFilter.Geometry = pCountry.Shape
  pSpatialFilter.SpatialRel = esriSpatialRelContains
  Dim pCityLayer As IFeatureLayer
  Dim pCityFClass As IFeatureClass
  Set pCityLayer = m_pMxDoc.FocusMap.Layer(0)
  Set pCityFClass = pCityLayer.FeatureClass
  Dim pCityCursor As IFeatureCursor
  Dim pCity As IFeature
  Dim intCount As Integer
  Set pCityCursor = pCityFClass.Search(pSpatialFilter, True)
  Set pCity = pCityCursor.NextFeature
  Do Until pCity Is Nothing
      intCount = intCount + 1
      Set pCity = pCityCursor.NextFeature
  Loop
  MsgBox "This country has " & intCount & " cities over 2 million"
End Sub
```

12

Symbolizing elements and layers

contents

Exercise 12: Symbolizing elements and layers

In this exercise, you will work with many of the ArcObjects required to symbolize graphic elements and feature layers. You will write code to create new symbols, colors, and color ramps, and then apply these objects to layers in the map. You will also learn to write code to save layers as layer files on disk, as well as how to programmatically add layers stored in a layer file to an ArcMap document.

Exercise shortcut

1. Open *ex12.mxd* from ..\IPAO\Maps. Test the *Basic Renderers* button. In the *frmRenderer* form, complete the *cmdRender_Click* event to symbolize the selected renderer if the *Simple* option is chosen. Use the RGB values the user picks in the form for the fill color.

2. Create a new UIButtonControl named *btnSaveLayer*. Code btnSaveLayer to save the selected layer as a .lyr file. If no layer is selected, disable the button.

3. Create a new UIButtonControl named *btnAddLayer*. Code btnAddLayer to prompt the user for a name of a .lyr file stored in C:\Student to add to ArcMap. If the user enters a non-existent layer, add an error indicating the layer doesn't exist.

Step 1: Create a fill symbol for displaying polygons

In this step, you will start ArcMap, open a map document, and write code for a user form. The form (which has already been created for you) applies one of three basic renderers to a polygon layer in the map. You will write code that creates a new fill symbol (FillSymbol object), set some of the symbol's properties (such as its color), and then apply the symbol to a polygon layer using a SimpleRenderer.

☐ Start *ArcMap*.

☐ Navigate to *C:\Student\IPAO\Maps* and open *ex12.mxd*.

☐ Start the *Visual Basic Editor*.

☐ From the *Project Explorer* window, under *Project* > *Forms*, open the form called *frmRenderer*.

Notice the controls contained on the form. The form will allow the user to select a polygon feature layer in the current map (from the combo box at the top), define the type of rendering desired for the layer (simple, unique values, or class breaks), and then apply the renderer to the layer to redraw it in the map.

☐ Double-click the *Render* command button on the form (cmdRender) to view the code for its Click event.

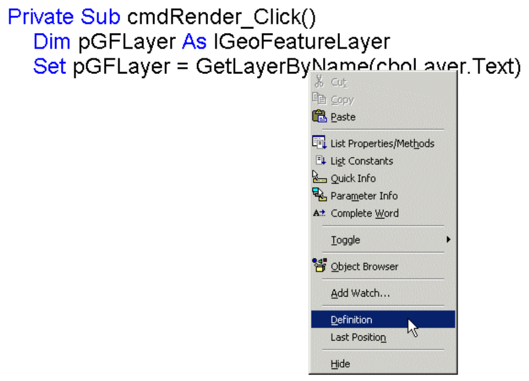
When the user clicks this button, a new renderer will be applied to the specified layer. The first step, therefore, is to reference the layer the user has selected.

☐ Examine the first two lines of code. This code is creating a reference to the chosen layer by passing the name (selected in the cboLayer combo box) to a function that has been written for you.

```
Dim pGFLayer As IGeoFeatureLayer
Set pGFLayer = GetLayerByName(cboLayer.Text)
```

You will browse the code in the GetLayerByName function to get an idea of how it works.

☐ Right-click the word *GetLayerByName* (in the code listed above). In the context menu that appears, choose *Definition* to jump to this function in the code editor.



```
Private Sub cmdRender_Click()
    Dim pGFLayer As IGeoFeatureLayer
    Set pGFLayer = GetLayerByName(cboLayer.Text)
```

You can use this technique to quickly jump to any sub or function procedure in your project (even if it's in another code module).

☐ Read through the code in the GetLayerByName function. How does the function get a reference to just the FeatureLayers in the map?

Notice that the code uses the Layers property on IMap to return layers from the active data frame. If none of the optional arguments for this property are specified, all layers in the map are returned. The code in your function, however, uses a unique identifier (UID) to return only layers of a specified type, in this case the unique ID feature layer (IGeoFeatureLayer). There are also unique IDs for Group Layers, Coverage Annotation Layers, Graphics Layers, etc. For a description and listing of additional UIDs you can use, search the ArcObjects Developer Help index for the IMap::Layers property.

☐ When you are done viewing the GetLayerByName function, return to the *Click* event procedure for *cmdRender*.

After the selected layer is referenced, a Select Case statement is used to construct the desired feature renderer. The Select Case is based upon the m_strOption variable, which is set every time the combo boxes are changed. You will complete the code that creates a simple renderer.

☐ You will begin by creating a new Symbol object to display the layer. Write the following code after the Case "Simple" statement.

```
Case "Simple"
  Dim pSymbol As ISimpleFillSymbol
  Set pSymbol = New SimpleFillSymbol
```

Because the frmRenderer automatically filters out the polygon feature classes, you have created a new SimpleFillSymbol to apply to the polygon feature classes. There are also SimpleMarkerSymbol and SimpleLineSymbol objects that can be used for points and lines respectively.

☐ Next, assign a style to your symbol, as shown below.

```
pSymbol.Style = esriSFSSolid
```

By default, a new symbol's color is black. You will assign the red, green, and blue values input on the form to create a new color for your symbol.

☐ Write the code below to create a new Color object. Because you will define the color using red, green, and blue values, you will create a new RgbColor.

```
Dim pColor As IRgbColor
Set pColor = New RgbColor
```

☐ Define the color using *one* of the methods below. You can either use the built-in Visual Basic RGB function to define the color, or set the red, green, and blue values individually.

```
pColor.RGB = RGB(cboRed.Value, cboGreen.Value, cboBlue.Value)
'-or-
pColor.Red = cboRed.Value
pColor.Green = cboGreen.Value
pcolor.Blue = cboBlue.Value
```

☐ Next, complete the code below to assign the new Color (pColor) to the new Symbol (pSymbol).

```
pSymbol.________ = pColor
```

☐ Finally, you will call another procedure that has been written for you to create and apply a SimpleRenderer with your symbol.

```
ApplySimple pGFLayer, pSymbol
```

The two required arguments for the ApplySimple sub are a (polygon) FeatureLayer and a fill symbol. If you want to view the code for this procedure, right-click the word *ApplySimple* and choose *Definition* from the context menu that appears.

☐ Test the form. First, close the *Visual Basic Editor* and bring *ArcMap* to the front of your display. Click the *Basic Renderers* button on the *Feature Layer Rendering* toolbar to launch the form.

☐ Try applying different colors to the various polygon layers in the map. Experiment with the other two renderers (UniqueValue and ClassBreaks).

STEP 2: SAVE A LAYER IN A LAYER FILE

There may be times when creating layer symbology “from the ground up” is your only option. It will probably be more common, however, (and a lot easier) for you to have your most frequently used layers stored on disk as layer files. Layer files contain all the same information you would define in ArcMap’s Layer Properties dialog, including the data source, symbology, and labeling options. A good strategy, therefore, may be to define layer properties interactively in ArcMap, then save them as layer files that you can load programmatically later. Adding a layer to ArcMap from a layer file takes only a few lines of code, and saves you the trouble of having to define each aspect of the layer’s display.

In this step, you will write code to save a layer in the map as a layer file on disk. Later, you will write code that loads the layers you’ve saved back to the map.

☐ Use the *ArcMap Customize Dialog* to create a new UIButtonControl. Name the control **`btnSaveLayer`**. Drag your button onto the *Feature Layer Rendering* toolbar, and give it a new icon.

☐ Right-click *btnSaveLayer* and choose *View Source* to jump to the code for its Click event procedure.

When the button is clicked, whichever layer is selected in the document will be saved to disk.

☐ Begin your code by referencing the IMxDocument interface on the current document.

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
```

☐ Now get a reference to the currently selected layer in the document.

```
Dim pLayer As ILayer
Set pLayer = _______.SelectedLayer
```

☐ Write the code below to create a new GxLayer object. A GxLayer is the ArcObjects equivalent of a layer file (*.lyr).

```
Dim pGxLayer As IGxLayer
Set pGxLayer = New GxLayer
```

Before assigning the selected layer (pLayer) to the layer file (pGxLayer), you need to specify the pathname for the new file. The Path property is on the IGxFile interface, which requires you to Queryinterface.

☐ Complete the code below to reference the IGxFile interface on pGxLayer.

```
Dim pGxFile As IGxFile
Set pGxFile = _________ '-QueryInterface (QI)
```

☐ Assign a pathname for the new layer file, as shown below.

```
pGxFile.Path = "C:\Student\" & pLayer.Name & ".lyr"
```

The code above uses the layer's name (as it appears in the map) as the output file name in your *Student* folder. The *lyr* extension must also be added to the file for ArcGIS to recognize it as a layer file.

☐ Specify the layer to save by setting pGxLayer's Layer property.

```
Set pGxLayer.Layer = pLayer
```

☐ Complete the code below to commit the layer file to disk. Make sure to tell the user where the file was saved.

```
________.Save
```

```
MsgBox pLayer.Name & " has been saved to " & vbNewLine & _
                 pGxFile.Path
```

Because your button requires a selected layer in the ArcMap Table of Contents, you will add code to enable and disable the button according to whether or not the document has a selected layer.

☐ As a final step, add the following code to the button's *Enabled* event procedure.

```
Private Function btnSaveLayer_Enabled() As Boolean
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument
    btnSaveLayer_Enabled = Not pMxDoc.SelectedLayer Is Nothing
End Function
```

☐ Close the Visual Basic Editor.

☐ Test your code. Bring *ArcMap* to the front of your display and make sure you have a selected layer in the Table of Contents. Click your button to save the layer.

☐ Click the *Add Data* button and navigate to your *Student* directory. Verify that the layer file was written there. Add the layer to ArcMap.

In the next step, you will create a new UIButtonControl that adds a layer to ArcMap from a layer file.

STEP 3: ADD A LAYER FROM A LAYER FILE

In a previous section, you wrote code to access a dataset on disk and then added it to ArcMap as a new layer. As you recall, there was a lot of code required to access data on disk, and even more is necessary if you want to change the default single-symbol rendering (simple renderer) for the new layer. Because a layer file contains all of this information (and more), adding layers from predefined layer files can be a more concise solution.

In this step, you will write code to access a layer file on disk, then add its layer to ArcMap.

☐ Use the *ArcMap Customize Dialog* to create a new UIButtonControl. Name the control **`btnAddLayer`**. Drag your button onto the *Feature Layer Rendering* toolbar, and give it a new icon.

☐ Right-click *btnAddLayer* and choose *View Source* to jump to the code for its *Click* event procedure.

When the button is clicked, a layer from a layer file will be added to the focus map.

☐ Begin your code by creating a new GxLayer object. You will use the IGxFile interface on the GxLayer so you can specify its file path.

```
Dim pGxFile As IGxFile
Set pGxFile = New GxLayer
```

☐ Next, prompt the user for the layer name using an InputBox statement.

```
Dim strLayerName As String
strLayerName = InputBox("What's the name of the layer file" & vbNewLine & _
"(C:\Student\____.lyr)?", "Enter a lyr file", "MyLayer")
```

The code above will prompt the user for the layer name only. You will construct the full path name from the layer name they enter.

☐ Construct the file's pathname by adding the path to the Student folder and a .lyr extension.

```
strLayerName = "C:\Student\" & strLayerName & ".lyr"
```

☐ Assign the string created above to the GxLayer's Path property.

```
pGxFile.Path = strLayerName
```

By assigning the GxLayer's path property, you are indicating where the layer file lives on disk. In order to access the layer object stored inside the file, you will need to use the IGxLayer interface.

☐ QueryInterface pGxFile for the IGxLayer interface.

```
Dim pGxLayer As IGxLayer
Set pGxLayer = pGxFile
```

There is only one property on the IGxLayer interface, *Layer*. You will use the Layer property to return the layer stored in the layer file.

Because you relied on the user to simply type the name of the layer file as input, you should not assume that the file will be found. What if the user spelled the name wrong, or typed the name of a file that doesn't exist? The code you've written to this point will work without error, even if the filename provided doesn't exist. Before attempting to actually reference the contents of the file, however, you should be prepared to handle errors that may occur.

☐ Write the code below to jump to an error handler if the file is not found.

```
On Error GoTo NoFile
```

This statement tells Visual Basic that if an error occurs below this line, immediately jump to a section of the procedure marked with the tag *NoFile*. The code below the NoFile tag will respond to the error. You will write the NoFile error handler at the end of this procedure.

☐ Get the layer from the layer file.

```
Dim pLayer As ILayer
Set pLayer = pGxLayer.Layer
```

If an erroneous layer file name was provided by the user, execution will jump to the NoFile error handler at this line.

☐ Add the layer to the document's active data frame.

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
pMxDoc.FocusMap.AddLayer pLayer
```

If the layer file was correctly specified by the user, the layer will now appear in ArcMap.

☐ To ensure that the error handling code is not executed after successfully adding the layer, exit the sub procedure.

```
Exit Sub
```

☐ Write the error handling routine for the procedure.

```
NoFile:
    MsgBox "Layer file not found, exiting ...", vbCritical
```

The error handling code above is very simple. By providing error handlers like these, however, you can exit procedures more gracefully and prevent users from seeing "ugly" run time errors.

☐ Test your code. Bring *ArcMap* to the front of your display. Make sure that you have a layer file stored in your Student folder (if you do not, use the button from the previous exercise step to create one). Try providing a bad file name to test the code's ability to handle this error.

Challenge: Experiment with scale dependent rendering

In addition to the three basic renderers you worked with in the first step of the exercise (Simple, UniqueValue, and ClassBreaks), ArcObjects provides a ScaleDependentRenderer. This renderer allows you to provide different symbology for a layer depending on the scale at which it is being viewed.

Scale dependent rendering is not possible from the ArcMap user interface, it can only be accomplished by writing ArcObjects code. Unfortunately, writing code to produce and apply a ScaleDependentRenderer can be quite lengthy, as it involves the creation of several renderers to be contained within the ScaleDependentRenderer. In this challenge step, you will use a form that has been created for you to experiment with scale dependent layer rendering.

☐ Click the *Scale Dependent Renderer* button on the *Feature Layer Rendering* toolbar to launch the form.

☐ Select a layer in the form's combo box.

☐ Make sure the layer you selected is checked on, and zoom in close to a portion of the layer.

☐ Right-click the layer and use the layer properties dialog to apply a distinct symbology. (Unique values is suggested)

☐ Click the *Add Current Renderer* button on the form. The current symbology (renderer) will be used at the current scale (and closer). Information about the renderer and scale is added to the form.

☐ Zoom out a little on the layer. Use the layer properties dialog again to apply different symbology.

☐ Click the *Add Current Renderer* button to apply the renderer for this scale.

☐ Repeat the above process for as many renderers as you would like to add. When you have defined a renderer for each scale you desire, click the *Apply* button.

☐ Test by zooming in and out on the layer. As you zoom past the scale thresholds you've defined, the layer rendering should change.

> *Note:* If you zoom out past your last (smallest) scale threshold, the layer will not render.

Exercise end

SOLUTIONS

```
'frmRenderer
Private Sub cmdRender_Click()
    Dim pGFLayer As IGeoFeatureLayer
    Set pGFLayer = GetLayerByName(cboLayer.Text)

    Select Case m_strOption
    Case "Simple"
'***Step-1: code here to create a new fill symbol and color
           Dim pSymbol As ISimpleFillSymbol
           Set pSymbol = New SimpleFillSymbol
           pSymbol.Style = esriSFSSolid

           Dim pColor As IRgbColor
           Set pColor = New RgbColor
           pColor.RGB = RGB(cboRed.Value, cboGreen.Value, cboBlue.Value)
'     -or-
'           pColor.Red = cboRed.Value
'           pColor.Green = cboGreen.Value
'           pColor.Blue = cboBlue.Value
           pSymbol.Color = pColor
'-pass the layer and symbol to the ApplySimple sub
        ApplySimple pGFLayer, pSymbol
    Case "Unique"
'-call the ApplyUniqueValue sub ...
          ApplyUniqueValue pGFLayer, cboUniqueVals.Text
    Case "Breaks"
'-call the ApplyClassBreaks sub ...
          ApplyClassBreaks pGFLayer, cboNumericVals.Text, CLng(cboBreaks.Value)
    Case Else
    End Select
End Sub


'ThisDocument module (Project)

'***Step-2: button to save the selected layer to disk
Private Sub btnSaveLayer_Click()
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument

    Dim pLayer As ILayer
    Set pLayer = pMxDoc.SelectedLayer

    Dim pGxLayer As IGxLayer
    Set pGxLayer = New GxLayer

    Dim pGxFile As IGxFile
    Set pGxFile = pGxLayer

    pGxFile.Path = "C:\Student\" & pLayer.Name & ".lyr"

    Set pGxLayer.Layer = pLayer

    pGxFile.Save
    MsgBox pLayer.Name & " has been saved to " & vbNewLine & _
                 pGxFile.Path
End Sub
```

```
'-Only enable the button if there is a selected layer ...
Private Function btnSaveLayer_Enabled() As Boolean
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument
    btnSaveLayer_Enabled = Not pMxDoc.SelectedLayer Is Nothing
End Function

'***Step-3: button to add a layer from a layer file
Private Sub btnAddLayer_Click()
    Dim pGxFile As IGxFile
    Set pGxFile = New GxLayer

    Dim strLayerName As String
    strLayerName = InputBox("What's the name of the layer file" & vbNewLine & _
                       "(C:\Student\____.lyr)?", "Enter a lyr file", "MyLayer")
    strLayerName = "C:\Student\" & strLayerName & ".lyr"

    pGxFile.Path = strLayerName

    Dim pGxLayer As IGxLayer
    Set pGxLayer = pGxFile

On Error GoTo NoFile

    Dim pLayer As ILayer
    Set pLayer = pGxLayer.Layer

    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument
    pMxDoc.FocusMap.AddLayer pLayer

    Exit Sub
NoFile:
    MsgBox "Layer file not found, exiting ...", vbCritical
End Sub
```

13

Working with layout elements (Optional)

contents

EXERCISE 13: WORKING WITH LAYOUT ELEMENTS

In this exercise, you will write some procedures to help you add elements to a page layout. You will begin by writing a sub procedure that takes an element and page coordinates as inputs, then adds the element to the specified area on the page. You will then write code that adds the north arrow to the layout.

In the challenge step, you will call a procedure that exports your layout as either a JPEG or an Adobe Acrobat file (*.pdf).

EXERCISE SHORTCUT

1. Add a sub procedure with two parameters for objects that support IElement and IGeometry. The procedure will use these parameters to determine the position on the page layout.

2. Create a text element that displays your name and the date to be positioned five inches from the left edge and one inch above the bottom of the page layout. Use the procedure from Step 1 to add the text element.

3. If you are not familiar with the Page Layout Elements, view the elements in the Style Gallery from the ArcMap interface.

4. Create a north arrow to be positioned two inches from the left edge and two inches from the bottom of the page layout. Use the procedure from Step 1 to add the north arrow.

5. Create a sub procedure that exports the layout as a .jpeg file.

STEP 1: WRITE A SUB PROCEDURE THAT ADDS AN ELEMENT TO THE LAYOUT

Writing modular code means writing subs and functions that can be called from other procedures and modules and even used in other projects. The main advantage of modular code, of course, is that you do not have to constantly *reinvent the wheel* by writing the same basic functionality each time you need it.

To produce modular code, you need to keep the code generic enough that it can be flexible. By allowing a programmer more freedom in defining input parameters when they use your procedure, you can create a more all-purpose sort of procedure. Rather than writing one procedure that works with points, one with lines, and one with polygons, for example, you could write a single procedure that takes *any* geometry as input (IGeometry), then uses a Select Case statement to perform the proper action.

In this step, you will produce a sub procedure that adds an element to the page layout. Because there are several types of elements that can be added to a map, your procedure will take IElement as an input parameter because *all* elements support this interface. Likewise, to define the element's position on the layout page, you will use an IGeometry parameter.

☐ Start *ArcMap* and open a new map.

☐ Add some layers to the map (it does not matter which ones).

☐ Start the *Visual Basic Editor*.

☐ Insert a new module (standard module) and name it **`PageElements`** (use the *Properties* window to change the name).

☐ At the top of the *PageElements* module (*General Declarations*), declare the following module-level variables with the Private keyword.

```
Private m_pMxDoc As IMxDocument
Private m_pPageLayout As IPageLayout
Private m_pGContainer As IGraphicsContainer
```

☐ Define a new sub procedure in the *PageElements* module, as shown below.

```
Private Sub AddElement(AnElement As IElement, PagePosition As IGeometry)
```

The AddElement procedure will take two parameters: an element, and some geometry to define the element's position on the page (a point or an envelope, for example).

☐ Begin the procedure by setting some of the module-level variables you declared above.

```
Set m_pMxDoc = ThisDocument
Set m_pPageLayout = m_pMxDoc.PageLayout
```

☐ Complete the code below to define the element's position (on the page) with the geometry passed in to the procedure.

```
AnElement.__________ = PagePosition
```

☐ Set the GraphicsContainer variable (m_pGContainer) by using QueryInterface on the PageLayout variable (m_pPageLayout).

☐ Call the AddElement method on m_pGContainer. Add the (passed in) element as the first item in the graphics container.

Your procedure is nearly complete. At this point, your code has successfully added the new element to the page. You will need to refresh the page, however, before the element will be displayed.

☐ Complete the chain of code below to refresh the display.

```
m_pMxDoc.______________.Refresh
```

Your procedure is now complete, but you will not be able to test it until you write some client code that calls it. In the next step, you will write a procedure that produces a new element to add to the page, then calls your AddElement procedure, passing in the element and the desired page position.

STEP 2: CALL YOUR PROCEDURE TO ADD TEXT TO THE LAYOUT

The AddElement procedure you wrote above will take any element (any object that supports the IElement interface) and add it to the layout. In this step, you will write a simple procedure that calls AddElement to place some text on the layout.

☐ Define a new procedure in the *PageElements* module, as shown below.

```
Public Sub AddText( )
```

☐ Declare a new variable called **pTextElement** that points to the ITextElement interface.

☐ Set pTextElement equal to a new TextElement object.

☐ Assign the following text to the element's Text property.

```
"Map produced by <YourName> on " & Date
```

NOTE: <*YourName*> should be your name (check your name tag if you don't remember it ;-).

☐ Now make a new point to define the center point of the text element on the page, as shown below.

```
Dim pPoint As IPoint
Set pPoint = New Point
```

☐ Assign the point's X and Y properties to center the element five inches from the left edge of the page and one inch above the bottom.

☐ Pass pTextElement and pPoint to the AddElement procedure.

You are now ready to test the AddText and the AddElement procedures.

☐ Bring *ArcMap* to the front of your display and make sure the map is in *Layout* view.

☐ Choose *Tools* > *Macros* > *Macros* to launch the Macros dialog.

☐ Run the *AddText* procedure.

You should see your new text appear near the bottom of the layout page.

☐ Hover over the center of the element with your mouse and verify (in the ArcMap status bar) that its page coordinates are approximately 5, 1.

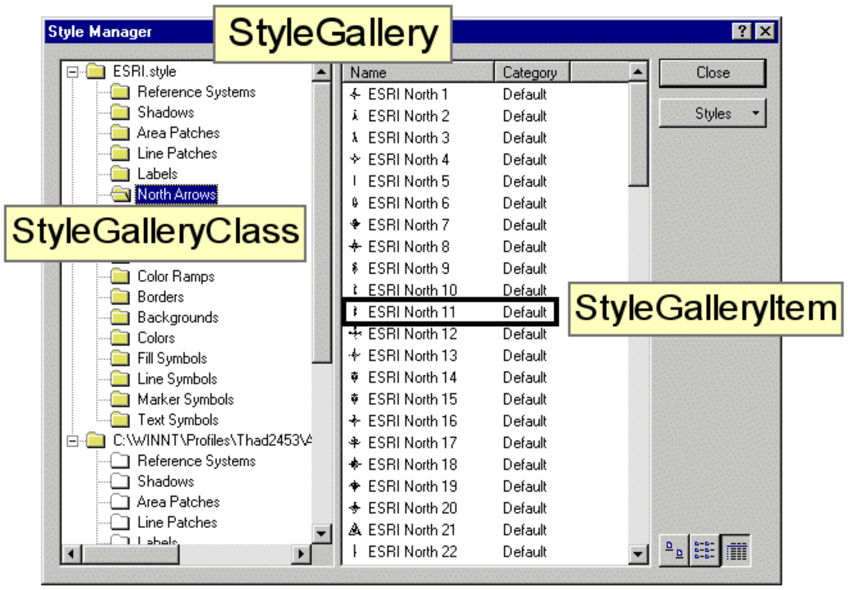
STEP 3: EXAMINE THE STYLE GALLERY

In this step, you will examine the Style Gallery. It is important to understand the ArcMap user interface before programming with it.

☐ Bring *ArcMap* to the front of your display. From the ArcMap *Tools* menu, choose *Styles* > *StyleManager*.

The ArcMap style manager dialog opens.

☐ Expand the listing for *ESRI.style*. Explore the contents of the Style Manager. Take note of the objects that you will work with in your code, as shown in the graphic below.



Name	Category
ESRI North 1	Default
ESRI North 2	Default
ESRI North 3	Default
ESRI North 4	Default
ESRI North 5	Default
ESRI North 6	Default
ESRI North 7	Default
ESRI North 8	Default
ESRI North 9	Default
ESRI North 10	Default
ESRI North 11	Default
ESRI North 12	Default
ESRI North 13	Default
ESRI North 14	Default
ESRI North 15	Default
ESRI North 16	Default
ESRI North 17	Default
ESRI North 18	Default
ESRI North 19	Default
ESRI North 20	Default
ESRI North 21	Default
ESRI North 22	Default

Notice how items in the style manager are organized. There are several available styles to choose from ("ESRI.style", "Civic.style", "Crime.style", etc.). These styles are saved as files on disk (in the ArcGIS installation directory). Each style has a set of style classes; such as scalebars, north arrows, and color ramps. Individual style items are organized into their appropriate class, and are described with a category and a name. All of these objects are accessible using ArcObjects code, as illustrated by the function that has been written for you.

STEP 4: CALL THE GETSTYLEITEM FUNCTION TO CREATE A NORTH ARROW

You will now write code that takes advantage of both the GetStyleItem function and the AddElement procedure you wrote earlier. To start, you will create a new MapSurroundFrame that contains a north arrow from the style gallery. You will then add the element (a MapSurroundFrame is a subclass of Element) to the layout, specifying its position on the page.

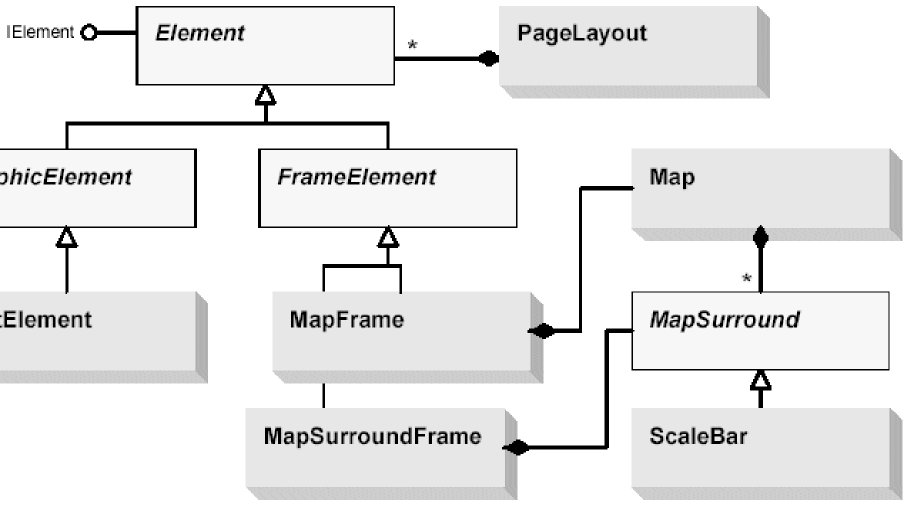
☐ Define the sub procedure below in your *PageElements* module.

```
Public Sub AddNorthArrow ( )
```

☐ Begin by setting the m_pMxDoc variable equal to the current document.

```
Set m_pMxDoc = ThisDocument
```

☐ Examine the simplified object model diagram below.



Notice that the Map class is composed of (multiple) MapSurrounds. MapSurrounds are things such as north arrows, legends, and scale bars. To display MapSurrounds on a PageLayout, ArcMap uses MapSurroundFrames. A MapSurroundFrame (as the name implies) is nothing more than a container for a MapSurround. Notice that MapSurroundFrame is a subclass of Element and that a PageLayout is composed of (multiple) elements.

In the following code, you will create objects representing MapSurroundFrame and MapFrame. The MapSurroundFrame will hold a north arrow. The MapFrame will reference a particular map, in this case the active data frame, so the north arrow will know which map to attach itself to. The MapFrame object will have the ability to create a SurroundFrame, which will reference the NorthArrow.

☐ Declare the variable below to contain a MapSurroundFrame.

```
Dim pNorthSurroundFrame As IMapSurroundFrame
```

☐ Set pNorthSurroundFrame equal to a new MapSurroundFrame.

☐ Declare a variable as IMapFrame that will point to a specific map's frame.

```
Dim pMapFrame As IMapFrame
```

☐ Declare a variable to represent a UID.

```
Dim pUID As New UID
```

The UID coclass can be used to represent the GUID (Global Unique Identifier) of an object. Each individual object has a GUID. In this case, the UID variable you declared will represent a North Arrow. The UID will then be assigned to the MapSurroundFrame.

☐ Set the pID equal to an esri NorthArrow object.

```
pUID.Value = "esriCarto.MarkerNorthArrow"
```

NOTE: esriCarto represents the library in which the MarkerNorthArrow object is stored.

☐ Set m_pGContainer equal to the graphics container of the page layout.

```
Set m_pGContainer = m_pMxDoc.PageLayout
```

☐ Set pMapFrame equal to the Frame of the active data frame.

```
Set pMapFrame = m_pGContainer.FindFrame(m_pMxDoc.FocusMap)
```

☐ Use the CreateSurroundFrame method from the map's frame to create a SurroundFrame for the north arrow you just created.

```
Set pNorthSurroundFrame = pMapFrame.CreateSurroundFrame(pUID, Nothing)
```

If you wanted to change the symbol of your north arrow, you would create a new MarkerNorthArrow object to represent what is inside the MapSurroundFrame. You would then be able to set the MarkerSymbol of the MarkerNorthArrow. You could optionally let the user pick the north arrow with the NorthArrowSelector object.

You are now ready to add the north arrow to the layout. You will call your AddElement procedure to place the MapSurround on the layout page. Before you do, however, you need to create some geometry to define where the north arrow should go. Instead of using a Point object like you did for your text, your code below will create an Envelope to define a box in which to place the north arrow.

☐ Declare a new variable called **`pEnv`** as a pointer to the IEnvelope interface.

☐ Set pEnv equal to a new Envelope.

☐ Declare a new variable called **`pPoint`** as a pointer to the IPoint interface.

☐ Set pPoint equal to a new Point object.

☐ Define pPoint’s coordinates as two inches from the left edge of the page and two inches above the bottom of the page.

☐ Set pEnv’s lower-left corner equal to pPoint.

☐ Redefine pPoint’s coordinates as three inches from the left edge of the page and three inches above the bottom of the page.

☐ Set pEnv’s upper-right corner equal to pPoint.

Finally, you will add the element to the PageLayout’s GraphicsContainer.

☐ Call the AddElement procedure, passing in the MapSurroundFrame (pNorthSurroundFrame) and the Envelope (pEnv).

☐ Bring *ArcMap* to the front of your display and make sure the map is in *Layout* view.

☐ Save the map to your *Student* directory.

☐ Run the *AddNorthArrow* macro from the *Tools* menu. Was a new north arrow added to the map?

If you want to experiment with exporting or printing your layout, proceed to the Challenge step. Otherwise, export the PageElements module and save your map to your student directory.

CHALLENGE: CALL A FUNCTION TO EXPORT THE LAYOUT

Using ArcObjects code, it’s possible to print or export your layout (or map) to a variety of output formats. The ArcObjects for printing and exporting can be found on the Output object model diagram.

In this challenge step, you will write code that calls a sub procedure that’s already been written for you. The procedure will export your layout to a file on disk in the format you indicate (JPEG or Adobe Acrobat) and at a specified resolution.

☐ Import the following module into your project: *C:\Student\IPAO\Results\Ex13\Output.bas*.

☐ Locate the *ExportLayout* sub procedure. The definition for this procedure is listed below.

```
Public Sub ExportLayout(Format As String, FileName As String, DPI As Integer)
```

☐ Write a macro that calls this sub procedure to export the layout as a jpeg in the student directory. Experiment with different formats and resolutions (DPI).

☐ The module also contains a procedure for printing, shown below.

```
Public Sub PrintLayout(ToFile As Boolean, Optional FileName As String)
```

☐ Write a macro that prints the layout to a file in the student directory or to the default printer (if one is available).

EXERCISE END

SOLUTIONS

```
'PageElements module--
Private Sub AddElement(AnElement As IElement, PagePosition As IGeometry)
  Set m_pMxDoc = ThisDocument
  Set m_pPageLayout = m_pMxDoc.PageLayout
  AnElement.Geometry = PagePosition
  Set m_pGContainer = m_pPageLayout
  m_pGContainer.AddElement AnElement, 0
  m_pMxDoc.ActiveView.Refresh
End Sub

Public Sub AddText()
  Dim pTextElement As ITextElement
  Set pTextElement = New TextElement
  pTextElement.Text = "Map produced by Thad on " & Date

  Dim pPoint As IPoint
  Set pPoint = New Point
  pPoint.X = 5
  pPoint.Y = 1
  AddElement pTextElement, pPoint
End Sub

Public Sub AddNorthArrow()
    Set m_pMxDoc = ThisDocument
    Dim pNorthSurroundFrame As IMapSurroundFrame
    Set pNorthSurroundFrame = New MapSurroundFrame
    Dim pMapFrame As IMapFrame
    Dim pUID As New UID
     pUID.Value = "esriCarto.MarkerNorthArrow"
     Set m_pGContainer = m_pMxDoc.PageLayout
     Set pMapFrame = m_pGContainer.FindFrame(m_pMxDoc.FocusMap)
    Set pNorthSurroundFrame = pMapFrame.CreateSurroundFrame(pUID, Nothing)

    Dim pEnv As IEnvelope
    Set pEnv = New Envelope
    Dim pPoint As IPoint
    Set pPoint = New Point
    pPoint.PutCoords 2, 2
    pEnv.LowerLeft = pPoint
    pPoint.PutCoords 3, 3
    pEnv.UpperRight = pPoint

    AddElement pNorthSurroundFrame, pEnv
End Sub

'****Challenge
Public Sub ExportLayout(Format As String, FileName As String, DPI As Integer)
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument

    Dim pLayout As IActiveView
    Set pLayout = pMxDoc.PageLayout

    Dim rectOut As tagRECT
    rectOut = pLayout.exportFrame

    Dim pEnv As IEnvelope
    Set pEnv = New Envelope
    pEnv.PutCoords rectOut.Left, rectOut.Top, rectOut.Right, rectOut.bottom

    Dim pExporter As IExporter
    If Format = "jpeg" Then
        Set pExporter = New JpegExporter
    Else
        Set pExporter = New PDFExporter
    End If

    pExporter.ExportFileName = FileName
    pExporter.PixelBounds = pEnv
    pExporter.Resolution = DPI
'Recalc the export frame to handle the increased number of pixels
    Set pEnv = pExporter.PixelBounds

    Dim xMin As Double, yMin As Double
    Dim xMax As Double, yMax As Double
    pEnv.QueryCoords xMin, yMin, xMax, yMax

    rectOut.Left = xMin
    rectOut.Top = yMin
    rectOut.Right = xMax
    rectOut.bottom = yMax

    'Do the export
    Dim hDc As Long
    hDc = pExporter.StartExporting

    pLayout.Output hDc, DPI, rectOut, Nothing, Nothing
    pExporter.FinishExporting

    MsgBox "Export complete!", vbInformation
End Sub

Public Sub PrintLayout(ToFile As Boolean, Optional FileName As String)
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument

    Dim pPageLayout As IPageLayout
    Set pPageLayout = pMxDoc.PageLayout

    Dim pMxApp As IMxApplication
    Set pMxApp = Application
    Dim pPrinter As IPrinter
    Set pPrinter = pMxApp.Printer

    Set pPrinter.Paper = pMxApp.Paper

    If ToFile = True Then
        pPrinter.PrintToFile = FileName
        MsgBox "Layout printed to " & FileName, vbInformation
        Exit Sub
    End If

    Dim pPrintEnv As IEnvelope
    Set pPrintEnv = New Envelope
    pPageLayout.Page.GetDeviceBounds pPrinter, 0, 0, pPrinter.Resolution, pPrintEnv
    Dim rectOut As tagRECT
    rectOut.Left = pPrintEnv.xMin
    rectOut.Top = pPrintEnv.yMin
    rectOut.Right = pPrintEnv.xMax
    rectOut.bottom = pPrintEnv.yMax

    Dim pPageBounds As IEnvelope
    Set pPageBounds = New Envelope
    pPageLayout.Page.GetPageBounds pPrinter, 0, 0, pPageBounds

    Dim hDc As OLE_HANDLE
    hDc = pPrinter.StartPrinting(pPrintEnv, 0)

    Dim pActiveView As IActiveView
    Set pActiveView = pPageLayout
    pActiveView.Output hDc, pPrinter.Resolution, rectOut, pPageBounds, Nothing

    pPrinter.FinishPrinting
    MsgBox "Printing complete!", vbInformation, pPrinter.DriverName
End Sub
```

14

Using tools

contents

EXERCISE 14A: CREATE A TOOL TO DRAW POINT GRAPHICS

In this exercise, you will create an ArcMap tool to draw point graphics on the display. You will take advantage of the various event procedures associated with a tool control including MouseDown, MouseUp, and KeyDown.

EXERCISE SHORTCUT

1. Open *ex14a.mxd* from ../IPAO/Maps. Declare four modular level variables pointing to IPoint, ISimpleMarkerSymbol, IElement, and IRgbColor. At the bottom of the *UIToolControl1_Select* event, initialize the following variables for the applications current display: m_pMxApp, m_pMxDoc and m_pDisplay.

2. Initialize the variable you set to IPoint in the tool's *MouseDown* event to the location the user clicks on the screen display. Use the DisplayTransformation method to convert from pixel to map coordinates.

3. Using the variables you declared in Step 1, create a new marker element and add it to the GraphicsContainer in the *MouseDown* event.

4. In the *MouseUp* event, refresh the display with partial refresh. Expand the envelope of the point to determine the area to be refreshed.

5. In the *KeyDown* event, change the color of the marker symbol added to the graphics container based on the key the user presses on the keyboard.

STEP 1: INITIALIZE OBJECT VARIABLES IN A TOOL'S SELECT EVENT PROCEDURE

You will need to use a tool control for any application that requires user interaction with the display. Tool controls have several event procedures that can facilitate capturing mouse and keyboard input. Because tools generally consist of several event procedures working together, it is usually a good idea to declare the object variables used by the tool as module-level variables in the General Declarations. You will begin your code by declaring several object variables in General Declarations, then initializing (setting) these variables in the tool's Select event procedure.

☐ Start *ArcMap*.

☐ Open *ex14a.mxd* in *C:\Student\ipao\maps*.

☐ Right-click on the tool with the *pushpin* icon and choose to *View Source* .

☐ Navigate to the *General Declarations* section of the tool's code module. Notice that several object variables have already been declared for you here.

☐ Declare the following variables with the Private keyword.

```
m_pPoint As IPoint
m_pSym As ISimpleMarkerSymbol
m_pElem As IElement
m_pColor As IRgbColor
```

These variables have a module-level scope, meaning they can be referenced by any sub procedure inside this module (ThisDocument under project). Remember that module-level variables are declared in General Declarations, but they must be initialized (set) inside of an event procedure. It is very common, therefore, to initialize module-level variables in a tool's Select event procedure.

Question 1: Navigate to the tool's Select event procedure. When will this code execute?

____

☐ Notice that several variables are being initialized here. At the bottom of the procedure, initialize the variables shown below.

```
Set m_pMxApp = Application
Set m_pMxDoc = ThisDocument
Set m_pDisplay = m_pMxApp.Display
```

Each event procedure for this tool will now be able to use these module-level variables. Not only is using module variables more convenient than declaring the same variables in every procedure, but it also allows you to reference the same object from different event procedures.

STEP 2: CAPTURE USER POINTS WITH A TOOL'S MOUSEDOWN EVENT PROCEDURE

When a user clicks the mouse button, two events actually get fired: MouseDown and MouseUp. You will write code for both of these events. First, you will code the tool's MouseDown event to capture a point.

☐ Navigate to the tool's *MouseDown* event procedure.

Question 2: Look at the parameter list for MouseDown. What are the four parameters passed into this procedure? _________________________________
Where are they passed in from? ______________________________________

Notice that the last two parameters are the x- and y-coordinates of the user's mouse click. These coordinates, however, are in pixels, and must first be converted to map units in order to create a point. Fortunately, the IDisplayTransformation interface on the ScreenDisplay class "knows" how to transform a point from pixels into a point in the current map units.

☐ To convert the pixel coordinates to map units, use the ToMapPoint method on IDisplayTransformation. Store the returned point in the variable m_pPoint, as shown below.

```
Set m_pPoint = m_pDisplay.DisplayTransformation.ToMapPoint(x,y)
```

There are several event procedures that have x- and y-coordinates passed to them from the mouse. These coordinates are always in pixels (display units) and must be converted to map units (in most instances) using one of the methods on IDisplayTransformation. Other tool event procedures that use mouse locations in pixels are MouseUp and MouseMove.

STEP 3: MAKE AN ELEMENT TO REPRESENT THE POINT, THEN ADD IT TO THE DISPLAY

In this step, you will continue coding the MouseDown event procedure. Now that you have a point that is in map units, you can place this point on the display as a graphic. To place a graphic on the display, you will first get the active view's IGraphicsContainer interface. You will then add the point (as a graphic element) to the map's graphics container.

Continue your code in the MouseDown procedure below the line that set m_pPoint.

☐ Set the variable m_pElem equal to a New MarkerElement.

☐ Assign m_pElem's Geometry property equal to the point you transformed to map units above (m_pPoint).

Next, you will write code that sets the new element's marker symbol. In the tool's Select event procedure, two module-level variables that you will use were already initialized: m_pColor (an RgbColor object) and m_pSym (a SimpleMarkerSymbol object).

☐ Assign m_pSym's Color property equal to m_pColor.

To assign the symbol to the element, you need to query interface for the IMarkerElement interface, which has the Symbol property.

☐ Dim a new variable called pMElem as a pointer to the IMarkerElement interface.

☐ Set pMElem equal to m_pElem.

☐ Assign pMElem's Symbol property with m_pSym.

Elements that you may add to the graphics container are things such as points, lines, and polygons, as well as text or picture elements. These elements behave like graphics on the display. Rather than simply being painted on the screen, they can be selected and moved, resized, or deleted.

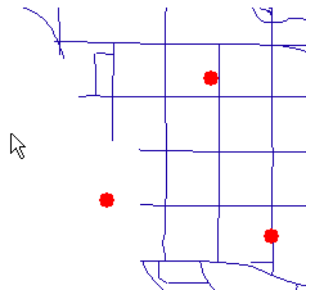
☐ To add an element to the map, use the AddElement method on the ActiveView's IGraphicsContainer interface, as shown below.

```
m_pMxDoc.ActiveView.GraphicsContainer.AddElement m_pElem, 0
```

Question 3: Notice the arguments for the AddElement method. The first argument is the element to add to the graphics container. What is the second argument?

____

Question 4: Test your tool. Move ArcMap to the front of the display, select your tool, then click a few times on the display. Did it work as you expected? ________



Question 5: Refresh the display, either by clicking the Refresh button (lower left of the display) or by resizing the application. Do the points now appear? __________

After you draw or add elements to a map's graphics container, you need to refresh the display or the user will not see the changes. In the next step, you will write code that refreshes the portion of the display where graphics are added.

STEP 4: PARTIALLY REFRESH THE DISPLAY IN THE MOUSEUP EVENT PROCEDURE

There are two methods on the IActiveView interface that can be used to refresh the display. The Refresh method will redraw the entire display, which is not very efficient if you only need to refresh the area covered by a small graphic. For the present purpose, the PartialRefresh method is better suited. The PartialRefresh method has three arguments: a constant specifying what should be redrawn (geography, graphics, selection, etc.), a pointer to data to refresh, and an envelope indicating the portion of the display to refresh.

☐ Return to the *Visual Basic Editor*.

☐ Navigate to the tool's *MouseUp* event procedure. Notice that the same arguments that were passed to MouseDown are passed into the MouseUp procedure (button, shift, x,y).

You will need to define a portion of the display for ArcMap to refresh.

☐ Declare the following variable with the Dim statement:

```
pEnv As IEnvelope
```

☐ Set pEnv equal to m_pElem's Geometry's Envelope.

All geometry has an envelope, which is simply a bounding rectangle. For point geometry, this envelope is empty (it has a height and width of 0). If you were to refresh with a point's (empty) envelope, ArcMap would assume that you want to refresh the entire display. Next, you will expand the envelope to cover the area where the graphic was just drawn.

☐ On pEnv, use the Expand method. For the three required parameters, give a height of 200, a width of 200, and specify false to indicate that these are not proportional values (i.e., percentages).

☐ Finally, write the code below to perform the partial refresh of the display, specifying Nothing as the data pointer.

```
m_pMxDoc.ActiveView.PartialRefresh esriViewGraphics, Nothing, pEnv
```

☐ Test the tool. Bring *ArcMap* to the front of your display.

☐ Make sure to unselect, then select your tool (you need to reset the variables declared in the Select event procedure).

☐ Click a few times on the display. New points should appear on the display as soon as you click.

☐ Click on the display and hold the mouse button down. Notice that the point doesn't appear.

☐ Keep holding the mouse button down and move your cursor off the display. Now, release the mouse button.

Notice how the point is defined when (and where) the mouse button is pressed (MouseDown), then refreshed when the button is released (MouseUp).

Step 5: Set marker color using the KeyDown event procedure

In this step, you will write code that will allow the user to change the marker color by pressing keys on the keyboard while the tool is selected.

☐ Return to the *Visual Basic Editor*.

☐ Navigate to the tool's *KeyDown* event procedure.

Notice the arguments that are passed to this procedure, keyCode and shift, both of which are long data type. The value of shift will either be 0 (Shift is not pressed) or 1 (Shift is pressed). Each key on the keyboard has a numeric value (code), and it's this code that gets passed to the KeyDown procedure.

☐ In the *KeyDown* event procedure, write the following line of code to explore some key code values.

```
MsgBox "keyCode = " & keyCode
```

☐ Bring *ArcMap* to the front of your display. Make sure that your tool is selected. Press some keys on the keyboard to see the message box with the key code.

Question 6: What are the ASCII code values for the following keys? a ______, e ______, TAB ______, ESC ______, 1 ______, Ctrl ______

Many programming references will contain ASCII charts that provide the code for each character. To convert the numeric code to its character equivalent, you can use the *Chr* function.

☐ Modify your message box so that it uses the Chr function, as shown below.

```
MsgBox "keyChar = " & Chr(keyCode)
```

☐ Bring *ArcMap* to the front of your display, select your tool, then press some keys to see the character appear in the message box.

You will use the KeyDown event procedure to allow the user to change the marker color by pressing certain keys. When the user presses *R*, you will make the marker red, *G* will be green, and *B* will be blue.

☐ Return to the *Visual Basic Editor* and navigate again to the *KeyDown* procedure.

☐ Comment out the MsgBox statement.

☐ Begin a Select Case statement using the character represented by the (passed in) keyCode, as shown below.

```
Select Case Chr(keyCode)
```

☐ In the Case that the character is *R*, make the module-level variable m_pColor's RGB property equal to vbRed.

☐ In the Case it is *G*, make m_pColor's RGB property vbGreen.

☐ In the Case it is *B*, make m_pColor's RGB property vbBlue.

☐ In the Case that the character is anything Else, Exit the Sub.

☐ End the Select statement.

☐ Test your code. Bring *ArcMap* to the front of your display.

☐ Again, make sure to unselect, then select your tool to reset variables initialized in the Select procedure.

☐ Draw some points. Press `B` on the keyboard; the next points you draw should be blue.

CHALLENGE: REPORT THE CURRENT MARKER COLOR IN THE STATUS BAR

At the moment, there is no way of telling the current color of your marker until you actually add a point. Write code that will display the current marker color in the ArcMap status bar. Think about which event procedure(s) you will need to code. Should the message be displayed when the tool is initially selected? (Hint: Two module-level variables that you will need [m_strColor and m_pStatusBar] have already been declared for you.)

CHALLENGE: USE DIFFERENT CLICK EVENTS

Examine the tool's Select event procedure. Find the line of code that sets the marker symbol's style (circle). In the MouseDown event procedure, write code that will use a square marker style if the Shift key is pressed, and a circle marker if the Shift key is not pressed. (Hint: If the Shift key is pressed, shift = 1. Otherwise, shift = 0.)

Examine the MouseDown event procedure. In addition to the x and y pixels from the user's click, the mouse button and shift values are also passed to this procedure. Introduce some logic to this procedure that will perform different actions depending on whether it was a right-click (button=2) or a left-click (button=1). Example: if it was a left-click, make the point from the mouse position; if a right-click, use an InputBox to get x- and y-coordinates to construct the point.

> **NOTE:** When using the Input boxes to construct the point, your display may not refresh properly. Try refreshing the display at the bottom of MouseDown.

CHALLENGE: CREATE A RUBBERBANDING TOOL

☐ Bring *ArcMap* to the front of your display.

☐ Open the *Customize* dialog box and add a new UIToolControl to one of your toolbars.

☐ Right-click the new tool and choose to view its source.

☐ In *General Declarations*, declare a variable to reference the application and another to reference a Rubberband object.

☐ When the tool is selected, set the application and rubberband variables appropriately.

☐ Finally, when the mouse is clicked (down), track a new rubberband on the display. (Hint: You can use nothing for a symbol.)

☐ Test your tool.

EXERCISE END

ANSWERS TO EXERCISE 14A QUESTIONS

Question 1: Navigate to the tool's Select event procedure. When will this code execute?

Answer: The procedure will execute each time the user clicks on the tool (only if the tool is not already selected).

Question 2: Look at the parameter list for MouseDown. What are the four parameters passed into this procedure? Where are they passed in from?

Answer: Button, shift, x, and y are listed as parameters for the MouseDown event procedure. These values are passed in from the map document whenever the users clicks on a mouse button inside the map display.

Question 3: Notice the arguments for the AddElement method. The first argument is the element to add to the graphics container. What is the second argument?

Answer: The second argument in the AddElement method represents the element's position inside the graphicscontainer, which ultimately defines the display order.

Question 4: Test your tool. Move ArcMap to the front of the display, select your tool, then click a few times on the display. Did it work as you expected?

Answer: No. The points do not appear on the map display.

Question 5: Refresh the display, either by clicking the Refresh button (lower left of the display) or by resizing the application. Do the points now appear?

Answer: Yes, now the points appear on the map display.

Question 6: What are the ASCII code values for the following keys? a ______, e ______, TAB ______, ESC ______, 1 ______, Ctrl ______

Answer: a _65_, e _69_, TAB _9_, ESC _27_, 1 _49_, Ctrl _17_

SOLUTIONS:

```
'Graphic drawing tool
Private m_pMxApp As IMxApplication
Private m_pMxDoc As IMxDocument
Private m_pDisplay As IScreenDisplay
Private m_pStatusBar As IStatusBar
Private m_strColor As String

Private m_pPoint As IPoint
Private m_pSym As ISimpleMarkerSymbol
Private m_pElem As IElement
Private m_pColor As IRgbColor
'**Challenge - Rubberband tool
Private m_pRubberPoly As IRubberBand

Private Sub UIToolControl1_KeyDown(ByVal keyCode As Long, ByVal shift As Long)
'MsgBox "KeyChar = " & Chr(keyCode)
  Select Case Chr(keyCode)
  Case "R"
      m_pColor.RGB = vbRed
      m_strColor = "Red"   '**challenge
  Case "G"
      m_pColor.RGB = vbGreen
      m_strColor = "Green" '**challenge
  Case "B"
      m_pColor.RGB = vbBlue
      m_strColor = "Blue"  '**challenge
  Case Else
      Exit Sub
  End Select
  m_pStatusBar.Message(0) = "Marker Color: " & m_strColor '**challenge
End Sub

Private Sub UIToolControl1_MouseDown(ByVal button As Long, ByVal shift As Long, _
ByVal x As Long, ByVal y As Long)
  Set m_pPoint = m_pDisplay.DisplayTransformation.ToMapPoint(x, y)

'**challenge
      If button = 1 Then
          Set m_pPoint = m_pDisplay.DisplayTransformation.ToMapPoint(x, y)
      Else
          Set m_pPoint = New Point
          m_pPoint.x = InputBox("X Coord?")
          m_pPoint.y = InputBox("Y Coord?")
      End If
'**

  Set m_pElem = New MarkerElement
  m_pElem.Geometry = m_pPoint
  m_pSym.Color = m_pColor
  Dim pMElem As IMarkerElement
  Set pMElem = m_pElem
  pMElem.Symbol = m_pSym
  m_pMxDoc.ActiveView.GraphicsContainer.AddElement m_pElem, 0
End Sub

Private Sub UIToolControl1_MouseMove(ByVal button As Long, ByVal shift As Long, _
ByVal x As Long, ByVal y As Long)
```

```
  m_pStatusBar.Message(0) = "Marker Color: " & m_strColor '**challenge
End Sub

Private Sub UIToolControl1_MouseUp(ByVal button As Long, ByVal shift As Long, _
ByVal x As Long, ByVal y As Long)
  Dim pEnv As IEnvelope
  Set pEnv = m_pElem.Geometry.Envelope
  pEnv.Expand 200, 200, False
  m_pMxDoc.ActiveView.PartialRefresh esriViewGraphics, Nothing, pEnv
End Sub

Private Sub UIToolControl1_Select()
  Set m_pStatusBar = Application.StatusBar
  Set m_pSym = New SimpleMarkerSymbol
  Set m_pColor = New RgbColor
  m_pColor.RGB = vbRed
  m_pSym.Style = esriSMSCircle
  m_strColor = "Red"

  Set m_pMxApp = Application
  Set m_pMxDoc = ThisDocument
  Set m_pDisplay = m_pMxApp.Display
End Sub

'**Challenge - Rubberband tool
Private Sub UIToolControl2_Select()
  Set m_pMxApp = Application
  Set m_pRubberPoly = New RubberPolygon
End Sub

Private Sub UIToolControl2_MouseDown(ByVal button As Long, ByVal shift As Long, _
ByVal x As Long, ByVal y As Long)
  m_pRubberPoly.TrackNew m_pMxApp.Display, Nothing
End Sub
```

EXERCISE 14B: CREATE A PARCEL PROXIMITY TOOL

In this exercise, you work for the GottGas oil company. When GottGas builds a new gas station, you are in charge of notifying neighboring residents in accordance with local government regulations. One of your biggest challenges is the fact that each municipality has a different radius for determining residents you need to notify. In some cities for example, you have to notify all of the residents within 30 meters of a new gas station, while in others, all residents within 150 meters must be contacted. You will create a tool for ArcMap to easily select parcels within a specified distance of the parcel under development.

EXERCISE SHORTCUT

1. Open *ParcelProximity.mxd* from ..\IPAO\Maps. Create a new UIToolControl. Create a point that represents the map coordinate values of where the user clicks with the UIToolControl.
2. Create a new Spatial Filter that identifies a parcel intersected by the point where the user clicks.
3. Create a Feature Cursor that will hold the record for a parcel if it is intersected based on the SpatialFilter
4. If a parcel was intersected, display the Land Use from the attribute table in a message box. If no parcel was intersected, indicate so in a message box.
5. Display the name of the projection of the parcel based on the Spatial Reference of the Feature Class in a message box.
6. Allow the user to type in a value to buffer the parcel intersected in an input box. Select all neighboring parcels that are within the buffer distance.
7. Remove the original parcel that was clicked on from the selection environment, while leaving all neighboring parcels selected.

STEP 1: CREATE A POINT OBJECT FROM A MOUSE CLICK

The most basic type of geometry is the point. In this step, you will create a point based on a location the user clicks. The point will represent the location of the new gas station. First, you will need to start ArcMap and open the ParcelProximity.mxd map document.

☐ Start *ArcMap*.

☐ Navigate to *C:\student\ipao\maps* and open *ParcelProximity.mxd*.

Next, you will add a new tool to an existing toolbar.

☐ Use the *Customize* dialog box to create a new UIToolControl in the current document (ParcelProximity.mxd).

☐ Drag the new tool control to a toolbar.

☐ Right-click on the new tool control and click *View Source*.

You will add code to the MouseDown event. Your user will click the mouse on the ArcMap display to identify a new gas station. Your utility will eventually use this point to identify the neighboring parcels.

☐ Navigate to the control's *MouseDown* event procedure.

Notice the last two parameters passed to this procedure: the x- and y-coordinates where the user clicked. These coordinates are in display units (pixels), so you will need to convert them to map units before creating a point to represent the new gas station.

☐ Declare the following variables with the Dim keyword.

```
pMxApp As IMxApplication
pPoint As IPoint
```

To convert the pixel coordinates to a point in the current map units, you need to use the ToMapPoint method on the IDisplayTransformation interface.

☐ Set pMxApp equal to Application.

☐ Set pPoint equal to pMxApp.Display.DisplayTransformation.ToMapPoint(x, y).

The pPoint variable now contains a point in map units based on the user's mouse click. This point represents a new GottGas gas station.

STEP 2: CREATE A SPATIAL FILTER

Spatial filters are a type of query filter that retrieve features based on their spatial relationship (e.g., touching, intersecting, within) with a geometric search feature, such as a point, area, or line.

You will use this spatial filter to select the parcel that intersects the point you created in Step 1.

Add the following code below the line that creates the map point (pPoint).

☐ Declare the following variable with a Dim statement:

```
pSpatialFilter As ISpatialFilter
```

SpatialFilters and QueryFilters are CoClasses, which means that you can create them with the New keyword.

☐ Set pSpatialFilter equal to a New SpatialFilter.

Two properties of a SpatialFilter are Geometry and SpatialRel. The Geometry property is used to specify the geometric search features (the new gas station), and the SpatialRel property is used to define the spatial relationship to apply to the search feature (within/intersects/contains, etc.).

You will begin by selecting the parcel that the gas station is in.

☐ Set pSpatialFilter's geometry property equal to pPoint (Hint: This is a property put by reference).

☐ Specify a value for the SpatialRel property that will identify a parcel intersected by the point.

You have now defined the spatial criterion needed to identify the parcel where the new gas station will be built. In the following step, you will apply your SpatialFilter object to the parcels layer in the map.

STEP 3: USE A SPATIAL FILTER TO CREATE A CURSOR

In this step, you will use the spatial filter to create a cursor containing the parcel that contains the new gas station.

Continue coding the tool's *MouseDown* event procedure.

☐ Declare the following variables with the Dim keyword:

```
pMxDoc As IMxDocument
pLayer As ILayer
pFLayer As IFeatureLayer
pFCursor As IFeatureCursor
```

To access the layers of a map, you must have the IMxDocument interface to the Document object.

☐ Set pMxDoc equal to the current document.

☐ Set pLayer equal to the first layer in the activated data frame.

Next, you will make sure the layer is a feature layer. The layer could also be a Graphics layer, Annotation layer, Image layer, or Group layer. You will use the TypeOf statement to test whether the layer supports the IFeatureLayer interface.

☐ Add code (an *If Then* block) to determine if the first layer (pLayer) is a feature layer.

☐ If it is, write the code below to use QueryInterface:

```
Set pFLayer = pLayer
```

☐ If it is not, exit the procedure.

Question 1: Complete the code below to instantiate your feature cursor (pFCursor). This will be a read-only cursor. Should you choose to recycle memory? ________

```
Set pFCursor = pFLayer._______(___________, ______)
```

☐ Write the following message box statement to test if a parcel was identified by the spatial filter:

```
MsgBox "Clicked on a parcel? " & (Not pFCursor.NextFeature Is Nothing)
```

☐ Bring *ArcMap* to the front of the display.

The ParcelProximity.mxd map document contains a bookmark that zooms you into an area for adding a new gas station. To test your utility, you will be adding a gas station close to a major Interstate access area. Notice the Interstate access area near the northwest corner of the Redlands city limits. You will use the bookmark to zoom into this location, then test your tool by clicking on the screen display.

☐ Click *View* > *Bookmarks* > *Interstate access area*

☐ Select your tool.

☐ Click on a parcel.

You should see a message box that indicates a parcel was clicked.

☐ Use your tool to click outside of the parcels.

The message box should indicate that no parcel was clicked.

STEP 4: GET A FEATURE FROM THE CURSOR

You now have a cursor object with the gas station's parcel. Next, you will get a single feature from the cursor. Once you have the feature, you will use it to search for neighboring residents.

☐ Return to your tool's *MouseDown* event in the *Visual Basic Editor*.

☐ Comment out the message box statement from the last step.

You will continue your code below this statement.

☐ At the end of the procedure, declare and set the following variable:

```
Dim pFeature As IFeature
Set pFeature = pFCursor.NextFeature
```

Next, you will test whether the user actually clicked on a feature by using the same syntax you used for the message box.

☐ Complete the following code to test whether you have a valid feature:

```
If pFeature Is ________ Then
  MsgBox "A feature was not selected."
  _____ Sub
End If
```

Next, you will use a message box to report the land use for the chosen parcel. Using the FindField method, you will display the landuse code from an attribute called *Landuse*.

☐ Add the following code:

```
MsgBox "The landuse value is " & pFeature.Value(pFCursor.FindField("Landuse"))
```

☐ Bring ArcMap to the front of the display.

☐ Test your tool by clicking on your tool and then clicking on a parcel.

If a parcel is selected, its land use value will be displayed in a message box.

☐ Click outside of the parcels to test your code.

You should see a message box stating that a feature was not selected.

STEP 5: GET THE SPATIAL REFERENCE FOR A GEODATASET

Before you can buffer the selected feature, you will need to determine the buffer distance (in the parcel layer's native map units). You will have the user specify the buffer distance in meters, then convert to the units of the feature class before buffering. To find these units, you will return the SpatialReference object from the parcel feature class. A SpatialReference object stores coordinate system information for a dataset.

☐ Declare the following variables:

```
pGeoDataset As IGeoDataset
pSpatialRef As ISpatialReference
```

NOTE: A dataset's SpatialReference can be returned from the SpatialReference property defined on the IGeoDataset interface.

☐ Set pGeoDataset equal to pFLayer.FeatureClass.

☐ Set pSpatialRef equal to pGeoDataset.SpatialReference.

You will now determine if the feature class is projected. Once you have determined that the feature class is projected, you can find the units for the feature class from the ILinearUnit interface of the LinearUnit object. All projected coordinate systems are composed of a LinearUnit object.

☐ Declare the following variables:

```
pProjCoordSys As IProjectedCoordinateSystem
pLinearUnit As ILinearUnit
```

☐ Use the same If Then structure as in Step 3 to determine if pSpatialRef has the IProjectedCoordinateSystem interface.

☐ If pSpatialRef does have the IProjectedCoordinateSystem interface, add the following code:

```
Set pProjCoordSys = pSpatialRef  'QueryInterface
Set pLinearUnit = pProjCoordSys.CoordinateUnit
MsgBox "projection = " & pSpatialRef.Name
```

☐ If pSpatialRef does not have the IProjectedCoordinateSystem interface, send a message to the user that the feature layer is not projected and Exit the Sub.

☐ End the If statement.

☐ Test the tool.

STEP 6: BUFFER THE GAS STATION PARCEL AND SELECT THE NEIGHBORING PARCELS

In this step, you will get the buffer distance from the user. You will then take that distance and divide it by the number of meters that are in a unit for this feature layer. This will convert the value provided (meters) into the native map units for the parcels layer.

☐ Declare the following variable:

```
BuffDistance As Double
```

☐ Add the following code:

```
BuffDistance = InputBox("Enter a buffer distance (m)") / pLinearUnit.MetersPerUnit
```

Question 2: The code above prompts the user for a buffer distance (in meters), then converts the measurement (to the native dataset units) by dividing the input by the number of meters in one dataset unit. If the parcels dataset was in meters, what value would the MetersPerUnit property return? ___________________________

____

Next, you will get an interface for the parcels geometry (polygon) that has the Buffer method.

☐ Declare the following variables:

```
pTopo As ITopologicalOperator
pBufferPoly As IPolygon
```

The ITopologicalOperator interface exists on the Point, MultiPoint, Polygon, and Polyline objects. Among other things, the ITopologicalOperator interface gives you access to the Buffer method.

☐ View the ITopologicalOperator interface on the Geometry object model. Notice the Buffer method.

☐ Set pTopo equal to pFeature.Shape.

☐ Set pBufferPoly equal to pTopo.Buffer(BuffDistance).

Next, you will select the neighboring parcels. The IMap interface contains the SelectByShape method. It has three arguments: a shape, an environment, and a Boolean value specifying if you want to select one feature or many features. A true value will select only one feature. A false value will select many features.

☐ Complete the code below to make the selection:

```
pMxDoc.FocusMap.SelectByShape __________, pMxApp.SelectionEnvironment, ______
```

☐ Test the tool.

☐ For now, resize the display (forcing it to refresh) to see the selected features; you will insert code to refresh the display in the next step.

STEP 7: UNSELECT THE GAS STATION PARCEL

In this step, you will create a new selection environment that removes the gas station parcel from the current selection. You only need to notify residents in the surrounding parcels, not in the gas station parcel itself. You will use the geometry for the gas station (point) to remove its parcel from the selection set. First, you will need to change the selection environment so that features are subtracted from the current selection set.

Here is the pseudo code for the code you will write in this step:

```
Get the current selection environment (new selection)
Change it to subtract features from the current selection
Subtract the gas station parcel
Set the selection environment back to the original setting (new)
```

☐ Declare the following variables:

```
pSelectionEnv As ISelectionEnvironment
intSelMethod As Integer
```

☐ Set pSelectionEnv equal to a pMxApp.SelectionEnvironment.

☐ Change the selection method (to subtract features from the selection) by assigning the proper constant to the CombinationMethod property of pSelectionEnv.

Next, you will use the SelectByShape method on the IMap interface to *subtract* the gas station parcel from the selection set.

☐ Write the following code to subtract the gas station polygon from the selection:

```
pMxDoc.FocusMap.SelectByShape pPoint, pSelectionEnv, True
```

☐ Write code to refresh the active view.

☐ From the *Selection* menu, choose I*nteractive Selection Method* > *Create New Selection*.

Choose this option every time before you run your tool because you are programmatically setting it to *Remove from Selected*. You could programmatically reset this back to *Create New Selection* every time your tool executes.

☐ Test your parcel selection tool.

If you have time, proceed to the Challenge step. Otherwise, save your map and exit ArcMap.

CHALLENGE: DISPLAY THE BUFFER POLYGON

In this challenge step, you will display the buffer that was used to make the selection. This will give the user a visual clue as to how the selection was made.

Do you remember how to add graphic elements to the map or layout? If you need to refresh your memory, refer to the previous lesson on layout elements.

☐ Write code to add the buffer polygon as a graphic element to the map.

☐ Save the document and test your tool.

CHALLENGE: REMOVE THE BUFFER POLYGON

In this challenge step, write code to remove each buffer polygon that is added to the graphicscontainer.

☐ Select and test your parcel selection tool.

A new polygon element is added each time the tool is used. However, the previously created polygon element still is shown on the display.

☐ Write code that deletes any previously created buffer polygons from the graphicscontainer.

Congratulations. Now you have a tool to select neighboring parcels around new gas station. The tool allows you to define and display the buffer based on a given distance. You could now use the Report Writer to make a nice report of all the selected parcels, or to address form letters to neighbors selected by your tool.

EXERCISE END

ANSWERS TO EXERCISE 14B QUESTIONS

Question 1: Complete the code below to instantiate your feature cursor (pFCursor). This will be a read-only cursor. Should you choose to recycle memory?

Answer: Yes, memory should be recycled (TRUE) because the Search cursor has read-only access to the retrieved records.

Question 2: The code above prompts the user for a buffer distance (in meters), then converts the measurement (to the native dataset units) by dividing the input by the number of meters in one dataset unit. If the parcels dataset was in meters, what value would the MetersPerUnit property return?

Answer: If the parcel dataset was in meters, then the MetersPerUnit property would return the value of 1.

MetersPerUnit = (Meters/Dataset unit)

MetersPerUnit = (Meters/Meters)

MetersPerUnit = 1

SOLUTIONS:

```
'Parcel Proximity Tool

'*Challenge Step
Private m_pBuffElem as IElement

Private Sub UIToolControl1_MouseDown(ByVal button As Long, ByVal shift As _
Long, ByVal x As Long, ByVal y As Long)
'*Step 1
  Dim pMxApp As IMxApplication
  Dim pPoint As IPoint
  Set pMxApp = Application
  Set pPoint = pMxApp.Display.DisplayTransformation.ToMapPoint(x, y)
'*Step 2
  Dim pSpatialFilter As ISpatialFilter
  Set pSpatialFilter = New SpatialFilter
  Set pSpatialFilter.Geometry = pPoint
  pSpatialFilter.SpatialRel = esriSpatialRelIntersects
'*Step 3
  Dim pMxDoc As IMxDocument
  Dim pLayer As ILayer
  Dim pFLayer As IFeatureLayer
  Dim pFCursor As IFeatureCursor
  Set pMxDoc = ThisDocument
  Set pLayer = pMxDoc.FocusMap.Layer(0)
  If Not TypeOf pLayer Is IFeatureLayer Then Exit Sub
  Set pFLayer = pLayer
  Set pFCursor = pFLayer.Search(pSpatialFilter, True)
'*Step 4
'  MsgBox "Clicked on a parcel? " & (Not pFCursor.NextFeature Is Nothing)
  Dim pFeature As IFeature
  Set pFeature = pFCursor.NextFeature
  If pFeature Is Nothing Then
      MsgBox "A feature was not selected"
      Exit Sub
  End If
  MsgBox "The landuse value is " & _
          pFeature.Value(pFCursor.FindField("Landuse"))
'*Step 5
  Dim pGeoDataset As IGeoDataset
  Dim pSpatialRef As ISpatialReference
  Set pGeoDataset = pFLayer.FeatureClass
  Set pSpatialRef = pGeoDataset.SpatialReference
  Dim pProjCoordSys As IProjectedCoordinateSystem
  Dim pLinearUnit As ILinearUnit
  If TypeOf pSpatialRef Is IProjectedCoordinateSystem Then
      Set pProjCoordSys = pSpatialRef
      Set pLinearUnit = pProjCoordSys.CoordinateUnit
      MsgBox "Projection = " & pProjCoordSys.Name
  Else
      MsgBox "The feature layer is not projected"
      Exit Sub
  End If
```

```
'*Step 6
  Dim BuffDistance As Double
  BuffDistance = InputBox("Enter a buffer distance (m)") / _
                           pLinearUnit.MetersPerUnit
  Dim pTopo As ITopologicalOperator
  Dim pBufferPoly As IPolygon
  Set pTopo = pFeature.Shape
  Set pBufferPoly = pTopo.Buffer(BuffDistance)
  pMxDoc.FocusMap.SelectByShape pBufferPoly, pMxApp.SelectionEnvironment, _
                                False
'*Step 7
  Dim pSelectionEnv As ISelectionEnvironment
  Dim intSelMethod As Integer
  Set pSelectionEnv = pMxApp.SelectionEnvironment
  intSelMethod = pSelectionEnv.CombinationMethod
  pSelectionEnv.CombinationMethod = esriSelectionResultSubtract
  pMxDoc.FocusMap.SelectByShape pPoint, pSelectionEnv, True
  pSelectionEnv.CombinationMethod = intSelMethod
  pMxDoc.ActiveView.Refresh
'*Challenge Step
  Dim pElem As IElement
  Set pElem = New PolygonElement
  pElem.Geometry = pBufferPoly
  Dim pGContainer As IGraphicsContainer
  Set pGContainer = pMxDoc.FocusMap
'*Challenge Step
   If Not m_pBuffElem is Nothing Then
     pGContainer.DeleteElement m_pBuffElem
   End If
   pGContainer.AddElement pElem, 0
   Set m_pBuffElem = pElem
End Sub
```

15

Data management

contents

Exercise 15A: Data conversion

You have already worked with many of the geodatabase objects related to accessing and creating datasets on disk. In a previous exercise, you wrote code to access existing data, to make a new personal geodatabase, and also to produce and edit a new table. In this exercise, you will work with some additional objects for creating datasets (feature classes, personal geodatabases, attribute domains, etc.). You will also work with the FeatureDataConverter object, which allows you to convert feature classes, feature datasets, and tables (Shapefile, dBASE, INFO, Coverage, etc.) to other formats (Shapefile, Geodatabase).

Exercise shortcut

1. Create a sub procedure utilizing IFeatureDataConverter that converts the UnitedStates polygon coverage from ...\Data\Coverages to a geodatabase feature class in ...\Data\States.mdb.

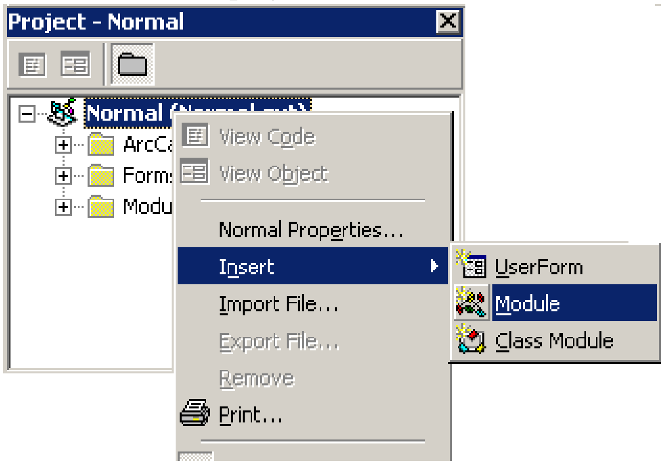
Step 1: Convert an ArcInfo feature class to a geodatabase feature class

In this step, you will write a procedure in ArcCatalog to convert an ArcInfo coverage to a new feature class in a Personal Geodatabase. The code you write here could also be written in ArcMap.

☐ Start *ArcCatalog*. You will write a procedure to convert an ArcInfo feature class into a Personal Geodatabase feature class.

☐ Start the *Visual Basic Editor*.

☐ *Insert* a new standard *module* to the project. Name it **`ManageData`**.



☐ Define a new sub procedure, as shown below.

```
Public Sub ConvertCoverToGDB ()
```

You will begin your procedure by creating new FeatureClassName objects for the input and output datasets. Remember that all Name objects are simply lightweight versions of the objects they represent. The abstract Name class has several creatable subtypes such as TableName, TinName, FeatureClassName, and (one you have already worked with in a previous exercise) WorkspaceName. Name objects can be used as surrogates for (existing or new) dataset or workspace objects. In the code you write below, you will use a FeatureClassName to represent an existing ArcInfo feature class and another to represent a geodatabase feature class that will be created later.

☐ Write the code below to create two new FeatureClassName objects. One will be for the input (coverage), and the other will be for the output (personal geodatabase).

```
Dim pCoverFCName As IDatasetName
Set pCoverFCName = New FeatureClassName

Dim pAccessFCName As IDatasetName
Set pAccessFCName = New FeatureClassName
```

The variables above were dimensioned to the IDatasetName interface because this interface supports properties for specifying the filename and location of a new or existing dataset. The location of the dataset is set with a WorkspaceName object. Therefore, the code you write below will open an existing workspace (for the input and output datasets), then retrieve the WorkspaceName from each.

As with code you wrote in an earlier exercise, your starting point will be the *WorkspaceFactory* CoClass.

☐ Declare the variables below with a Dim statement.

```
pArcInfoWSFactory As IWorkspaceFactory
pArcInfoWS As IDataset 'This gives access to the Workspace's Name object
pArcInfoWSName As IWorkspaceName
```

☐ Initialize these variables as shown below. As a result of the following code, you will have a reference to your input workspace's WorkspaceName object (pArcInfoWSName).

```
Set pArcInfoWSFactory = New ArcInfoWorkspaceFactory
Set pArcInfoWS = pArcInfoWSFactory.OpenFromFile ("C:\Student\IPAO\Data\Coverages", _
                                                 Application.hWnd)
Set pArcInfoWSName = pArcInfoWS.FullName '<--FullName is on the IDataset interface
```

☐ Dimension the variables below to reference the output (Geodatabase) WorkspaceName.

```
pAccessWSFactory As IWorkspaceFactory
pAccessWS As IDataset
pAccessWSName As IWorkspaceName
```

☐ Initialize the variables above using the same basic method you used for the ArcInfo WorkspaceName. In this case you are not creating a new geodatabase, but are putting a feature class in an existing geodatabase. Thus, you need to specify the geodatabase as your workspace you want to put the new feature class in. The pathname for your output workspace is *C:\Student\IPAO\Data\States.mdb.*

Now that you have a reference to the required workspaces, you will specify the filename and location for the input and output datasets.

☐ Write the code below to specify the filename for the input and output datasets.

```
pCoverFCName.Name = "unitedstates:polygon"    '<--polygons from the U.S. coverage
pAccessFCName.Name = "UnitedStates"    '<--the output geodatabase
```

Finally, you need to specify the workspace for the input and output FeatureClassName objects.

☐ Find a property on pCoverFCName and pAccessFCName that can be used to specify their workspace. Set this property with the appropriate object (an object you created above).

Now you have all the required objects to convert the coverage to a geodatabase feature class.

☐ Write the code below to make a new FeatureDataConverter object.

```
Dim pFDConverter As IFeatureDataConverter
Set pFDConverter = New FeatureDataConverter
```

A FeatureDataConverter object provides methods for converting datasets between various formats. Optionally, when converting data, you can return all invalid features (in an enum). Each feature (or row) that is stored in this enum will have an error description that can give you more information on why the feature is invalid.

☐ Call the ConvertFeatureClass method on the FeatureDataConverter. Use the information below for the arguments (notice that Nothing can be used for several of them).

Argument	Value
InputDatasetName	pCoverFCName
InputQueryFilter	Nothing
outputFDatasetName	Nothing
outputFClassName	pAccessFCName
OutputGeometryDef	Nothing
OutputFields	Nothing
configKey	""
FlushInterval	1000
parentHWnd	Application.hWnd

☐ As a final step, display a message box that verifies the conversion.

☐ Run your procedure. If you receive an error, check the pathnames you provided for the workspaces. Bring ArcCatalog to the front of your display. Verify that the new geodatabase was created. Preview it to see if all the features (states) were converted. You may have to refresh the screen in order to see the changes (F5).

NOTE: F5 will not refresh unless the states.mdb is selected; you can also right-click the states.mdb and choose refresh.

CHALLENGE: MAKE YOUR CODE MORE MODULAR

Much of the code you wrote above could be useful in other applications that work with data. Because much of the information used in the code is hard-coded, these procedures would need to be modified each time they were used (e.g., to use different filenames). To make the most of the code you write, it is a good idea to make it as flexible as possible.

☐ Rewrite and reorganize the code in the module so it is flexible enough to work with any dataset without modification.

☐ To do this, you may need to provide input parameters for some of the procedures.

☐ You might also find it useful to define some of these sub procedures as functions instead (functions can return a value or object).

☐ When you have finished updating the procedure, export the ManageData module to your student directory so these procedures can be used in other applications.

CHALLENGE: USE THE FIELD CHECKER TO ENSURE ALL FIELDS ARE BEING CONVERTED PROPERLY

Thus far, you have converted the United States polygon into a feature class in a geodatabase. However, some of the fields did not convert properly due to different data formats and you are unable to use the attribute table. To overcome this, you can use IFieldChecker on the input fields to validate all fields before using them.

☐ In ArcCatalog, delete the *UnitedStates* feature class in the States.mdb geodatabase.

☐ Comment out the last two lines of code in the *ConvertToGDB* procedure.

☐ Use the online help, object model diagrams, and other resources to research how to use IFieldChecker:ValidateWorkspace.

☐ Complete the code and re-execute the *ConvertToGDB* procedure ensuring all fields are valid. The results are provided towards the end of the exercise if you need assistance.

EXERCISE END

SOLUTIONS

```
'**CreateData module **
'Public Sub ConvertToGDB()
    'Make a new Name object for the input coverage...
     Dim pCoverFCName As IDatasetName
    Set pCoverFCName = New FeatureClassName
    'Make a new Name object for the output shapefile...
    Dim pAccessFCName As IDatasetName
    Set pAccessFCName = New FeatureClassName
    'Set up the input (ArcInfo) workspace...
    Dim pArcInfoWSFactory As IWorkspaceFactory
    Dim pArcInfoWS As IDataset
    Dim pArcInfoWSName As IWorkspaceName

    Set pArcInfoWSFactory = New ArcInfoWorkspaceFactory
    Set pArcInfoWS = pArcInfoWSFactory.OpenFromFile("C:\Student\IPAO\Data\Coverages",
Application.hWnd)
    Set pArcInfoWSName = pArcInfoWS.FullName
    'Set up the output (Geodatabase) workspace...
    Dim pAccessWSFactory As IWorkspaceFactory
    Dim pAccessWS As IDataset
    Dim pAccessWSName As IWorkspaceName

    Set pAccessWSFactory = New AccessWorkspaceFactory
    Set pAccessWS = pAccessWSFactory.OpenFromFile("C:\Student\IPAO\Data\States.mdb",
Application.hWnd)
    Set pAccessWSName = pAccessWS.FullName
    'Specify the filenames for the input and output datasets
    pCoverFCName.Name = "unitedstates:polygon"
    pAccessFCName.Name = "UnitedStates"
    'Set the WorkspaceName for input and output datasets
    Set pCoverFCName.WorkspaceName = pArcInfoWSName
    Set pAccessFCName.WorkspaceName = pAccessWSName
    'Use the FeatureDataConverter object to make the conversion...
    Dim pFDConverter As IFeatureDataConverter
    Set pFDConverter = New FeatureDataConverter
    pFDConverter.ConvertFeatureClass pCoverFCName, Nothing, Nothing, pAccessFCName,
Nothing, Nothing, "", 1000, Application.hWnd

    MsgBox "Conversion complete", vbExclamation
End Sub

'*Challenge
Public Function ConvertCoverToGDB(inDSName As IDatasetName, outGDB As String,
outfilename As String) As IEnumInvalidObject
    Dim pAccessfcName As IDatasetName
    Set pAccessfcName = New FeatureClassName
    'Set up the output workspace
    Dim pAccessWSFactory As IWorkspaceFactory
    Dim pAccessWS As IDataset
    Dim pAccessWSName As IWorkspaceName
    Set pAccessWSFactory = New AccessWorkspaceFactory
    Set pAccessWS = pAccessWSFactory.OpenFromFile(outGDB, Application.hWnd)
    Set pAccessWSName = pAccessWS.FullName
    pAccessfcName.Name = outfilename
    Set pAccessfcName.WorkspaceName = pAccessWSName
    Dim pFDConverter As IFeatureDataConverter
    Set pFDConverter = New FeatureDataConverter
    pFDConverter.ConvertFeatureClass inDSName, Nothing, Nothing, pAccessfcName,
Nothing, Nothing, "", 1000, Application.hWnd
    MsgBox "Conversion complete", vbExclamation
End Function


‘>>>CHALLENGE
‘Comment out the last line of code and replace with the code below
‘ pFDConverter.ConvertFeatureClass inDSName, Nothing, Nothing, pAccessfcName,
‘Nothing, Nothing, "", 1000, Application.hWnd

'Get fields for input feature class and run them through the field checker
     Dim pName As IName
     Set pName = pCoverFCName
     Dim pFClass As IFeatureClass
     Set pFClass = pName.Open
     Dim pFieldChecker As IFieldChecker
     Dim pFields As IFields
     Set pFields = pFClass.Fields
     Dim pOutfields As IFields
     Set pFieldChecker = New FieldChecker
     Dim pCoverWorkspaceName As IName
     Set pCoverWorkspaceName = pCoverFCName.WorkspaceName
     Dim pCoverWS As IWorkspace
     Set pCoverWS = pCoverWorkspaceName.Open
     pFieldChecker.InputWorkspace = pCoverWS
     Set pFieldChecker.ValidateWorkspace = pAccessWS
     pFieldChecker.Validate pFields, Nothing, pOutfields

      pFDConverter.ConvertFeatureClass pCoverFCName, Nothing, Nothing, pAccessFCName,
Nothing, pOutfields, "", 1000, Application.hWnd
    MsgBox "Conversion complete", vbExclamation
End Function
```

EXERCISE 15B: DATA MANAGEMENT

In this exercise you will use the ObjectLoader, which will enable you to transfer attributes from one feature class to another.

EXERCISE SHORTCUT

1 Create a QueryFilter referencing the ..\Data\Shapefiles\Counties.shp feature class, holding records for Mobile Homes > 10000 per County. Set the SubFields property on the QueryFilter to *"Shape,NAME,STATE_NAME, Mobilehome,Pop1999"*. Create a variable to reference ..\Data\States.mdb\ManyMobilehomes, which you will load the data into.

2 Import the *AddSubFields.bas* module from ..results/Ex15. Call the *MakeFields* procedure, passing in the Counties feature class, the QueryFilter, and ManyMobilehomes feature class objects from Step 1. Create a new ObjectLoader to Load the attributes of the QueryFilter to the ManyMobilehomes feature class, using fields returned from the MakeFields procedure.

3 Add a field named *PopPerMH* to the ..\Data\States.mdb\ManyMobileHomes feature class.

4 Use an update cursor to populate the *PopPerMH* Field with the population per mobile home based on the *Pop1999* and *Mobilehome* fields.

STEP 1: LOAD DATA USING THE OBJECTLOADER

In this step, you will create a new geodatabase feature class that contains a subset of features from an existing dataset. To accomplish this, you will use a query filter to make the subset (FeatureCursor), then use the object loader to load all features to the new feature class.

☐ Start *ArcCatalog* and open the *Visual Basic editor.*

☐ Define a new procedure in your ManageData module, as shown below.

```
  Public Sub LoadObjects()
```

As with code you have written before that accesses data on disk, your starting point will be the WorkspaceFactory. You will access the shapefile dataset from which you will create the subset.

☐ Dimension the variables below to begin your new procedure.

```
pShapefileFactory As IWorkspaceFactory
pShapefileWorkspace As IFeatureWorkspace
pShapefileFClass As IFeatureClass
```

☐ Set pShapefileFactory equal to a new ShapefileWorkspaceFactory.

☐ Use the OpenFromFile method on pShapefileFactory to open the workspace at *C:\Student\IPAO\Data\Shapefiles*. Store the returned Workspace in the pShapefileWorkspace variable.

Because you dimensioned pShapefileWorkspace as a pointer to the IFeatureWorkspace interface, this variable has access to the OpenFeatureClass method on the Workspace object. You will use this method to open an existing shapefile.

☐ Set *pShapefileFClass* by calling the *OpenFeatureClass* method on *pShapefileWorkspace*. Open the shapefile called *Counties.shp*.

You now need to access the feature class that you are going to load the objects (features) into.

☐ Dimension the variables below to reference the Access feature class that will contain the subset of features.

```
pAccessFactory As IWorkspaceFactory
pAccessWorkspace As IFeatureWorkspace
pAccessFClass As IFeatureClass
```

☐ Instantiate pAccessFactory by creating a new AccessWorkspaceFactory.

Instead of using the *OpenFromFile* method on pAccessFactory to access an existing workspace, you will use the *Open* method. Both methods accomplish the same thing, only differently.

Open is generally used to access workspaces that you must connect to such as an ArcSDE geodatabase. A required parameter for the Open method is a PropertySet object, which contains all the information required to get the workspace. For your purposes, the PropertySet will simply contain the path to the workspace. If needed, it could also contain information such as user name and password.

☐ Create a new PropertySet object and specify the pathname to the Access database using the SetProperty method, as shown below.

```
Dim pProps As IPropertySet
Set pProps = New PropertySet
```

```
pProps.SetProperty "DATABASE", "C:\Student\IPAO\Data\States.mdb"
```

☐ Call the Open method on pAccessFactory using the PropertySet created above and the application's hWnd. Store the returned Workspace in the pAccessWorkspace variable.

☐ Use the code below to get the desired feature class from the Access database.

```
Set pAccessFClass = pAccessWorkspace.OpenFeatureClass("ManyMobileHomes")
```

☐ The ManyMobileHomes feature class has already been created for you in order to save time. Typically, when performing a task like this, you would need to create a new feature class to load your data into.

☐ Make a new query filter. Set its WhereClause property to select all counties that have more than 10,000 mobile homes with the code below. Set the SubFields equal to the fields that you want to carry over into the new feature class.

```
Dim pQueryFilter As IQueryFilter
Set pQueryFilter = New QueryFilter
pQueryFilter.SubFields = "Shape,NAME,STATE_NAME,Mobilehome,Pop1999"
pQueryFilter.WhereClause = "Mobilehome > 10000"
```

NOTE: Make sure there are no spaces between the commas and field names when using setting the SubFields property.

STEP 2: ADD AN EXISTING MODULE TO ADD FIELDS

☐ Right-click on *modules* and choose *Import file*. Navigate to *C:/student/ipao/results/Ex15* and add the *AddSubFields.bas* module.

This function creates the output fields that will be used in the empty feature class. It is taking the SubFields property you just defined and delimiting the field names into actual fields. The function asks for the input feature class, the query filter to obtain the sub fields, and the output feature class to put the fields into.

☐ Dimension a variable as IFields and set it equal to the return value of your function in the Loadobjects procedure.

```
Dim pOutputFields As IFields
Set pOutputFields = MakeFields(pShapefileFClass, pQueryFilter, pAccessFClass)
```

☐ Dimension a variable as IObjectLoader and set it equal to a new Object Loader.

```
Dim pObjectLoader As IObjectLoader
Set pObjectLoader = New ObjectLoader
```

☐ Dimension a variable as IEnumInvalidObject.

```
Dim pEnumInvalidObject as IEnumInvalidObject
```

The LoadObjects method from the ObjectLoader class returns IEnumInvalidObject. This enumeration will point to all features that may not load properly.

☐ Run the Load objects method. Pass in the input feature class, the query filter, the output feature class you are writing to, the output fields, and a flush interval. You may want to look up in the help the additional parameters that are being passed as nothing.

```
pObjectLoader.LoadObjects Nothing, pShapefileFClass, pQueryFilter, pAcccesFClass, _
          pOutputFields, False, 0, False, False, 10, pEnumInvalidObject
```

☐ Dimension a variable as IInvalidObjectInfo to check to see if there were any features that did not load. If so, display a message box displaying that some features did not load, otherwise proceed.

```
Dim pInvalidObject As IInvalidObjectInfo
Set pInvalidObject = pEnumInvalidObject.Next
If Not pInvalidObject Is Nothing Then
  MsgBox "Some or all features did not load"
Else
  MsgBox "Conversion Complete"
End If
```

☐ Export your module to the student directory.

☐ Test the procedure.

Question 1: Navigate to the dataset in ArcCatalog and refresh. How many features did it insert into the ManyMobileHomes feature class? ______________________

STEP 3: ADD A NEW FIELD TO AN EXISTING TABLE

In this step, you will add a new field to the existing ManyMobileHomes feature class. You have created fields before, but in the context of creating a Fields collection to be used to create a new Table (or FeatureClass). To add a field to an existing dataset, you do not need to access the Fields collection. Instead, you can call the AddField method directly on the Table or FeatureClass itself.

☐ Return to the *Visual Basic Editor*.

☐ Insert the sub procedure shown below into the ManageData module.

```
Public Sub AddField( )
```

☐ Copy and paste the code you wrote previously (in the LoadObjects procedure) to reference the States.mdb database and the ManyMobileHomes feature class. The code is listed here (but do not type it; that is too much work.).

```
Dim pAccessFactory As IWorkspaceFactory
Dim pAccessWorkspace As IFeatureWorkspace
Dim pAccessFClass As IFeatureClass

Set pAccessFactory = New AccessWorkspaceFactory
Set pAccessWorkspace = _
pAccessFactory.OpenFromFile("C:\student\ipao\data\states.mdb", Application.Hwnd)
Set pAccessFClass = pAccessWorkspace.OpenFeatureClass("ManyMobileHomes")
```

☐ Dimension a variable called **`pField`** as a pointer to the IFieldEdit interface.

☐ Instantiate pField by creating a new Field object.

☐ Assign the following values to some of pField's properties.

Property	**Value**
Name	"PopPerMH"
Type	esriFieldTypeInteger
Length	16
AliasName	"People per Mobile Home"

☐ Call the AddField method on pAccessFClass to add pField to the feature class.

☐ Run your procedure.

☐ After the procedure has executed, navigate to the *ManyMobileHomes* feature class in ArcCatalog. Double-click the feature class in the catalog to launch its *Properties* dialog.

☐ Click the *Fields* tab in the Properties dialog.

Question 2: Was the new field successfully added to the feature class? _________
Are there any values currently stored in this field? ________________________

☐ In the properties dialog, highlight the *PopPerMH* field and press **`Delete`** on your keyboard to remove the field.

The PopPerMH field will be recreated when you test your code in the following step, in which you will write code to calculate values for the field.

STEP 4: CALCULATE POPULATION PER MOBILE HOME

In this step, you will write code to populate the new PopPerMH field with the number of people per mobile home for each record (county) in the dataset. To accomplish this you will use an update cursor, which is used to modify existing features in a dataset.

☐ Return to the *Visual Basic Editor*.

You will add the following code to the bottom of the AddField procedure.

☐ Dimension a variable called **`pUpdateFeatures`** as a pointer to the IFeatureCursor interface.

☐ Initialize pUpdateFeatures by calling the Update method on pAccessFClass. Use Nothing for the required QueryFilter argument and specify that memory should not be recycled.

> NOTE: The code you wrote above produces an update cursor. By specifying *Nothing* for the query filter, you have chosen to return all features in the feature class and place them in the cursor for editing. Because it is an editing cursor, memory should not be recycled.

☐ Declare the variables below (using Dim) to store the index position of fields required to calculate the new field.

```
intNewFld As Integer
intPopFld As Integer
intHomeFld As Integer
```

☐ Use the code below to assign the index of the new field **`"PopPerMH"`** to intNewFld.

```
intNewFld = pUpdateFeatures.FindField("PopPerMH")
```

> NOTE: The FindField method is available on IFeatureClass, ITable, IFields, ICursor, and IFeatureCursor.

☐ Assign the indexes of the other two fields to their corresponding variables using the same syntax above. The population field is named "Pop1999", and the mobile home field is "Mobilehome".

☐ Declare the variables below. One will store records (features) pulled out of the cursor. The other two will store values for population and number of mobile homes.

```
Dim pFeature As IFeature
Dim lngPop As Long
Dim lngHomes As Long
```

To update the values of features in a cursor, you will pull a feature from the cursor, calculate a new value for the "PopPerMH" field (based on the existing values for population and mobile homes), then call the UpdateFeature method on the cursor to make the change in the database.

You will create a loop in order to update every feature in the cursor.

☐ Set pFeature equal to the next feature in the cursor.

☐ Begin a loop that will execute until pFeature is nothing.

☐ Get the value for the feature's population field by writing the code below:

```
lngPop = pFeature.Value(intPopFld)
```

☐ Also get the value for the number of mobile homes using the syntax above (store it in the appropriate variable).

☐ Complete the code below to calculate a value for the PopPerMH field. The value should be the number of people per mobile home.

```
pFeature.________________(_____________) = ____________ / ___________________
```

☐ Call the UpdateFeature method on pUpdateFeatures. Pass in the feature to update (pFeature).

☐ Store the next feature from the cursor (pUpdateFeatures) in the pFeature variable.

! If you do not add the line of code above inside the loop, you will have an infinite loop.

☐ End the loop.

☐ Finally, add a message box statement to notify the user that the update is complete.

☐ Run the procedure.

☐ Once again, use ArcCatalog to navigate to the *ManyMobileHomes* feature class. Preview the attribute table.

NOTE: You may need to refresh the States personal geodatabase in order to see the changes.

Did the procedure successfully calculate these values? Export your ManageData module to the student directory.

EXERCISE END

ANSWERS TO EXERCISE 15B QUESTIONS

Question 1: Navigate to the dataset in ArcCatalog and refresh. How many features did it insert into the ManyMobileHomes feature class?

Answer: 95 features should have been added to ManyMobileHomes feature class.

Question 2: Was the new field successfully added to the feature class? Are there any values currently stored in this field?

Answer: Yes, a new field should have been added. There should not be any values currently stored in the field.

SOLUTIONS

```
Public Sub LoadObjects()
  'Set up the input feature class, i.e. the data source
   Dim pShapefileFClass As IFeatureClass
   Dim pShapefileFactory As IWorkspaceFactory
   Dim pShapefileWorkspace As IFeatureWorkspace
   Set pShapefileFactory = New ShapefileWorkspaceFactory
   Set pShapefileWorkspace =
   pShapefileFactory.OpenFromFile("C:\Student\Ipao\Data\Shapefiles", 0)
   Set pShapefileFClass = pShapefileWorkspace.OpenFeatureClass("counties.shp")
   'Set up the feature class you are loading data into
   Dim pAccessFClass As IFeatureClass
   Dim pAccessFactory As IWorkspaceFactory
   Dim pAccessWorkspace As IFeatureWorkspace
   Set pAccessFactory = New AccessWorkspaceFactory
   Dim pProps As IPropertySet
   Set pProps = New PropertySet
   pProps.SetProperty "DATABASE", "C:\Student\IPAO\Data\States.mdb"
   Set pAccessWorkspace = pAccessFactory.Open(pProps, Application.hWnd)
   Set pAccessFClass = pAccessWorkspace.OpenFeatureClass("ManyMobileHomes")

   ' Specify a subset of the input data
   Dim pQueryFilter As IQueryFilter
   Set pQueryFilter = New QueryFilter
   pQueryFilter.SubFields = "Shape,NAME,STATE_NAME,Mobilehome,Pop1999"
   pQueryFilter.WhereClause = "Mobilehome > 10000"

   Dim pOutputFields As IFields
   Set pOutputFields = MakeFields(pShapefileFClass, pQueryFilter, pAccessFClass)

   Dim pObjectLoader As IObjectLoader
   Set pObjectLoader = New ObjectLoader
   Dim pEnumInvalidObject As IEnumInvalidObject
   pObjectLoader.LoadObjects Nothing, pShapefileFClass, pQueryFilter, _
                        pAccessFClass, pOutputFields, False, 0, False, False, 10, _
                        pEnumInvalidObject

  Dim pInvalidObject As IInvalidObjectInfo
  Set pInvalidObject = pEnumInvalidObject.Next
  If Not pInvalidObject Is Nothing Then
    MsgBox "Some or all features did not load"
  Else
    MsgBox "Conversion Complete"
  End If

  'refresh ArcCatalog (not required in exercise)
  Dim pGxApp As IGxApplication
  Set pGxApp = Application
  pGxApp.Refresh "C:\Student\Ipao\Data\States.mdb"

End Sub

Public Sub AddField()
  Dim pAccessFactory As IWorkspaceFactory
  Dim pAccessWorkspace As IFeatureWorkspace
  Dim pAccessFClass As IFeatureClass

  Set pAccessFactory = New AccessWorkspaceFactory
  Set pAccessWorkspace = _
  pAccessFactory.OpenFromFile("C:\Student\Ipao\Data\States.mdb",Application.Hwnd )
  Set pAccessFClass = pAccessWorkspace.OpenFeatureClass("ManyMobileHomes")
'
 Dim pField As IFieldEdit
  Set pField = New Field
  pField.Name = "PopPerMH"
  pField.Type = esriFieldTypeInteger
  pField.Length = 16
  pField.AliasName = "People Per Mobile Home"
' pAccessFClass.AddField pField

  Dim pUpdateFeatures As IFeatureCursor
  Set pUpdateFeatures = pAccessFClass.Update(Nothing, False)

  Dim intNewFld As Integer
  Dim intPopFld As Integer
  Dim intHomeFld As Integer

  intNewFld = pUpdateFeatures.FindField("PopPerMH")
  intPopFld = pUpdateFeatures.FindField("Pop1999")
  intHomeFld = pUpdateFeatures.FindField("Mobilehome")

  Dim pFeature As IFeature
  Dim lngPop As Long
  Dim lngHomes As Long

  Set pFeature = pUpdateFeatures.NextFeature
  Do Until pFeature Is Nothing
      lngPop = pFeature.Value(intPopFld)
      lngHomes = pFeature.Value(intHomeFld)
      pFeature.Value(intNewFld) = lngPop / lngHomes
      pUpdateFeatures.UpdateFeature pFeature

      Set pFeature = pUpdateFeatures.NextFeature
  Loop
  MsgBox "Update complete"
End
```

16

Application framework and events

contents

EXERCISE 16A: PROGRAM THE USER INTERFACE

This exercise will teach you how to use the existing customization framework in your own scripts. You will create a new shortcut menu and add new commands to it.

EXERCISE SHORTCUT

1. Open *ex16a.mxd* from ..\IPAO\maps. In the MxDocument_OnContextMenu event, create an ICommandBar object that references the View Menu from the ArcID code module. Return *True* for the MxDocument_OnContextMenu to display the View Menu.

2. Create a new empty menu named *MyMenu* as a shortcut menu. Display this menu instead of the View Menu as in Step 1.

3. If the Active View is a Data Frame, add the Add Data, Fixed Zoom In, and Fixed Zoom Out tools to the menu. If the Active View is the Page Layout, add the Fixed Zoom Out and Fixed Zoom Out tools that work with the Page Layout.

4. Add a sub menu named *Layer On/Off*. Create a loop that iterates through all layers in the Table of Contents, adding each one as a choice on the submenu.

5. Create an ICommandItem object that represents the item chosen on the menu. If the item is on the submenu, call the *TurnOnOffLayer* procedure provided in the map document.

STEP 1: USE EXISTING SHORTCUT MENUS

In this step, you will write code to pop up the View menu when the user right-clicks on the ArcMap screen display. First, view the existing menus.

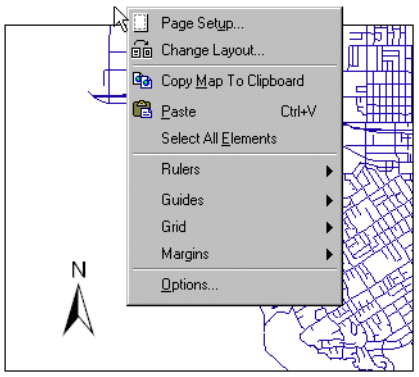
☐ Start *ArcMap*.

☐ Navigate to *C:\Student\ipao\maps* and open *ex16a.mxd*.

☐ Make sure the map is in *Layout* view.

☐ Right-click anywhere on the screen display.

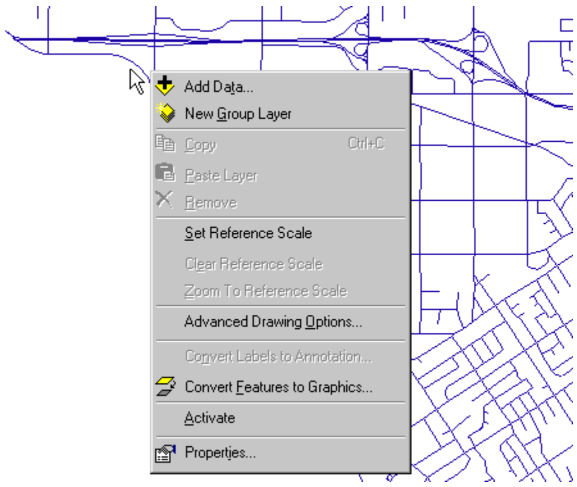
The default context menu for the screen display in Layout view is the Layout context menu, as shown below.



☐ Switch to *Data* view.

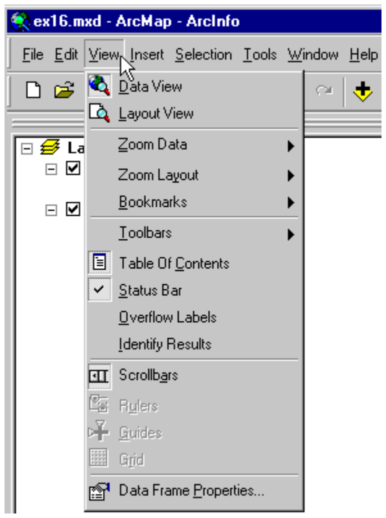
☐ Right-click somewhere on the screen display.

The default context menu for the screen display in Data view is the Map View context menu, as shown below.



You will write code to pop up the View menu instead of the Layout or Map View context menus when you right-click on the screen display.

☐ Click the *View* menu on the standard menu bar and examine the choices. Do not click on anything.



This is the menu that you will pop up as the Data view context menu.

☐ Launch the *Visual Basic Editor*.

☐ In the *Project* window, double-click on *ThisDocument* under *Project > ArcMap Objects* to open the project's ThisDocument code module.

☐ In the object pulldown list, choose *MxDocument*.

You will add code to the OnContextMenu event. The OnContextMenu event fires when the user right-clicks on the ArcMap screen display.

☐ For *procedure*, click *OnContextMenu*.

The Find method on the CommandBars collection can be used to return an individual command or an entire command bar (toolbar or menu). Because you are finding a menu, you will use the ICommandBar interface for the returned object.

☐ Dimension the following variables to the interfaces provided:

```
pCommandBar As ICommandBar
Set pCommandBar = ThisDocument.CommandBars.Find(ArcID.View_Menu)
pCommandBar.Popup
```

Whenever you write code for a context menu, you need to let the customization framework (i.e., Application) know that you have handled the event. Otherwise, the default context menu will be displayed in addition to the custom one.

You will first test your code without notifying the application that you have handled the event with your own menu.

Question 1: Close the Visual Basic Editor. In ArcMap, right-click on the display (while in Data view). Which context menu is displayed? ____________________

Question 2: Right-click again. Now which menu is displayed?________________

To tell the application that you have handled the OnContextMenu event, you need to set MxDocument_OnContextMenu equal to True. Otherwise, your menu and the default menu will alternately appear.

The Customization framework will look for custom context menus in the following order: OnContextMenu for the document, OnContextMenu for a base template (if one exists), then OnContextMenu for the normal template. As soon as the OnContextMenu event is handled (i.e., it is set to True), the application will stop searching.

☐ Open the VBA editor.

☐ Return *True* for the MxDocument_OnContextMenu event.

☐ Bring *ArcMap* to the front of your display.

☐ Right-click several times on the screen display.

Only the View menu should appear each time you click.

STEP 2: CREATE A NEW SHORTCUT MENU

In this step, you will create your own (new) context menu. You will use the CommandBars collection to create the new menu, then pop it up in the OnContextMenu event. In later steps, you will add custom menu choices to the context menu.

☐ In the *Visual Basic Editor*, open the *ThisDocument* module under *ArcMap Objects > Project*.

☐ Navigate back to the *OnContextMenu* event for *MxDocument*.

☐ Comment out the first three lines of code in the OnContextMenu procedure.

The CommandBars collection allows you to create new command bars. When creating a command bar, there are two things that you must specify: Name and Type. The Type argument determines what kind of command bar you are creating: toolbar, menu, or shortcut menu. In this example, you are creating a shortcut menu that will cease to exist when the document is closed.

All the code you add in the rest of this step should go before the line that sets Document_OnContextMenu equal to True.

☐ Declare the following variable with the Dim keyword:

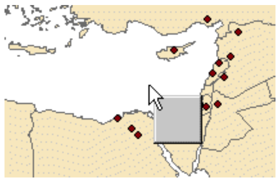
```
pMenu As ICommandBar
```

☐ Add the following code:

```
Set pMenu = ThisDocument.CommandBars.Create("MyMenu", esriCmdBarTypeShortcutMenu)
pMenu.Popup
```

☐ Bring ArcMap to the front of the display.

☐ Right-click on the display to test your new shortcut menu.



You should see a context menu that has no choices. In the rest of this exercise, you will add choices to this context menu.

STEP 3: MAKE SHORTCUT MENUS DYNAMIC

In this step, you will add logic that populates the context menu with different commands depending on whether the application is in Layout or Data view.

All the code you add in the rest of this step should go before the line that displays the menu (pMenu.Popup).

☐ Declare the following variables with the Dim keyword.

```
pMxDoc As IMxDocument
pActiveView As IActiveView
```

☐ Set pMxDoc equal to the current document.

☐ Set pActiveView equal to pMxDoc.ActiveView.

☐ Add the following code:

```
If TypeOf pActiveView Is IMap Then
    pMenu.Add ArcID.File_AddData
    pMenu.Add ArcID.PanZoom_ZoomInFixed
    pMenu.Add ArcID.PanZoom_ZoomOutFixed
Else
    pMenu.Add ArcID.PageLayout_PageZoomInFixed
    pMenu.Add ArcID.PageLayout_PageZoomOutFixed
End If
```

☐ Test your context menu in both *Layout* and *Data* view.

You should see a dynamic context menu that contains different choices when it is accessed in Layout view than when it is accessed in Data view.

Step 4: Add a submenu to a shortcut menu that references a Macro

Now you will enhance your shortcut menu by adding a pullright menu for turning layers on and off. The new menu will eventually contain the names of each layer in the focused map.

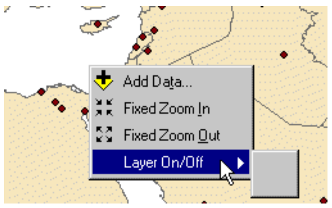
All the code you add in the rest of this step should go before you show the menu (pMenu.Popup).

☐ Use Dim to declare the following variable:

```
pSubMenu As ICommandBar
```

☐ Set pSubMenu equal to pMenu.CreateMenu("Layer On/Off").

☐ Test your shortcut menu.



Your context menu should have a pullright menu with no choices in it. Using this technique, you could create an unlimited hierarchy of submenus.

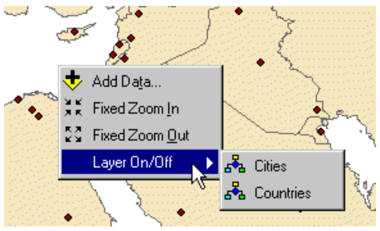
Next, you will create a command that runs another procedure when it is executed. The command will execute the TurnLayerOnOff procedure that has been created for you.

All the code you add in the rest of this step should go before pMenu.Popup and after pSubMenu is created.

☐ Complete the code below to loop through all layers in the focused map, then dynamically add each layer name as a choice to the submenu.

```
Dim pEnumLayers As _____________
Dim pLayer As ILayer
Set pEnumLayers = pMxDoc.FocusMap._________
Set pLayer = pEnumLayers._____
Do Until _______ Is Nothing
    pSubMenu.CreateMacroItem pLayer.Name
    Set pLayer = pEnumLayers._______
Loop
```

☐ Test your shortcut menu.



You should see all layers in the active data frame listed in the context menu's submenu.

Question 3: What happens if you choose one of the layer names in the submenu?
____

Is there any code associated with these menu choices? ________________________

STEP 5: USE PLACEHOLDER COMMANDS

In this step, you will write code to capture the menu item that was chosen by the user. If the menu choice was one of the layer names you added to the submenu, you will call the TurnOnOffLayer procedure, passing in the layer name.

☐ Delete this line of code: pMenu.Popup.

All the code you add in the rest of this step should go before the line that sets MxDocument_OnContextMenu equal to True.

☐ Use Dim to declare the following variable:

```
pItemChosen As ICommandItem
```

☐ Set pItemChosen equal to pMenu.Popup.

When the user makes a selection in the context menu, the Popup method will return the CommandItem that was selected by the user (stored in the pItemChosen variable). Now you will test the chosen CommandItem to see if it is one of the items on the submenu (i.e., a layer name). If it is, you will pass the layer name to the TurnOnOffLayer procedure to toggle its visibility.

☐ Write the code below to see if the chosen menu item is one of the layer names on the submenu.

```
If pItemChosen.Parent Is pSubMenu Then
```

By using the *Parent* property of a CommandItem, you can see which menu or toolbar it is on. The code you wrote above will check if the chosen item is on the context menu's submenu.

☐ If the chosen item is on the submenu (it is a layer name), execute the code below.

```
TurnOnOffLayer pItemChosen.Name
```

Remember that the item was named with the layer name. By getting the Name property from the command item, you are getting the layer name selected by the user.

☐ Read through the TurnOnOffLayer procedure to see how it works.

Question 4: Test your shortcut menu. Are you able to turn layers on and off using the menu choices? ____

Your shortcut (context) menu should allow you to run the ArcMap commands you added, as well as the TurnLayerOnOff procedure.

If you have time, proceed to the Challenge step. Otherwise, save your map and exit ArcMap.

CHALLENGE: ADD A PULLDOWN MENU TO A TOOLBAR

Use the following sample code to create a pulldown menu on a toolbar.

```
Sub MakeMenu()
    Dim pCmdBar As ICommandBar
    Dim newMenu As ICommandBar
    Dim newSubMenu As ICommandBar

    'Find the toolbar you want to add a new menu to
    'This example adds new menu to the Standard toolbar
    Set pCmdBar = ThisDocument.CommandBars.Find(ArcID.Standard_Toolbar)

    'Create the new menu and add stuff to it
    Set newMenu = pCmdBar.CreateMenu("MyMenu")
    newMenu.Add ArcID.PanZoom_Down
    newMenu.Add ArcID.PanZoom_FullExtent

    'Create the new pullright menu on your
    'new menu and add stuff to it
    Set newSubMenu = newMenu.CreateMenu("MySubMenu")
    newSubMenu.Add ArcID.PanZoom_Down
    newSubMenu.Add ArcID.PanZoom_FullExtent

End Sub
```

Question 5: Test the code. Do you see the new menu added to the standard toolbar?

____

EXERCISE END

Answers to Exercise 16A questions

Question 1: Close the Visual Basic Editor. In ArcMap, right-click on the display (while in Data view). Which context menu is displayed?

Answer: The View Menu context menu is displayed.

Question 2: Right-click again. Now which menu is displayed?

Answer: The Data View context menu is displayed.

Question 3: What happens if you choose one of the layer names in the submenu? Is there any code associated with these menu choices?

Answer: Nothing happens, there is not code associated with these menu choices yet.

Question 4: Test your shortcut menu. Are you able to turn layers on and off using the menu choices?

Answer: Yes, if you are unable to turn layers on and off, error check your code.

Question 5: Test the code. Do you see the new menu added to the standard toolbar?

Answer: Yes, if you do not see the new menu, error check your code.

SOLUTIONS

```
'** ThisDocument Module (Ex16a.mxd)**

Private Function MxDocument_OnContextMenu(ByVal X As Long, ByVal Y As Long) As
Boolean
'    Dim pCommandBar As ICommandBar
'    Set pCommandBar = ThisDocument.CommandBars.Find(ArcID.View_Menu)
'    pCommandBar.Popup

    Dim pMenu As ICommandBar
    Set pMenu = ThisDocument.CommandBars.Create _
    ("MyMenu", esriCmdBarTypeShortcutMenu)

    Dim pMxDoc As IMxDocument
    Dim pActiveView As IActiveView
    Set pMxDoc = ThisDocument
    Set pActiveView = pMxDoc.ActiveView
    If TypeOf pActiveView Is IMap Then
        pMenu.Add ArcID.File_AddData
        pMenu.Add ArcID.PanZoom_ZoomInFixed
        pMenu.Add ArcID.PanZoom_ZoomOutFixed
    Else
        pMenu.Add ArcID.PageLayout_PageZoomInFixed
        pMenu.Add ArcID.PageLayout_PageZoomOutFixed
    End If

    Dim pSubMenu As ICommandBar
    Set pSubMenu = pMenu.CreateMenu("Layer On/Off")

    Dim pEnumLayers As IEnumLayer
    Dim pLayer As ILayer
    Set pEnumLayers = pMxDoc.FocusMap.Layers
    Set pLayer = pEnumLayers.Next
    Do Until pLayer Is Nothing
        pSubMenu.CreateMacroItem pLayer.Name
        Set pLayer = pEnumLayers.Next
    Loop
    Dim pItemChosen As ICommandItem
    Set pItemChosen = pMenu.Popup
    If pItemChosen.Parent Is pSubMenu Then
        TurnOnOffLayer pItemChosen
    End If
    MxDocument_OnContextMenu = True
End Function


Public Sub TurnOnOffLayer(aLayerName As String)
    Dim pMxDoc As IMxDocument
    Dim pLayers As IEnumLayer
    Dim pLayer As ILayer

    Set pMxDoc = ThisDocument
    Set pLayers = pMxDoc.FocusMap.Layers
    Set pLayer = pLayers.Next
    Do
        If pLayer.Name = aLayerName Then Exit Do
        Set pLayer = pLayers.Next
    Loop
```

```
        pLayer.Visible = Not pLayer.Visible
        pMxDoc.ActiveView.Refresh
        pMxDoc.UpdateContents
    End Sub
```

EXERCISE 16B: CODING ARCOBJECTS EVENTS

In addition to having characteristics a programmer can get and set (properties) and actions that they can perform (methods), objects can also respond to events. In other words, objects may have certain external actions that cause them to run some associated code. As an ArcObjects programmer, you have already written code associated with object events. Most of the user forms you have programmed, for example, respond to an *Initialize* event. The controls on the form, such as command buttons, have had code associated with several of their events. In these cases, however, the events were easily accessible. All you needed to do was choose the appropriate object and event in the form's code module. In this exercise, you will write code to access events that are associated with some of the ArcObjects.

EXERCISE SHORTCUT

1. Open *ex16b.mxd* from ..\IPAO\Maps. Declare a variable with the WithEvents keyword to capture events from the FeatureLayer class. Write code in the VisibilityChanged event that will display message box indicating whether a layer is turned on or off.
2. In the MxDocument_OpenDocument event, initialize the variable that references the events for the FeatureLayer class to the United States layer. Save the map and reopen it. Test to ensure the message box appears every time you turn the layer on and off.
3. Before the message box is displayed indicating the layer is turned on or off, reset the extent of the map display. If the United States layer is on, set the extent to that layer. If the United States Layer if off, set the extent to all layers in the data frame.

STEP 1: CAPTURE FEATURELAYER EVENTS

In this step, you will write code to *tap into* the events of a FeatureLayer in your map. You will learn how to access object events and how to associate code with a particular event.

The event code you eventually write will dynamically change the extent of the map display whenever the States layer is turned on or off.

☐ Start *ArcMap*.

☐ Open *ex16b.mxd* from the *C:\Student\IPAO\Maps* directory.

☐ Start the *Visual Basic Editor* and open the project's *ThisDocument* module.

There are some ArcObjects events that you already have access to, as you probably know. In the ThisDocument module, you can provide code for several document events such as *Open*, *Close*, and *New*. To code these events, all you need to do is select the desired event using the pulldown lists at the top of the module.

☐ Pull down the object list at the upper-left of the *ThisDocument* module. Choose *MxDocument*.

Question 1: Pull down the procedure list at the upper-right of the module. Notice that all the events related to the MxDocument class are listed here. How many events does the document have?____

By declaring an object variable in the right way, you can tap into all the events that it supports. Below, you will declare a variable to access FeatureLayer events.

☐ Move to the *General Declarations* section of the *ThisDocument* module.

☐ Declare the following module-level variables:

```
Private m_pMxDoc As IMxDocument
Private m_pUSExtent As IEnvelope
```

Next, you will use the ESRI Object Browser to determine which event you should use. The ESRI Object Browser is also helpful in determining whether an outbound interface is default or not. The syntax for how you declare your variables will differ depending on if the outbound interface is default or not.

☐ Click *Start* > *Programs* > *ArcGIS* > *Developer Tools* > *Object Browser*.

☐ Click *File* > *Object Library References*.

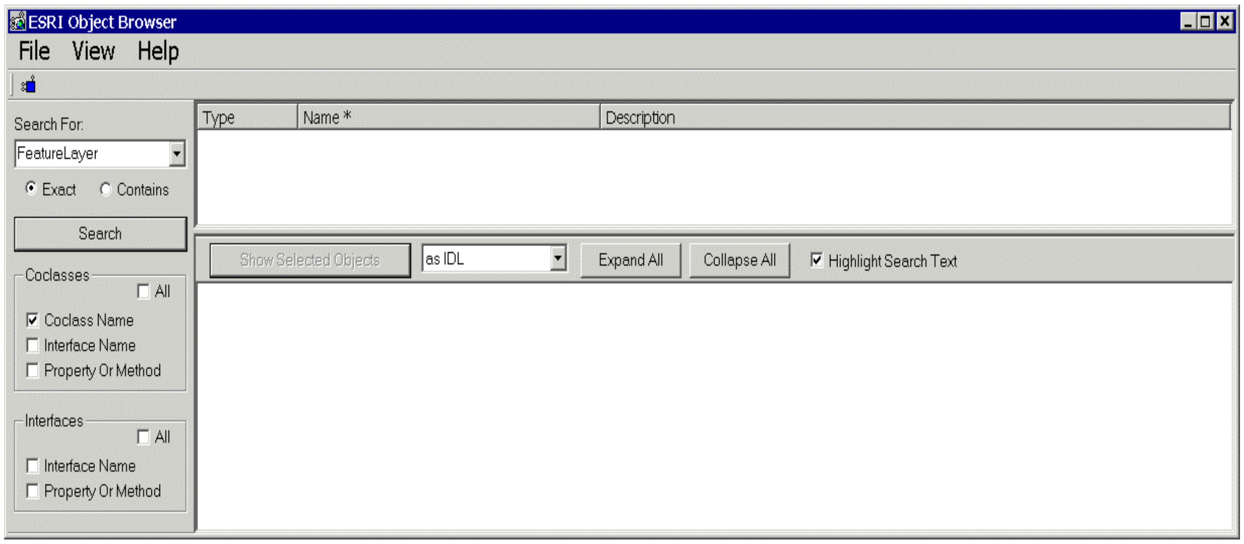
☐ In the *Object Library References* dialog, click *Add*.

☐ Under *Select From Registry*, click *ESRI Carto Object Library*.

☐ Click *OK* in both the *Select From Registry* and *Object Library References* dialogs.

☐ In the *ESRI Object Browser*, under *Search For*, type **`FeatureLayer`** and check *Exact*.

☐ Modify the *ESRI Object Browser* to only search for *CoClasses* > *CoClass Name* as below:



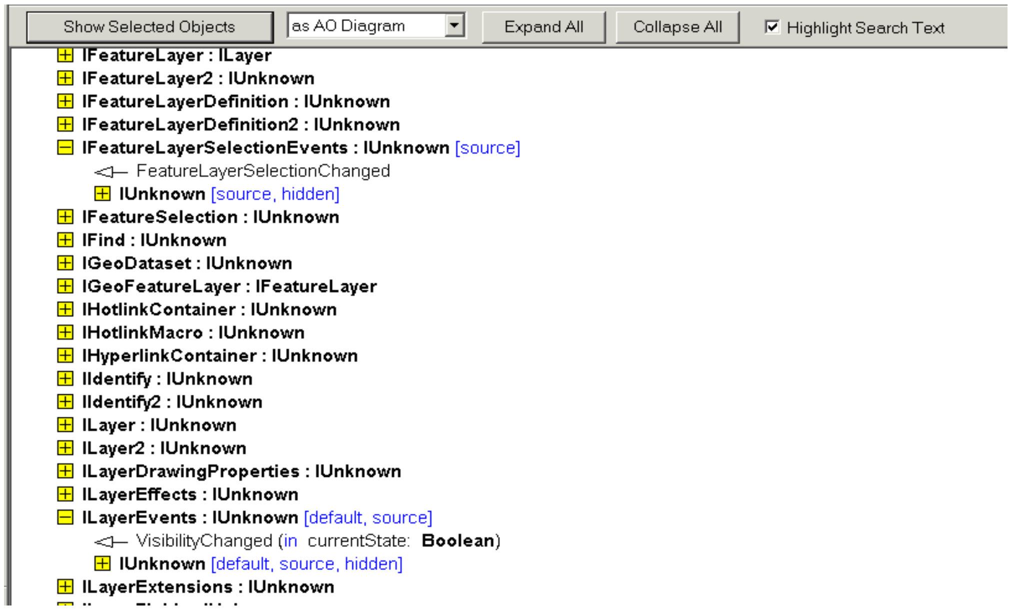
Search For: FeatureLayer
[✓ Exact] Contains
[✓ Coclass Name]
as IDL
[✓ Highlight Search Text]

☐ Click *Search*.

☐ Click on the returned result to display all interfaces supported by the FeatureLayer class.

☐ Change the Show Selected Objects from *As IDL* to *As AO Diagram.*

☐ Expand both the *IFeatureLayerSelectionEvents* and *ILayerEvents* interfaces.



```
⊞ IFeatureLayer : ILayer
⊞ IFeatureLayer2 : IUnknown
⊞ IFeatureLayerDefinition : IUnknown
⊞ IFeatureLayerDefinition2 : IUnknown
⊟ IFeatureLayerSelectionEvents : IUnknown [source]
     FeatureLayerSelectionChanged
   ⊞ IUnknown [source, hidden]
⊞ IFeatureSelection : IUnknown
⊞ IFind : IUnknown
⊞ IGeoDataset : IUnknown
⊞ IGeoFeatureLayer : IFeatureLayer
⊞ IHotlinkContainer : IUnknown
⊞ IHotlinkMacro : IUnknown
⊞ IHyperlinkContainer : IUnknown
⊞ IIdentify : IUnknown
⊞ IIdentify2 : IUnknown
⊞ ILayer : IUnknown
⊞ ILayer2 : IUnknown
⊞ ILayerDrawingProperties : IUnknown
⊞ ILayerEffects : IUnknown
⊟ ILayerEvents : IUnknown [default, source]
     VisibilityChanged (in currentState: Boolean)
   ⊞ IUnknown [default, source, hidden]
⊞ ILayerExtensions : IUnknown
```

Two outbound interfaces are listed, IFeatureLayerSelectionEvents and ILayerEvents in the ESRI Object Browser. Notice that ILayerEvents is listed as the default, whereas IFeatureLayerSelectionEvents is a secondary outbound interface. Because you want to capture the Visibility changes event, you will use ILayerEvents as it has the appropriate event.

☐ Next, return to the Visual Basic Editor and declare a variable using the *WithEvents* keyword, as shown below.

```
Private WithEvents m_pFLayer As FeatureLayer
```

Notice that the declaration statement above does not use an interface. Declaring a variable *WithEvents* is known as *implementing an outbound interface*. In other words, you are not using an interface in order to get at methods and properties of an object. Instead, you will be providing code for the object to execute each time the associated events fire.

Because you are using the ILayerEvents interface, which is the default, you declared the variable (m_pFLayer) as the name of the class (FeatureLayer). Had you wanted to use a secondary interface such as IFeatureLayerSelectionEvents, you would have declared the variable as the name of the interface minus the “I” (FeatureLayerSelectionEvents).

☐ Close the ESRI Object Browser.

☐ After you have declared the variable above, pull down the object list (upper-left) of the *ThisDocument* module. You should see a listing for the *m_pFLayer* variable.

Question 2: Now pull down the procedure list (upper-right). You should see all events supported by the FeatureLayer class. How many events does it support?

____

☐ Choose the *VisibilityChanged* event in the procedure list (it should be the only one).

Notice that the stub code for the event procedure is added to the module for you (as with the MxDocument events).

Question 3: When do you think this event will fire? ________________________
What is passed into the procedure?_______________________________________

☐ Write the If Then block below to test your code.

```
If currentState Then
   MsgBox "It’s turned on"
Else
```

```
    MsgBox "It's turned off"
End If
```

The *currentState* parameter for the event procedure is a Boolean (True/False) that specifies whether or not the layer is currently checked on or off in the Table of Contents. The code above should report the state in a message box when the layer is turned on or off.

Question 4: Bring ArcMap to the front of the display. Turn a layer on and off. Why doesn't the message box appear? ______________________________________

STEP 2: TAP INTO A LAYER'S EVENTS

The code for the VisibilityChanged procedure is not executing because it has not been associated with a particular layer. In other words, the m_pFLayer variable was declared (with events), but it was never instantiated (set equal to an object). You will now write some code to set the m_pFLayer variable (and the other module-level variables) when the document is opened.

☐ Return to the *Visual Basic Editor* and navigate to the *MxDocument_OpenDocument* event procedure.

☐ Write the following code to initialize the module-level variables when the document opens.

```
Set m_pMxDoc = ThisDocument
Set m_pFLayer = m_pMxDoc.FocusMap.Layer(3) 'The 4th layer is the US states layer
```

Now you will write code to store the layer's extent in the m_pUSExtent variable. The Extent property is on the IGeoDataset interface, so you must first QI to this interface.

☐ Use a Dim statement to declare the following pointer to the IGeoDataset interface.

```
pGDataset As IGeoDataset
```

☐ Set pGDataset equal to the same object that m_pFLayer is pointing to (QueryInterface).

☐ Write the code below to initialize the m_pUSExtent variable.

```
Set m_pUSExtent = pGDataset.Extent
```

You will now test your code by closing and opening the map.

☐ Save your map to your student directory.

☐ Click the *New Map* button to open a new map (the only way to close the current map is to open another one).

☐ Reopen the map you just saved (ex16b.mxd, unless you saved it with another name).

Question 5: Turn the United States layer on and off. Does the message box appear?

____

Turn some of the other layers on and off. Does the message box appear?

____

STEP 3: CHANGE THE MAP EXTENT WHEN LAYER VISIBILITY CHANGES

You will now add code to do something more than display a message box in response to the layer's visibility changing. The extent of the map display should zoom to the United States when the layer is turned on, then zoom out to the extent of the world when it is turned off. Because you have already set the active view's extent in an earlier exercise, you will not get specific instructions on how to do it here.

☐ Rewrite the *If Then* statement in the *m_pFLayer_VisibilityChanged* procedure so that when the United States layer is turned on, the active view's extent is set to the extent of that layer. When it is turned off, set the active view's extent equal to the extent of all the layers in the map. Write this code before each message box.

Hint: The *FullExtent* property represents all the layers in the map.

☐ After the code that sets the extent and before the message box, refresh the display.

☐ Save your map.

Question 6: Bring ArcMap to the front of the display. Turn the United States layer on and off. Does the extent change? ____________________________________

NOTE: You may have to close and then reopen your map in order to reinitialize the required variables.

If you have time, you might want to experiment with more object events by trying one of the challenge steps below. Otherwise, save your map to the student directory.

CHALLENGE: DISPLAY FILE INFORMATION FOR SELECTED FILES IN ARCCATALOG

One complaint that ArcCatalog users sometimes have is that file information, such as that found in Windows Explorer, is not displayed by default. It might be useful, for example, to know the file size and modification date for selected files. In this challenge step, you will write code to respond to a file being highlighted in ArcCatalog by calling a procedure to display file information in the ArcCatalog status bar.

☐ Start *ArcCatalog*.

☐ Start the *Visual Basic Editor*. Import the standard code module called *CatalogMacros* from *C:\Student\IPAO\Samples\ArcCatalog*.

☐ Open this module and review the code in the ReportFileProps procedure.

☐ Use the online Help, object browser, or object model diagrams to find an object that responds to a file being selected in the catalog.

☐ Write code to handle this event by calling the ReportFileProps procedure.

Question 7: Test your code by selecting various files. Does it work? _____ Are the properties accurately reported for every possible file type?___________________

NOTE: The information will be displayed as a message in the lower left portion of ArcCatalog.

Exit ArcCatalog when you are finished testing your code.

EXERCISE END

ANSWERS TO EXERCISE 16B QUESTIONS

Question 1: Pull down the procedure list at the upper-right of the module. Notice that all the events related to the MxDocument class are listed here. How many events does the document have?

Answer: 8

Question 2: Now pull down the procedure list (upper-right). You should see all events supported by the FeatureLayer class. How many events does it support?

Answer: 1

Question 3: When do you think this event will fire? What is passed into the procedure?

Answer: Whenever the visibility for a layer changes (it is checked on or off in the Table of Contents). A boolean value is passed in that specifies whether or not the layer is checked on or off.

Question 4: Bring ArcMap to the front of the display. Turn a layer on and off. Why doesn't the message box appear?

Answer: The VisibilityChanged event has not been associated with a particular layer yet.

Question 5: Turn the United States layer on and off. Does the message box appear? Turn some of the other layers on and off. Does the message box appear?

Answer: Yes, the message box should appear for the United States layer, but not the other layers.

Question 6: Bring ArcMap to the front of the display. Turn the United States layer on and off. Does the extent change?

Answer: Yes, the extent should change. If not, error check your code. You may also have to close and reopen your map to re-initialize the variables.

Question 7: Test your code by selecting various files. Does it work? Are the properties accurately reported for every possible file type?

Answer: Yes. If not, error check your code.

SOLUTIONS

```
'**Ex16b.mxd - ThisDocument module
'
Private m_pMxDoc As IMxDocument
Private m_pUSExtent As IEnvelope
Private WithEvents m_pFLayer As FeatureLayer

Private Function MxDocument_OpenDocument() As Boolean
  Set m_pMxDoc = ThisDocument
  Set m_pFLayer = m_pMxDoc.FocusMap.Layer(3)

  Dim pGDataset As IGeoDataset
  Set pGDataset = m_pFLayer
  Set m_pUSExtent = pGDataset.Extent
End Function

Private Sub m_pFLayer_VisibilityChanged(ByVal currentState As Boolean)
    If currentState Then
        m_pMxDoc.ActiveView.Extent = m_pUsExtent
            m_pMxDoc.ActiveView.PartialRefresh esriViewGeography, Nothing, Nothing
        MsgBox "It's turned on"
    Else
         m_pMxDoc.ActiveView.Extent = m_pMxDoc.ActiveView.FullExtent
         m_pMxDoc.ActiveView.PartialRefresh esriViewGeography, Nothing, Nothing
        MsgBox "It's turned off"
    End If
End Sub

'**Challenge
Private WithEvents m_pGxSelection As GxSelection

Private Sub GxDocument_OpenDocument()
    Dim pGxApplication As IGxApplication
    Set pGxApplication = Application
    Set m_pGxSelection = pGxApplication.Catalog.Selection
End Sub

Private Sub m_pGxCatalog_OnSelectionChanged(ByVal Selection As IGxSelection, _
initiator As Variant)
    Call ReportFileProps
End Sub
```

17

Creating COM classes with Visual Basic (Optional)

contents

EXERCISE 17: USE VISUAL BASIC TO CREATE A CUSTOM COM COMPONENT (OPTIONAL)

In this exercise, you will use a stand-alone programming environment to produce a custom control. Your control will be compiled as its own DLL (dynamic-link library), and then added to ArcMap using the Add From File option in the Customize dialog box.

EXERCISE SHORTCUT

1. Create a new Visual Basic 6 project, with a class named *ToggleSelectedLayer.* Reference the libraries that contain ICommand, IApplication and IMxDocument.

2. Implement ICommand in the ToggleSelectedLayer class and stub out all of its members.

3. Define the control's *Caption, Category, Checked, Message, Name and Tooltip* with values indicating that the button will turn a layer on and off. Set the *Bitmap* property equal to a picture from ..\Student\forms\frmPics.

4. Implement variables to reference the Application and Document. Code the *OnClick* event to turn the selected layer on or off. Compile the class as a .dll.

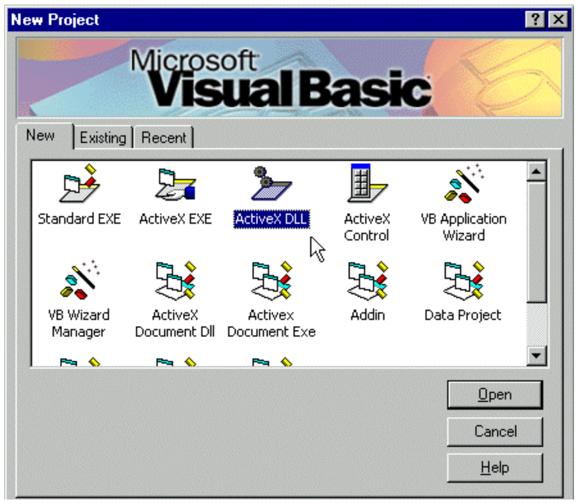
5. Use the customize dialog to add the .dll. Test the toggle selected button in ArcMap.

STEP 1: CREATE A NEW VISUAL BASIC PROJECT

☐ From the Windows *Start* menu, launch *Microsoft Visual Basic*.

When the program starts, you will have the option of creating one of several different project types.

☐ Choose to create a new *ActiveX DLL* project, then press *Open*.

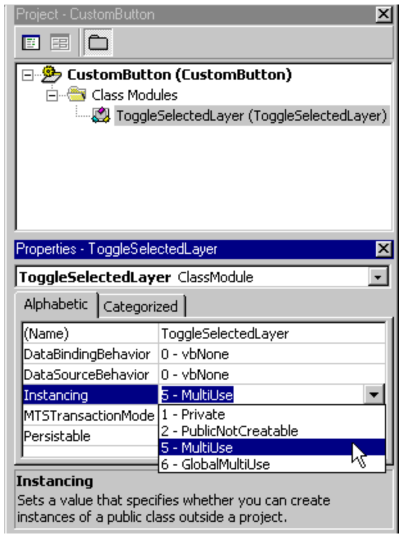


When the new Visual Basic ActiveX DLL project opens, you will have a single class module, called *Class1* by default. When your DLL is compiled, each class module in the project will become a class. In a single DLL, therefore, you could potentially deliver dozens of objects (buttons, tools, toolbars, etc.). The DLL you produce in the following steps will define a single button for use in ArcMap.

☐ Use the Visual Basic *Properties* window to change the name of the class module to **`ToggleSelectedLayer`**. This will be the name of the new control you deliver.

☐ Make sure that the instancing property of the class module is set to *Multiuse*. This will ensure that your user can create an instance of the new control.

☐ Change the name of the project to `CustomButton` (also using the properties window).



CustomButton (CustomButton)
Class Modules
ToggleSelectedLayer (ToggleSelectedLayer)

ToggleSelectedLayer ClassModule

(Name)	ToggleSelectedLayer
DataBindingBehavior	0 - vbNone
DataSourceBehavior	0 - vbNone
Instancing	5 - MultiUse
MTSTransactionMode	
Persistable	

1 - Private
2 - PublicNotCreatable
5 - MultiUse
6 - GlobalMultiUse

Instancing
Sets a value that specifies whether you can create instances of a public class outside a project.

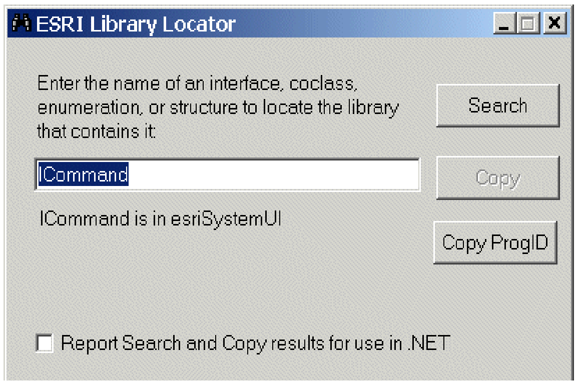
When programming in the Visual Basic for Applications environment in ArcMap or ArcCatalog, it is not necessary to explicitly make a reference to the libraries containing the ArcObjects you will be using, because they are automatically referenced. In a stand-alone development environment, however, you need to bring these objects into the project before you can begin using them in your code. Once you know the specific ArcObject you want to use, you can find which library it is contained in by using the Help, or a handy utility called the Library Locator.

☐ Open *Windows Explorer* and navigate to *C:\Program Files\ArcGIS\DeveloperKit\tools.*

☐ Double-click *LibraryLocator.exe*.

Because you know you are implementing a button that uses ICommand, you will want to find which library ICommand is contained in.

☐ In the Library Locator, type *ICommand* and click *Search*.



Enter the name of an interface, coclass, enumeration, or structure to locate the library that contains it:

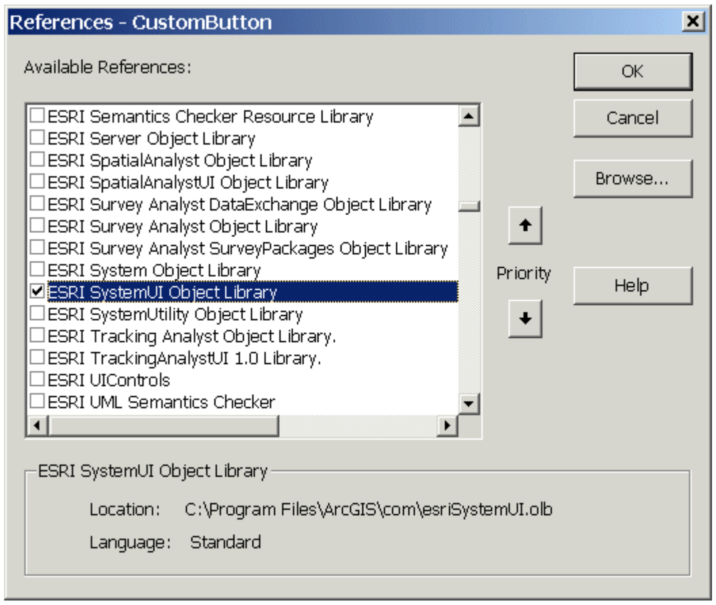
ICommand

ICommand is in esriSystemUI

Notice that the Library Locator indicates that ICommand is contained in the esriSystemUI library. You now need to make Visual Basic aware of this library.

☐ Return to Visual Basic.

☐ From the *Project* menu, choose *References*. Scroll down and check *ESRI SystemUI Object Library*, then click *OK*.



Available References:

☐ ESRI Semantics Checker Resource Library
☐ ESRI Server Object Library
☐ ESRI SpatialAnalyst Object Library
☐ ESRI SpatialAnalystUI Object Library
☐ ESRI Survey Analyst DataExchange Object Library
☐ ESRI Survey Analyst Object Library
☐ ESRI Survey Analyst SurveyPackages Object Library
☐ ESRI System Object Library
☑ ESRI SystemUI Object Library
☐ ESRI SystemUtility Object Library
☐ ESRI Tracking Analyst Object Library.
☐ ESRI TrackingAnalystUI 1.0 Library.
☐ ESRI UIControls
☐ ESRI UML Semantics Checker

ESRI SystemUI Object Library

Location: C:\Program Files\ArcGIS\com\esriSystemUI.olb

Language: Standard

☐ Using the *Library Locator*, find which libraries *IApplication* and *IMxDocument* are contained in.

Question 1: What Libraries contain the IApplication and IMxDocument interfaces?

____

☐ Return to Visual Basic and check the appropriate references.

☐ Click *OK* to close the *References* dialog.

You should have checked the ESRI Framework Object Library, ESRI ArcMapUI Object Libraries. You will see that these libraries are needed for Step 3.

STEP 2: IMPLEMENT AN ARCOBJECT INTERFACE

Remember COM? The component object model is what allows you to develop your own classes and have them communicate with ArcMap or ArcCatalog. In order to have ArcMap or ArcCatalog work with your new button, you will need to implement the interface that ArcGIS uses to communicate with controls on the user-interface.

☐ In the *General Declarations* section of your class module, type the following line of code. This is your *promise* to write all the necessary code to make this interface work.

```
Implements ICommand
```

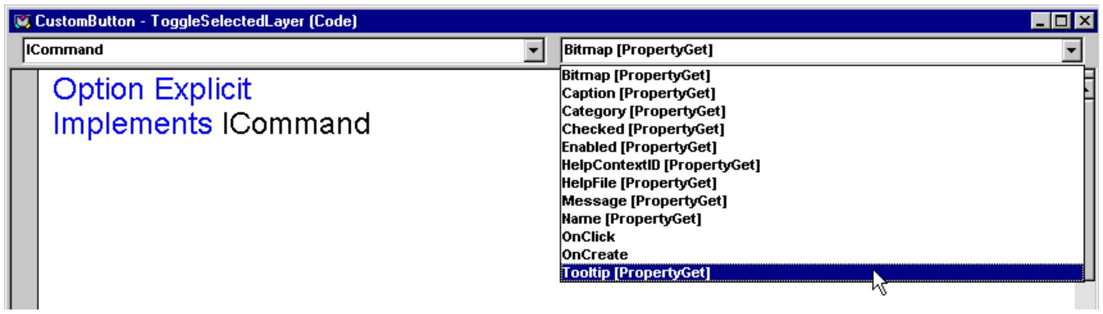
☐ Pull down the class module's object list (upper-left). You should now find a listing for the interface you promised to implement. Select *ICommand* in the object list.

NOTE: If you do not see ICommand listed here, try pressing Enter after the implements statement.

☐ Pull down the class module's procedure list (upper-right). You should see all the methods and properties defined for the ICommand interface.

☐ Select each member in the procedure list. Visual Basic will provide the stub code for each member you select.

NOTE: There is a handy Add-in called the Interface Implementer that will automatically do this for you. You can read more about this in the Developer Help.



```
Option Explicit
Implements ICommand
```

STEP 3: DEFINE YOUR CONTROL'S PROPERTIES

In the following steps, you will write code to define some of the properties of your new control. Later, you will write the code that makes your control work. The first property in your ToggleSelectedLayer class module (ICommand_Bitmap) defines the icon to appear on the control. To define this property, you will import a form to your project that has several icons to choose from.

☐ From the *Project* menu, choose *Add File*. Navigate to your Student directory. From the folder called *Forms*, add *frmPics*.

Each command button on this form has a different icon (picture property). You will use the name of a command button to assign one of these pictures to your control.

☐ Navigate to the *ICommand_Bitmap* property in the *ToggleSelectedLayer* class module.

☐ Write code like that shown below to assign one of the icons on frmPics to your control. (This code assigns the "Globe" icon).

```
Private Property Get ICommand_Bitmap() As esriSystem.OLE_HANDLE
   ICommand_Bitmap = frmPics.cmdGlobe.Picture.Handle
End Property
```

NOTE: The bitmap property is set with an OLE_HANDLE (which is a long integer), not with a *Picture* object.

☐ Navigate to the appropriate property procedures, and provide the values listed below to further define your button's appearance.

Property	**Value**
Caption	"Toggle Layer On/Off"
Category	"My Custom Controls"
Checked	False
Message	"Toggles the visibility of the selected layer"
Name	"ToggleButton"
ToolTip	"Layer On/Off"

NOTE: You will code the Enabled property in a later step. Although you will not provide values for HelpContextID and HelpFile properties, you must still provide the (empty) stub code to satisfy the rules of COM.

STEP 4: WRITE CODE TO MAKE YOUR CONTROL WORK

Unlike coding in Visual Basic for Applications, when you are programming in a stand-alone environment like Visual Basic or C++, you cannot use the preset Application and ThisDocument variables. When developing a COM component, you will have to use the method described below for referencing the current application, document, or both.

☐ Declare the following (module-level) variables in the *General Declarations* section of your class module.

```
Private m_pMxDoc As IMxDocument
Private m_pApp As IApplication
```

NOTE: The IApplication interface will not appear in the code completion list because it is the default interface on the Application class. Default interfaces are not shown in code completion. IUnknown is the default interface for most ArcObjects.

Each time your button is added to a toolbar from the Customize dialog box, its OnCreate event procedure fires. Notice the object that is passed into this procedure: hook As Object. *Hook* is actually a reference to the application (ArcMap or ArcCatalog), and can be treated the same as the Application preset variable. In order to use this object in all event procedures, you need to store it in a module-level variable.

☐ Write the code below in the *OnCreate* event procedure.

```
Set m_pApp = hook
Set m_pMxDoc = m_pApp.Document
```

The code that runs when a user clicks your button is stored in the OnClick event procedure. Now that you have a reference to the application and the document, you can write code that works with these objects. You will write some simple code to toggle the selected layer on or off.

☐ Navigate to the *OnClick* event procedure.

☐ Write the following line of code to toggle the selected layer's visibility.

```
m_pMxDoc.SelectedLayer.Visible = Not m_pMxDoc.SelectedLayer.Visible
```

☐ After toggling the layer's visibility, refresh the display and the Table of Contents with the following code.

```
m_pMxDoc.ActiveView.Refresh
m_pMxDoc.UpdateContents
```

Most implementation will occur within the OnClick procedure. Here you call additional sub procedures or call the show method for custom user forms. This example is simply toggling layers just to show implementation of ICommand.

To manage when your button is enabled or disabled, you will provide logic in the Enabled property procedure.

☐ Navigate to the *Enabled* property procedure.

☐ Write the following code to enable your button only when a layer is selected.

```
ICommand_Enabled = Not m_pMxDoc.SelectedLayer Is Nothing
```

At this point, your component should be ready to compile.

☐ Save your project to the *Student* directory.

☐ From the *File* menu, choose *Make CustomButton.dll*. Write your DLL to the student directory.

If your code was free of errors, the DLL should have been written successfully to disk.

STEP 5: TEST YOUR CONTROL IN ARCMAP

Another difference between stand-alone Visual Basic and Visual Basic for Applications is the ease of testing and debugging your code. In VBA, you are working in an *integrated programming environment*, which means you write, test, and debug your applications all within the same application. When producing COM components, you will write and compile your code in Visual Basic, test your component in ArcMap (or ArcCatalog), and then return to Visual Basic to fix errors and recompile.

☐ Start *ArcMap*.

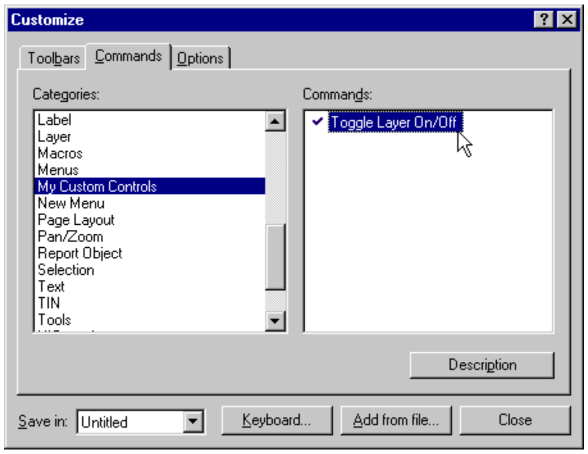
☐ Open an existing map or create a new map and add some layers.

☐ Launch the *Customize* dialog box and click the *Commands* tab. Make sure the *Save in* pulldown is set to the current map.

☐ Click the *Add From File* button on the *Customize* dialog, and navigate to your custom DLL. Select the DLL and click *Open*.

☐ ArcMap should indicate that one new object, ToggleSelectedLayer, was added to the document.

☐ Locate your control in the *Customize* dialog. It should be found in the *My Custom Controls* category, and its caption should read *Toggle Layer On/Off*.



☐ Drag the control onto one of the ArcMap toolbars.

Question 2: Close the Customize dialog and test your control. Is it enabled when you select a layer?____
Does it successfully toggle the selected layer on and off? ______________________

____

If you find errors while using your control, or want to make modifications to its code, you can return to your Visual Basic project. After making changes and before compiling your DLL again, you should specify binary compatibility with your existing DLL. To do this, go to the Project menu, choose Project properties, click on the Component tab, and select Binary Compatibility. Now when you return to ArcMap after recompiling, you will not have to add the DLL from file again.

Challenge: Keep the layer selected

Notice that when your control is clicked, the selected layer is toggled on or off, but the layer then becomes unselected. Modify your component so the selected layer remains selected in the Table of Contents when the button is clicked.

Exercise End

ANSWERS TO EXERCISE 17 QUESTIONS

Question 1: What Libraries contain the IApplication and IMxDocument interfaces?

Answer: IApplication is contained in the esriFramework library and IMxDocument is contained in the esriArcMapUI library.

Question 2: Close the Customize dialog and test your control. Is it enabled when you select a layer? Does it successfully toggle the selected layer on and off?

Answer: Yes, it should be enabled when you select a layer and work successfully. If it does not, error check your code. You will have to recompile the .dll if you make any changes to your code.

SOLUTIONS

```
Option Explicit
Implements ICommand
Private m_pMxDoc As IMxDocument
Private m_pApp As IApplication

Private Property Get ICommand_Bitmap() As esriSystem.OLE_HANDLE
     ICommand_Bitmap = frmPics.cmdCheck.Picture.Handle
End Property

Private Property Get ICommand_Caption() As String
    ICommand_Caption = "Toggle Layer On/Off"
End Property

Private Property Get ICommand_Category() As String
    ICommand_Category = "My Custom Controls"
End Property

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

Private Property Get ICommand_Enabled() As Boolean
    ICommand_Enabled = Not m_pMxDoc.SelectedLayer Is Nothing
End Property

Private Property Get ICommand_HelpContextID() As Long
End Property

Private Property Get ICommand_HelpFile() As String
End Property

Private Property Get ICommand_Message() As String
    ICommand_Message = "Toggles the visibility of the selected layer"
End Property

Private Property Get ICommand_Name() As String
    ICommand_Name = "ToggleButton"
End Property

Private Sub ICommand_OnClick()
    '* Challenge' Dim pSelLayer As ILayer
    '* Challenge' Set pSelLayer = m_pMxDoc.SelectedLayer
    m_pMxDoc.SelectedLayer.Visible = Not m_pMxDoc.SelectedLayer.Visible
    '* Challenge' m_pMxDoc.ContextItem = pSelLayer
    m_pMxDoc.ActiveView.Refresh
    m_pMxDoc.UpdateContents
End Sub

Private Sub ICommand_OnCreate(ByVal hook As Object)
    Set m_pApp = hook
    Set m_pMxDoc = m_pApp.Document
End Sub

Private Property Get ICommand_Tooltip() As String
    ICommand_Tooltip = "Layer On/Off"
End Property
```

A

ESRI data license agreement

contents

IMPORTANT—READ CAREFULLY
BEFORE OPENING THE SEALED MEDIA PACKAGE

ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE, INC. (ESRI), IS WILLING TO LICENSE THE ENCLOSED ELECTRONIC VERSION OF THIS TRAINING COURSE TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS AND CONDITIONS CONTAINED IN THIS ESRI DATA LICENSE AGREEMENT. PLEASE READ THE TERMS AND CONDITIONS CAREFULLY BEFORE OPENING THE SEALED MEDIA PACKAGE. BY OPENING THE SEALED MEDIA PACKAGE, YOU ARE INDICATING YOUR ACCEPTANCE OF THE ESRI DATA LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS AS STATED, THEN ESRI IS UNWILLING TO LICENSE THE TRAINING COURSE TO YOU. IN SUCH EVENT, YOU SHOULD RETURN THE MEDIA PACKAGE WITH THE SEAL UNBROKEN AND ALL OTHER COMPONENTS (E.G., THE CD–ROM, TRAINING COURSE MATERIALS, TRAINING DATABASE, AS APPLICABLE) TO ESRI OR ITS AUTHORIZED INSTRUCTOR FOR A REFUND. NO REFUND WILL BE GIVEN IF THE MEDIA PACKAGE SEAL IS BROKEN OR THERE ARE ANY MISSING COMPONENTS.

ESRI DATA LICENSE AGREEMENT

This is a license agreement, and not an agreement for sale, between you (Licensee) and ESRI. This ESRI data license agreement (Agreement) gives Licensee certain limited rights to use the electronic version of the training course materials, training database, software, and related materials (hereinafter collectively referred to as the “Training Course”). All rights not specifically granted in this Agreement are reserved to ESRI and its licensor(s).

Reservation of Ownership and Grant of License: ESRI and its licensor(s) retain exclusive rights, title, and ownership to the copy of the Training Course licensed under this Agreement and hereby grant to Licensee a personal, nonexclusive, nontransferable license to use the Training Course as a single package for Licensee's own personal use only pursuant to the terms and conditions of this Agreement. Licensee agrees to use reasonable efforts to protect the Training Course from unauthorized use, reproduction, distribution, or publication.

Proprietary Rights and Copyright: Licensee acknowledges that the Training Course is proprietary and confidential property of ESRI and its licensor(s) and is protected by United States copyright laws and applicable international copyright treaties and/or conventions.

Permitted Uses:

- Licensee may run the setup and install one (1) copy of the Training Course onto a permanent electronic storage device and reproduce one (1) copy of the Training Course and/or any online documentation in hard-copy format for Licensee's own personal use only.
- Licensee may use one (1) copy of the Training Course on a single processing unit.
- Licensee may make only one (1) copy of the original Training Course for archival purposes during the term of this Agreement, unless the right to make additional copies is granted to Licensee in writing by ESRI.

- Licensee may use the Training Course provided by ESRI for the stated purpose of Licensee's own personal GIS training and education.

Uses Not Permitted:

- Licensee shall not sell, rent, lease, sublicense, lend, assign, time-share, or transfer, in whole or in part, or provide unlicensed third parties access to the Training Course, any updates, or Licensee's rights under this Agreement.
- Licensee shall not separate the component parts of the Training Course for use on more than one (1) computer, used in conjunction with any other software package, and/or merged and compiled into a separate database(s) for other analytical uses.
- Licensee shall not reverse engineer, decompile, or disassemble the Training Course, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this restriction.
- Licensee shall not make any attempt to circumvent the technological measure(s) (e.g., software or hardware key) that effectively controls access to the Training Course, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this restriction.
- Licensee shall not remove or obscure any copyright, trademark, and/or proprietary rights notices of ESRI or its licensor(s).

Term: The license granted by this Agreement shall commence upon Licensee's receipt of the Training Course and shall continue until such time that (1) Licensee elects to discontinue use of the Training Course and terminates this Agreement or (2) ESRI terminates for Licensee's material breach of this Agreement. The Agreement shall automatically terminate without notice if Licensee fails to comply with any provision of this Agreement. Upon termination of this Agreement in either instance, Licensee shall return to ESRI or destroy all copies of the Training Course, and any whole or partial copies, in any form and deliver evidence of such destruction to ESRI, which evidence shall be in a form acceptable to ESRI in its sole discretion. The parties hereby agree that all provisions that operate to protect the rights of ESRI and its licensor(s) shall remain in force should breach occur.

Limited Warranty and Disclaimer: ESRI warrants that the media upon which the Training Course is provided will be free from defects in materials and workmanship under normal use and service for a period of ninety (90) days from the date of receipt.

EXCEPT FOR THE LIMITED WARRANTY SET FORTH ABOVE, THE TRAINING COURSE CONTAINED THEREIN IS PROVIDED “AS-IS,” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. ESRI DOES NOT WARRANT THAT THE TRAINING COURSE WILL MEET LICENSEE'S NEEDS OR EXPECTATIONS; THAT THE USE OF THE TRAINING COURSE WILL BE UNINTERRUPTED; OR THAT ALL NONCONFORMITIES, DEFECTS, OR ERRORS CAN OR WILL BE CORRECTED. THE TRAINING DATABASE HAS BEEN OBTAINED FROM SOURCES BELIEVED TO BE RELIABLE, BUT ITS ACCURACY AND COMPLETENESS, AND THE OPINIONS BASED THEREON, ARE NOT GUARANTEED. THE TRAINING DATABASE MAY CONTAIN SOME NONCONFORMITIES, DEFECTS, ERRORS, AND/OR OMISSIONS. ESRI AND ITS LICENSOR(S) DO NOT WARRANT THAT THE TRAINING DATABASE WILL MEET LICENSEE'S NEEDS OR EXPECTATIONS, THAT THE USE OF THE TRAINING DATABASE WILL BE UNINTERRUPTED, OR THAT ALL NONCONFORMITIES CAN OR WILL BE CORRECTED. ESRI AND ITS LICENSOR(S) ARE NOT INVITING RELIANCE ON THIS TRAINING DATABASE, AND LICENSEE SHOULD ALWAYS VERIFY ACTUAL DATA, WHETHER MAP, SPATIAL, RASTER, TABULAR INFORMATION, AND SO FORTH. THE DATA CONTAINED IN THIS PACKAGE IS SUBJECT TO CHANGE WITHOUT NOTICE.

Exclusive Remedy and Limitation of Liability: During the warranty period, Licensee's exclusive remedy and ESRI's entire liability shall be the return of the license fee paid for the Training Course upon the Licensee's deinstallation of all copies of the Training Course and providing a Certification of Destruction in a form acceptable to ESRI.

IN NO EVENT SHALL ESRI OR ITS LICENSOR(S) BE LIABLE TO LICENSEE FOR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOST SALES OR BUSINESS EXPENDITURES, INVESTMENTS, OR COMMITMENTS IN CONNECTION WITH ANY BUSINESS, LOSS OF ANY GOODWILL, OR FOR ANY INDIRECT, SPECIAL, INCIDENTAL, AND/OR CONSEQUENTIAL DAMAGES ARISING OUT OF THIS AGREEMENT OR USE OF THE TRAINING COURSE, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY, AND WHETHER OR NOT ESRI OR ITS LICENSOR(S) HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

No Implied Waivers: No failure or delay by ESRI or its licensor(s) in enforcing any right or remedy under this Agreement shall be construed as a waiver of any future or other exercise of such right or remedy by ESRI or its licensor(s).

Order for Precedence: This Agreement shall take precedence over the terms and conditions of any purchase order or other document, except as required by law or regulation.

Export Regulation: Licensee acknowledges that the Training Course and all underlying information or technology may not be exported or re-exported into any country to which the U.S. has embargoed goods, or to anyone on the U.S. Treasury Department's list of Specially Designated Nationals, or to the U.S. Commerce Department's Table of Deny Orders. Licensee shall not export the Training Course or any underlying information or technology to any facility in violation of these or other applicable laws and regulations. Licensee represents and warrants that it is not a national or resident of, or located in or under the control of, any country subject to such U.S. export controls.

Severability: If any provision(s) of this Agreement shall be held to be invalid, illegal, or unenforceable by a court or other tribunal of competent jurisdiction, the validity, legality, and enforceability of the remaining provisions shall not in any way be affected or impaired thereby.

Governing Law: This Agreement, entered into in the County of San Bernardino, shall be construed and enforced in accordance with and be governed by the laws of the United States of America and the State of California without reference to conflict of laws principles.

Entire Agreement: The parties agree that this Agreement constitutes the sole and entire agreement of the parties as to the matter set forth herein and supersedes any previous agreements, understandings, and arrangements between the parties relating hereto.

B

Workshop exercises

contents

EXERCISE B1: ADD A LAYER FILE (*.LYR) TO ARCMAP

In this exercise, you will write a sub procedure that adds predefined layer files (on disk) as layers to ArcMap.

- Difficulty:
- Objects used: FeatureLayer, GxLayer, Map, MxDocument
- Interfaces used: IFeatureLayer, IGxFile, IGxLayer, IMap, IMxDocument
- Relevant OMDs: ArcMap, Geodatabase, Map Layer

BASIC STEPS

☐ Create a new GxLayer object.

☐ QueryInterface to its IGxFile interface.

☐ Set the GxLayer object's Path property to a layer file (you will find some in the class database).

☐ Declare a variable as ILayer.

☐ Set the layer variable by getting the GxLayer's Layer property.

☐ Add the layer to the document's focused map.

EXERCISE END

SOLUTION B1: ADD A LAYER FILE TO ARCMAP

```
Sub AddLayerFromFile()
  Dim pMxDoc As IMxDocument
  Dim pGxLayer As IGxLayer
  Dim pGxFile As IGxFile

  Set pGxLayer = New GxLayer                ' *.lyr file ...
  Set pGxFile = pGxLayer                     'QueryInterface

  pGxFile.Path = "C:\Student\IPAO\Data\World\Country.lyr"

  Set pMxDoc = ThisDocument                  'QueryInterface

  Dim pFLayer As IFeatureLayer
  Set pFLayer = pGxLayer.Layer  ’Get the layer from the file
  pMxDoc.FocusMap.AddLayer pFLayer            'Add the layer
End Sub
```

EXERCISE B2: ZOOM TO A LAYER’S SELECTED FEATURES

In this exercise, you will write code to zoom the ArcMap display to the extent of all selected features in a feature layer.

- Difficulty:
- Objects used: Envelope, Feature, FeatureClass, FeatureCursor, FeatureLayer, Map, MxDocument, SelectionSet
- Interfaces used: IEnvelope, IFeature, IFeatureCursor, IFeatureSelection, IMap, IMxDocument, ISelectionSet
- Relevant OMDs: ArcMap, Geodatabase, Geometry, Map Layer

BASIC STEPS

☐ Declare a variable as IFeatureSelection.

☐ Set the above variable equal to the document’s selected layer.

☐ Get the SelectionSet from the layer.

☐ Put the selected features into a feature cursor.

☐ Make a new Envelope object for the new map extent.

☐ Loop through all features in the feature cursor.

☐ Inside the loop, get each feature shape’s envelope and union it with the new envelope you made above.

☐ When the loop has completed, set the active view’s extent with the new envelope.

☐ Refresh the display.

EXERCISE END

SOLUTION B2: ZOOM TO A LAYER'S SELECTED FEATURES

```
Public Sub ZoomToSelected()
  Dim pMxDoc As IMxDocument
  Set pMxDoc = ThisDocument

  Dim pLayer As IFeatureSelection
  Set pLayer = pMxDoc.SelectedLayer
  If pLayer Is Nothing Then Exit Sub

  Dim pSelSet As ISelectionSet
  Set pSelSet = pLayer.SelectionSet   'Get layer’s selection

  Dim pFCursor As IFeatureCursor
  pSelSet.Search Nothing, False, pFCursor     'create cursor

  Dim pEnv As IEnvelope
  Dim pBigEnv As IEnvelope
  Set pBigEnv = New Envelope    'This will be the new extent
  Dim pFeature As IFeature
  Set pFeature = pFCursor.NextFeature
  Do Until pFeature Is Nothing       'Loop thru each feature
      Set pEnv = pFeature.Shape.Envelope
      pBigEnv.Union pEnv  'Union each envelope into a big one
      Set pFeature = pFCursor.NextFeature
  Loop

  pMxDoc.ActiveView.Extent = pBigEnv         'Set the extent
  pMxDoc.ActiveView.Refresh      'Must refresh to see change
End Sub
```

EXERCISE B3: CALCULATE TOTAL AREA, LENGTH, OR COUNT

In this exercise, you write an all-purpose function that takes a FeatureClass as input and returns the total area if the features are polygons, the total length if they are lines, or a simple count if they are point features. This exercise is rated as high difficulty due to vague instructions.

- Difficulty:

- Objects used: Feature, FeatureClass, FeatureCursor, Fields, Field
- Interfaces used: IFeature, IFeatureClass, IFeatureCursor, IGeometry
- Relevant OMDs: Geodatabase

BASIC STEPS

☐ Define a new function. The function should take a feature class as a parameter and return a long integer.

☐ Put all the features from the (passed in) feature class into a search cursor.

☐ Write a Select Case statement based on the input feature class feature type (point, line, polygon).

☐ If the feature type is point, loop through the cursor and get the total count of features. Return the count.

☐ If the feature type is line, loop through the cursor and calculate total length of all features. Return the length.

☐ If the feature type is polygon, loop through the cursor and calculate total area. Return the area.

☐ If the feature type is something other than point, line, or polygon, return a flag that lets the calling function know an error was encountered.

☐ Call your function from another procedure. Test it by passing in point, line, and polygon feature classes. Display the count, length, or area in a message box.

EXERCISE END

SOLUTION B3: CALCULATE TOTAL AREA, LENGTH, OR COUNT

```
Public Function AreaLgthCnt(FClass As IFeatureClass) As Long
  Dim pFCursor As IFeatureCursor
  Dim pFeature As IFeature
  Set pFCursor = FClass.Search(Nothing, False)   'Get Cursor

  Select Case FClass.ShapeType
  Case 1, 2                                           'Point
      Dim lngCount As Long
      Set pFeature = pFCursor.NextFeature
      Do Until pFeature Is Nothing
          lngCount = lngCount + 1
          Set pFeature = pFCursor.NextFeature
      Loop
      CalcAreaLengthCount = lngCount           'Return count
  Case 3, 6, 13 To 16                                  'Line
      Dim lngLength As Long
      Dim pCurve As ICurve
      Set pFeature = pFCursor.NextFeature
      Do Until pFeature Is Nothing
          Set pCurve = pFeature.Shape
          lngLength = lngLength + pCurve.Length
          Set pFeature = pFCursor.NextFeature
      Loop
      CalcAreaLengthCount = lngLength         'Return length
  Case 4, 5, 11                                     'Polygon
      Dim lngArea As Long
      Dim pPoly As IArea
      Set pFeature = pFCursor.NextFeature
      Do Until pFeature Is Nothing
          Set pPoly = pFeature.Shape
          lngArea = lngArea + pPoly.Area
          Set pFeature = pFCursor.NextFeature
      Loop
      CalcAreaLengthCount = lngArea             'Return area
  Case Else                                           'Other
      CalcAreaLengthCount = -99                 'Return flag
  End Select
End Function

'Code to call the above function for the selected layer ...
Public Sub ReportAreaLengthCount()
  Dim pMxDoc As IMxDocument
  Dim pFLayer As IFeatureLayer

  Set pMxDoc = ThisDocument
  Set pFLayer = pMxDoc.SelectedLayer

  If pFLayer Is Nothing Then Exit Sub
  If Not TypeOf pFLayer Is IFeatureLayer Then Exit Sub

  Dim lngInfo As Long
  lngInfo = CalcAreaLengthCount(pFLayer.FeatureClass)
  MsgBox lngInfo
End Sub
```

EXERCISE B4: MAKE A SCALE-DEPENDENT RENDERER

In this exercise, you will create a scale-dependent renderer to display a layer with different symbology according to the scale at which it is being viewed. This exercise is rated as high difficulty because of the many steps required.

- Difficulty: █████░

- Objects used: FeatureLayer, Map, MxDocument, RgbColor, ScaleDependentRenderer, SimpleFillSymbol, SimpleLineSymbol, SimpleRenderer
- Optional objects: UniqueValueRenderer, ClassBreaksRenderer, Classify
- Interfaces used: IFeatureLayer, IFeatureRenderer, IMap, IMxDocument, IRgbColor, ISymbol
- Optional interfaces: IUniqueValueRenderer, IClassBreaksRenderer, IClassify, IEqualArea
- Relevant OMDs: Geodatabase, Map Layer, ArcMap

BASIC STEPS

☐ Reference a feature layer in the map.

☐ Create a variable that points to an interface on FeatureLayer that allows you to get or set the renderer.

☐ Reference the layer's feature class.

☐ Make a new SimpleFillSymbol.

☐ Set the fill symbol's color (any color you like).

☐ Set the fill symbol's style.

☐ Get the line symbol from the fill symbol (outline).

☐ Set the line symbol's color and width.

☐ Set the line symbol back to the fill symbol.

☐ Make a new SimpleRenderer.

☐ Set the renderer's symbol property using the fill symbol.

☐ Create at least one more renderer. You may create as many as you like. Each one will be assigned to a different scale break.

☐ (Intermediate) Make one or more SimpleRenderers by following the method above.

☐ (Advanced) Make one or more unique value or class break renderers. You will need to also create a color ramp object to define the colors for each class. For a UniqueValueRenderer, you will need to set each unique value by looping through a cursor of all features in the feature class. For a ClassBreaksRenderer, you can set the break points manually (recommended) or work with a Classify object to create breaks using a statistical method.

☐ Create a new ScaleDependentRenderer.

☐ Add the renderers you created above (you need at least two to see it work).

☐ Assign the scale thresholds for when each renderer should display.

☐ Assign the ScaleDependentRenderer to the layer.

☐ Refresh the display. Update the document's contents.

☐ Zoom in and out on the display. You should see different renderers used to display the layer when you break your scale thresholds.

EXERCISE END

SOLUTION B4: MAKE A SCALE-DEPENDENT RENDERER

```
Public Sub AssignScaleDependentRenderer()
'**Reference the required objects ...
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument
    Dim pGFLayer As IGeoFeatureLayer
    Set pGFLayer = pMxDoc.FocusMap.Layer(0)
    Dim pFClass As IFeatureClass
    Set pFClass = pGFLayer.FeatureClass

'**The code below makes a simple renderer ...
    Dim pSimpleRndr As ISimpleRenderer
    Set pSimpleRndr = New SimpleRenderer
    Dim pFillSym As ISimpleFillSymbol
    Set pFillSym = New SimpleFillSymbol
    Dim pColor As IRgbColor
    Set pColor = New RgbColor
    pColor.Red = 0
    pColor.Green = 170
    pColor.Blue = 230
    pFillSym.Color = pColor
    Dim pLineSym As ISimpleLineSymbol
    Set pLineSym = pFillSym.Outline
    pLineSym.Color = pColor
    pFillSym.Outline = pLineSym
    Set pSimpleRndr.Symbol = pFillSym

'**The code below makes a unique value renderer ...
    Dim pUniqueRndr As IUniqueValueRenderer
    Set pUniqueRndr = New UniqueValueRenderer
    pUniqueRndr.FieldCount = 1
    pUniqueRndr.Field(0) = "State_Name"
'--Put all features into a cursor ...
    Dim pFCursor As IFeatureCursor
    Set pFCursor = pFClass.Search(Nothing, True)
    Dim pFeature As IFeature
    Set pFeature = pFCursor.NextFeature
    Dim intStateFld As Integer
    intStateFld = pFeature.Fields.FindField("State_Name")
    Dim blnValueExists As Boolean
    Dim intValues As Integer
'--Add each feature's state name value to the renderer ...
'--Only add a value if it doesn't already exist ...
    Do Until pFeature Is Nothing
        blnValueExists = False
        For intValues = 0 To (pUniqueRndr.ValueCount - 1)
      If pUniqueRndr.Value(intValues) = pFeature.Value(intStateFld) Then
                blnValueExists = True
                Exit For
            End If
        Next intValues
        If Not blnValueExists Then
     pUniqueRndr.AddValue pFeature.Value(intStateFld), "State", pFillSym
        End If
        Set pFeature = pFCursor.NextFeature
    Loop
'--Make a ramp of 50 colors for the states ...
    Dim pClrRamp As IRandomColorRamp
    Set pClrRamp = New RandomColorRamp
    pClrRamp.Size = pUniqueRndr.ValueCount
    pClrRamp.CreateRamp True
'--Put the colors into an enum ...
    Dim pEnumColor As IEnumColors
    Set pEnumColor = pClrRamp.Colors
    pEnumColor.Reset
'--Loop thru to assign colors for each value ...
    Dim strStateName As String
    For intValues = 0 To (pUniqueRndr.ValueCount - 1)
        strStateName = pUniqueRndr.Value(intValues)
        If strStateName <> "" Then
            Set pFillSym = New SimpleFillSymbol
            Set pLineSym = pFillSym.Outline
            pFillSym.Color = pEnumColor.Next
            pLineSym.Color = pFillSym.Color
            pFillSym.Outline = pLineSym
            pUniqueRndr.Symbol(strStateName) = pFillSym
        End If
    Next intValues

'**Finally! Create a scale-dependent renderer ...
'--Add the renderers created above ...
    Dim pSDRndr As IScaleDependentRenderer
    Set pSDRndr = New ScaleDependentRenderer
    pSDRndr.AddRenderer pUniqueRndr
    pSDRndr.AddRenderer pSimpleRndr
'--Set the corresponding scale thresholds ...
    pSDRndr.Break(0) = 22000000
    pSDRndr.Break(1) = 100000000
'--Assign the new renderer to the layer ...
    Set pGFLayer.Renderer = pSDRndr
'--Refresh the map and the TOC ...
    pMxDoc.ActiveView.Refresh
    pMxDoc.UpdateContents
End Sub
```

EXERCISE B5: USE THE ARC AUTOMATION SERVER

In this exercise, you will use some additional ArcInfo libraries in order to run ArcInfo Workstation commands from your VBA application. Your code will run an ArcInfo Describe command from a VBA form, then report the results in a text box control on the form.

- Difficulty:

- Additional libraries used: ESRI Arc automation server, ESRIutil automation server
- Objects used: Arc, Strings (and Form controls)
- ArcObjects used: <none>
- Interfaces used: <none>
- Relevant OMDs: <none>

BASIC STEPS

☐ From the *Tools* menu in the Visual Basic Editor, choose *References*.

☐ Scroll through the list of additional libraries and check on these two libraries: ESRI Arc automation server, ESRIutil automation server.

☐ From the *File* menu, choose *Import*. Import the form called *frmDescribe* from the *Samples* directory of your class database. The form already has some of the code and the controls you will need.

☐ Run the form to check the code that has been written. Click the *ArcCatalog* button on the form. Choose an ArcInfo coverage; notice that the coverage name is displayed in a label on the form. Your code will pass the coverage specified to an ArcInfo *Describe* command.

☐ Open the form's code module. Navigate to the *Click* procedure for cmdDescribe.

☐ In the *cmdDescribe_Click* procedure, declare two variables: one as ESRI.Arc (use the library name) and one as ESRIutil.Strings.

☐ Set the variables equal to new ESRI.Arc and new ESRIutil.Strings objects.

☐ Call the *Command* method on your ESRI.Arc variable; this is like sending a command to ArcInfo.

☐ For the command string, use "Describe" & lblCoverName.Caption. Remember that the text (caption) on the label will be a pathname to a coverage.

☐ For the other required parameter, use your ESRI.Strings variable.

☐ Write code to clear the list box on the form (lboDescribeInfo).

☐ Declare a variable to manage a loop (as an integer).

☐ Write a For Next loop using the variable above. Loop from 0 to the number of strings in the ESRI.Strings variable - 1.

☐ Inside the loop, add an item to the list box (lboDescribeInfo).

☐ The item to add will be an item from the ESRI.Strings collection. Use the loop index variable to specify the item. Also add a carriage return to the list box (vbCrLf).

☐ Close the loop.

☐ Run the form. Choose a coverage. Click *Describe*.

☐ You should see the familiar information from an ArcInfo Describe command listed on the form.

EXERCISE END

SOLUTION B5: USE THE ARC AUTOMATION SERVER

```
'**Declare a variable for the GxDialog
Private m_pGxDialog As IGxDialog

'**When the form initializes, make the GxDialog
'--(This will make the dialog launch faster)
Private Sub UserForm_Initialize()
  Dim pGxObjFilter As IGxObjectFilter
  Set pGxObjFilter = New GxFilterCoverages

  Set m_pGxDialog = New GxDialog
  m_pGxDialog.AllowMultiSelect = False
  Set m_pGxDialog.ObjectFilter = pGxObjFilter
  m_pGxDialog.StartingLocation = "C:\Student\IPAO"
  m_pGxDialog.Title = "Specify a coverage to describe ..."
End Sub

'**Command button to show the GxDialog, get a cover name ...
Private Sub cmdOpenCover_Click()
  Dim pEnumGxObjs As IEnumGxObject
  Dim pGxObj As IGxObject
  m_pGxDialog.DoModalOpen Application.hWnd, pEnumGxObjs
  Set pGxObj = pEnumGxObjs.Next
  lblCoverName.Caption = pGxObj.FullName
End Sub

'**Command button to execute an ArcInfo "Describe" command
Private Sub cmdDescribe_Click()
'--Declare a variable as "Arc", one as "Strings" from the
'--Arc automation libraries. "Arc" is like the Arc prompt,
'--"Strings" is a collection of the info returned from a
'--command (the info that would be printed to the terminal)
  Dim pArc As ESRI.Arc
  Dim pStrings As ESRIutil.Strings

  Set pArc = New ESRI.Arc
  Set pStrings = New ESRIutil.Strings
'--Use the "Command" method. The parameters are 1)the
'--command that would be typed at the "Arc" prompt, and the
'--Strings variable to contain the returned info ...
  pArc.Command "Describe " & lblCoverName.Caption, pStrings

'--Clear any existing text from the list box, loop thru the
'--Strings collection, put the info in the list box ...
  lboDescribeInfo.Clear
  Dim intStringCount As Integer
  For intStringCount = 0 To pStrings.Count - 1
       lboDescribeInfo.AddItem pStrings.Item(intStringCount) & vbCrLf
  Next intStringCount
End Sub
```

EXERCISE B6: CODE OBJECT EVENTS

In this exercise, you will write code to automatically send raster layers to the bottom of the ArcMap Table of Contents. Your code will run in response to map events.

- Difficulty:
- Objects used: Map, MxDocument, RasterLayer, Variant
- Interfaces used: IActiveViewEvents, IMap, IMxDocument, IRasterLayer
- Relevant OMDs: ArcMap, Map Layer

BASIC STEPS

☐ Declare a module-level variable that captures events on the (default) *outbound* interface of the Map class.

☐ Declare a module-level variable to the IMxDocument interface.

☐ Whenever a new or existing document is opened, or the focused map in the document changes, the module-level Map variable should be set to the document's focused map.

☐ Write code that will execute whenever an *item* is added to the focused map.

☐ If the item is a RasterLayer, it should be placed at the bottom of the Table of Contents in the map.

☐ Save your map. Close it and reopen it. Add several layers to the map.

☐ Add a raster layer. Is it added to the bottom of the Table of Contents?

EXERCISE END

SOLUTION B6: CODE OBJECT EVENTS

```
'**Dimension some module-level variables ...
'--Use the "WithEvents" keyword to capture the events ...
Private WithEvents m_pMapEvents As Map
Private m_pMxDoc As IMxDocument

'**When a New or existing map is opened, call a sub
'--that will initialize the required variables.
'--(also call it when the focused map changes) ...
Private Function MxDocument_NewDocument() As Boolean
  Call InitVars
End Function
Private Function MxDocument_OpenDocument() As Boolean
  Call InitVars
End Function
Private Sub pMapEvents_FocusMapChanged()
  Call InitVars
End Sub

'**This is the sub that initializes the variables ...
Private Sub InitVars()
  Set m_pMxDoc = ThisDocument
  Set m_pMapEvents = m_pMxDoc.FocusMap
End Sub

'**This event procedure will fire when a new "item" is
'--added to the focused map. An item could be any kind
'--of layer or table. Move rasters to the bottom ...
Private Sub pMapEvents_ItemAdded(ByVal Item As Variant)
  If TypeOf Item Is IRasterLayer Then
      Dim pMap As IMap
      Set pMap = m_pMapEvents
      pMap.MoveLayer Item, (pMap.LayerCount - 1)
  End If
End Sub
```

EXERCISE B7: IDENTIFY LAYER FEATURES

In this exercise, you will make a tool that works like the existing ArcMap Identify tool. For the most part, this is a fairly straightforward task. The relatively high difficulty rating comes from using objects and interfaces that are (most likely) unfamiliar to you.

- Difficulty:

- Objects used: Application, DisplayTransformation, IdentifyObj, FeatureLayer, MxDocument, Point
- Interfaces used: IArray, IDisplayTransformation, IIdentify, IIdentifyObj, IMxApplication, IMxDocument
- Relevant OMDs: ArcMap, Map Layer

BASIC STEPS

☐ Add a new UIToolControl to the ArcMap interface.

☐ Navigate to the *MouseDown* procedure.

☐ Create a new point in map units from the location of the mouse click (which is in pixels).

☐ Refer to the Map Layer object model diagram for an interface that is supported by FeatureLayer (and many others, like RasterLayer, for example ...wink, wink) that allows you to identify features. (If you get stuck, refer to the *Interfaces used* listing above.)

☐ Declare a variable as a pointer to this interface.

☐ Set the variable equal to the focused map's selected layer.

☐ Call the Identify method on the variable, passing in the point you created above. Make sure to have an appropriate variable (a type of array) declared to store the returned object.

☐ Pull the first element from the array, store it in an appropriate variable (again, see the *Interfaces used* listing above if you get stuck).

☐ Use this variable to display information about the location (feature) identified in a message box. Test your tool.

EXERCISE END

SOLUTION B7: IDENTIFY LAYER FEATURES

```
Private Sub UIToolControl1_MouseDown(ByVal button As Long, ByVal shift As Long, ByVal
x As Long, ByVal y As Long)
  Dim pMxApp As IMxApplication
  Dim pMxDoc As IMxDocument
  Set pMxApp = Application
  Set pMxDoc = ThisDocument

  Dim pDispTrans As IDisplayTransformation
  Set pDispTrans = pMxApp.Display.DisplayTransformation

  Dim pPoint As IPoint
  Set pPoint = pDispTrans.ToMapPoint(x, y)

  Dim pIdentify As IIdentify
  Set pIdentify = pMxDoc.SelectedLayer
  If pIdentify Is Nothing Then Exit Sub

  Dim pIDArray As IArray
  Set pIDArray = pIdentify.Identify(aPoint)

  'Get the first IdentifyObject
  If Not pIDArray Is Nothing Then
      Dim pIdObj As IIdentifyObj
      Set pIdObj = pIDArray.Element(0)

      'Report info from IdentifyObject
      MsgBox "Layer: " & pIdObj.Layer.Name & vbNewLine & _
             "Feature: " & pIdObj.Name
  Else
       MsgBox "No feature was identified"
  End If
End Sub
```

EXERCISE B8: EXPORT THE ACTIVE VIEW AS A JPEG

In this exercise, you will create a UIButtonControl that exports the active view (focus map or page layout) to a JPEG image on disk.

- Difficulty:

- Objects used: CancelTracker, Envelope, JpegExporter, MxDocument, Point
- Interfaces used: IEnvelope, IExporter, IMxDocument, IPoint
- Relevant OMDs: ArcMap, Geometry, Output

BASIC STEPS

☐ Start ArcMap and create a new map. Add some layers and page layout elements (north arrow, legend, scale bar, etc.).

☐ Use the *Customize* dialog box to add a new UIButtonControl to the interface.

☐ In the button's click event, write code to make a new JpegExporter.

☐ Define the pathname of the new JPEG as *C:\Student\XTest.jpg*.

☐ Define the output pixel boundary with an envelope that has its upper-left corner at 0, 0 and its lower-right corner at 600, 600.

☐ Define the output resolution as 100.

☐ Dimension a variable as Long. Assign it the value of the exporter's StartExporting method (which is an OLE_HANDLE).

☐ Call the Draw method on the MxDocument's ActiveView. For the required arguments, pass in the OLE_HANDLE above and a new CancelTracker.

☐ Call the FinishExporting method on the exporter to complete the export process.

☐ Display a message box that tells the user the active view has been successfully exported.

☐ Test the button, then preview the JPEG in ArcCatalog. Experiment with different pixel bounds and resolutions.

EXERCISE END

SOLUTION B8: EXPORT THE ACTIVE VIEW AS A JPEG

```
Private Sub UIButtonControl1_Click()
  Dim pExporter As IExporter
  Set pExporter = New JpegExporter
  pExporter.ExportFileName = "C:\Student\XTest.jpg"

  Dim pEnv As IEnvelope
  Set pEnv = New Envelope

  Dim pPoint As IPoint
  Set pPoint = New Point

  pPoint.PutCoords 0, 0
  pEnv.UpperLeft = pPoint

  pPoint.PutCoords 600, 600
  pEnv.LowerRight = pPoint

  pExporter.PixelBounds = pEnv
  pExporter.Resolution = 100

  Dim pMxDoc As IMxDocument
  Set pMxDoc = ThisDocument

  Dim pHandle As Long
  pHandle = pExporter.StartExporting()
      pMxDoc.ActiveView.Draw pHandle, New CancelTracker
  pExporter.FinishExporting

  MsgBox "Finished exporting the active view"
End Sub
```

EXERCISE B9: USE ARCOBJECTS IN ANOTHER APPLICATION

In this exercise, you will ArcObjects in one of the Microsoft Office applications. You will use the ArcGIS MapControl (which is an ActiveX control) to embed a map into a Word document or Power Point slide.

- Difficulty:
- Additional libraries: Office, PowerPoint or Word
- ArcObjects used: MapControl, Envelope
- Interfaces used: IEnvelope, IMapControl, IMapControlEvents

Relevant OMDs: ArcObjects Controls, Geometry

BASIC STEPS

☐ From the *Start* menu, launch Microsoft Word or PowerPoint.

☐ On the *View* menu, choose *Toolbars*, and then select *Control Toolbox*.

☐ On the Control Toolbox toolbar, click *More Controls*.

☐ Scroll to and check on the ESRI MapControl.

☐ Draw a rectangle on the PowerPoint slide to add a MapControl.

☐ Right-click the *Map Control* and (on the context menu) choose *ESRI MapControl Object*, then click *properties*. Use the dialog to add some layers to the control.

☐ Run the PowerPoint presentation. Your map should be displayed, but looks like a simple graphic.

☐ Return to design mode. Double-click the *Map Control* to go to the *Visual Basic Editor*. Make a reference to the ArcObjects library.

☐ Write code that allows the user to zoom in, zoom out, or pan on the map by coding the appropriate Map Control events. You will need to reference the appropriate libraries.

☐ Test your control again by zooming and panning on the display.

EXERCISE END

Solution B9: Work with ArcObjects in another application

```
Private m_pEnv As IEnvelope

Private Sub MapControl1_OnKeyUp _
           (ByVal KeyCode As Long, ByVal ShiftState As Long)
' Zoom is performed with a shift-mouse drag.
' When the shift key is released, refresh.
    If KeyCode = 16 Then 'This is the ASCII code for Shift
        MapControl1.Extent = m_pEnv
        MapControl1.Refresh
' Zoom to the full extent when "F" is pressed.
    ElseIf Chr(KeyCode) = "F" Then
        MapControl1.Extent = MapControl1.FullExtent
        MapControl1.Refresh
    End If
End Sub

Private Sub MapControl1_OnMouseDown(ByVal button As Long, _
   ByVal shift As Long, ByVal x As Long, ByVal y As Long, _
   ByVal mapX As Double, ByVal mapY As Double)
' If shift held, track an envelope for the new extent
    If shift = 1 Then
        Set m_pEnv = MapControl1.TrackRectangle
' If shift is not being held, pan on the map display
    Else
        MapControl1.Pan
    End If
End Sub
```

Exercise B10: Work with display feedback

In this exercise, you will create a tool that allows the user to make sure new points are added at least 100 meters from existing features in the map. The tool will use display feedback to show a 100-meter buffer around the mouse cursor. As an optional challenge, you can write code to add a new graphic to the display when the mouse is right-clicked.

- Difficulty:
- Objects used: Envelope, MarkerElement, MoveGeometryFeedback, Point, Polygon, RgbColor, ScreenDisplay, SimpleMarkerSymbol
- Interfaces used: IElement, IMarkerElement, IMoveGeometryFeedback, IRgbColor, IScreenDisplay, ISimpleMarkerSymbol
- Relevant OMDs: ArcMap, Display, Geometry

Basic steps

☐ Start ArcMap and create a new map. Add the Redlands Streets shapefile (from the IPAO database) as a new layer.

☐ Use the *Customize* dialog box to add a new UIToolControl. Right-click the control and choose to *View Source*.

☐ Declare (module-level) variables in the General Declarations section to point to the following interfaces: IMxApplication, IMxDocument, IGeometry, IMoveGeometryFeedback, and IScreenDisplay.

☐ In the tool's *Select event* procedure, set variables to point to the Application, the current MxDocument, the application display, and a new MoveGeometryFeedback object.

☐ Also in the tool's Select event procedure, set the Display property of the MoveGeometryFeedback object.

☐ In the tool's *MouseDown* event procedure, capture the point clicked by the user and store it as a map point.

☐ QueryInterface to the ITopologicalOperator interface on the map point.

☐ Call the Buffer method on ITopologicalOperator to return a buffer of 100 meters. Store the buffer in the module-level IGeometry variable.

☐ Add the buffer polygon to the MoveGeometryFeedback.

☐ Call the Start method on IMoveGeometryFeedback. Use the map point as the required anchor point parameter.

☐ Navigate to the *MouseMove* event procedure.

☐ Create a point in map units using the mouse click coordinates.

☐ Call the MoveTo method on IMoveGeometryFeedback, using the map coordinate as the required parameter.

☐ Navigate to the *MouseUp* procedure.

☐ Clear the geometry from the MoveGeometryFeedback.

☐ Create a new variable that points to the IEnvelope interface.

☐ Store the buffer polygon's envelope in the variable.

☐ Create a map point from the location where the user clicked.

☐ Center the envelope at the map point.

☐ Call the PartialRefresh method on the document's active view, pass in the envelope, and refresh the foreground.

☐ Close the Visual Basic Editor and test your tool. The buffer should display when the mouse button is held, and then refresh when it is released.

☐ As a challenge, add code that places a new graphic point when the mouse is right-clicked.

EXERCISE END

SOLUTION B10: WORK WITH DISPLAY FEEDBACK

```
Private m_pMxApp As IMxApplication
Private m_pMxDoc As IMxDocument
Private m_pDisplay As IScreenDisplay
Private m_pPolyFB As IMoveGeometryFeedback
Private m_pBuffer As IGeometry

Private Sub UIToolControl1_MouseDown(ByVal button As Long, _
    ByVal shift As Long, ByVal x As Long, ByVal y As Long)
  Dim pClickPoint As IPoint
  Set pClickPoint = m_pDisplay.DisplayTransformation.ToMapPoint(x, y)
  Dim pTopoOp As ITopologicalOperator
  Set pTopoOp = pClickPoint
  Set m_pBuffer = pTopoOp.Buffer(100)

  If button = 1 Then
      m_pPolyFB.AddGeometry m_pBuffer
      m_pPolyFB.Start pClickPoint
  Else
      Dim pMElem As IMarkerElement
      Dim pElem As IElement
      Set pMElem = New MarkerElement
      Set pElem = pMElem
      Dim pMSym As IMarkerSymbol
      Set pMSym = pMElem.Symbol
      Dim pColor As IRgbColor
      Set pColor = New RgbColor
      pColor.RGB = vbRed
      pMSym.Color = pColor
      pMElem.Symbol = pMSym
      pElem.Geometry = pClickPoint
      m_pMxDoc.ActiveView.GraphicsContainer.AddElement pMElem, 0
      m_pMxDoc.ActiveView.PartialRefresh esriViewGraphics, Nothing,
m_pBuffer.Envelope
  End If
End Sub

Private Sub UIToolControl1_MouseMove(ByVal button As Long, _
    ByVal shift As Long, ByVal x As Long, ByVal y As Long)
m_pPolyFB.MoveTo _
       m_pDisplay.DisplayTransformation.ToMapPoint(x, y)
End Sub

Private Sub UIToolControl1_MouseUp(ByVal button As Long, _
  ByVal shift As Long, ByVal x As Long, ByVal y As Long)
  m_pPolyFB.ClearGeometry
  Dim pEnv As IEnvelope
  Set pEnv = m_pBuffer.Envelope
  pEnv.CenterAt _
       m_pDisplay.DisplayTransformation.ToMapPoint(x, y)
  m_pMxDoc.ActiveView.PartialRefresh esriViewForeground, _
       Nothing, pEnv
End Sub

Private Sub UIToolControl1_Select()
  Set m_pMxApp = Application
  Set m_pMxDoc = ThisDocument
  Set m_pDisplay = pMxApp.Display
```

```
    Set m_pPolyFB = New MoveGeometryFeedback
    Set m_pPolyFB.Display = m_pDisplay
  End Sub
```

EXERCISE B11: DYNAMICALLY ALTER MAP CONTENTS

In this exercise, you will produce a new UIComboBox control that allows the user to quickly alter the contents of the active data frame.

- Difficulty:

- Objects used: GxLayer, FeatureLayer, Map, MxDocument
- Interfaces used: IGxFile, IGxLayer, ILayer, IMap, IMxDocument
- Relevant OMDs: ArcCatalog, ArcMap, Map Layer

BASIC STEPS

☐ Start ArcMap with a new map. Use the customize dialog to add a new UIComboBoxControl to the interface.

☐ Launch the Visual Basic Editor. In the MxDocument_OpenDocument procedure, write code to add the following strings to the control: "Europe", "North America", "South America".

☐ Create a new sub procedure called AddLayersFromFolder that takes a string argument called "FolderName".

☐ In the AddLayersFromFolder procedure, write the following code:

- Create a new GxCatalog object.
- Call the CreateObjectFromFullName method on the GxCatalog variable to return the folder (GxObject) identified by "FolderName".
- Get the IGxObjectContainer interface on the folder.
- Get all the children (GxObjects) from the folder.
- Clear all the layers from the focus map.
- Loop through each child (GxObject) in the folder. While the GxObject is not nothing, see if it supports the IGxLayer interface.
- If it is a type of GxLayer, make a new GxLayer object. Set the GxLayer pathname using the FolderName and the GxObject’s name + ".lyr" (Hint: You may need to use QI.).
- Get the Layer from the GxLayer, and add it to the focus map.
- End the if statement.
- Get the next child (GxObject).
- End the loop.
- Zoom the focus map to the full extent.
- Refresh the active view, update the map contents.

☐ Navigate to the UIComboBoxControl's SelectionChange event.

☐ Write a Select Case statement that calls the AddLayersFromFolder procedure and passes the appropriate FolderPath, as described below.

☐ Europe = "C:\Student\IPAO\Data\Europe"

☐ North America = "C:\Student\IPAO\Data\North America"

☐ South America = "C:\Student\IPAO\Data\South America"

☐ Save the map, close it, and reopen it (to make sure the OpenDocument procedure runs).

☐ Test your control. The layers in the map should dynamically change according to what is chosen in the combo box.

EXERCISE END

SOLUTION B11: DYNAMICALLY ALTER MAP CONTENTS

```
Private Function MxDocument_OpenDocument() As Boolean
  UIComboBoxControl1.AddItem "Europe"
  UIComboBoxControl1.AddItem "North America"
  UIComboBoxControl1.AddItem "South America"
End Function

Public Sub AddLayersFromFolder(FolderPath As String)
  Dim pCatalog As IGxCatalog
  Set pCatalog = New GxCatalog

  Dim pGxObjCont As IGxObjectContainer
  Set pGxObjCont =  pCatalog.GetObjectFromFullName _
                             (FolderPath, 1)

  Dim pEnumGxObj As IEnumGxObject
  Set pEnumGxObj = pGxObjCont.Children

  Dim pMxDoc As IMxDocument
  Set pMxDoc = ThisDocument
  pMxDoc.FocusMap.ClearLayers

  Dim pGxObj As IGxObject
  Dim pGxFile As IGxFile
  Dim pGxLayer As IGxLayer

  Set pGxObj = pEnumGxObj.Next
  Do Until pGxObj Is Nothing
    If TypeOf pGxObj Is IGxLayer Then
        Set pGxLayer = New GxLayer
        Set pGxFile = pGxLayer
        pGxFile.Path = FolderPath & "\" & pGxObj.Name
        pMxDoc.FocusMap.AddLayer pGxLayer.Layer
    End If
    Set pGxObj = pEnumGxObj.Next
  Loop

  pMxDoc.ActiveView.Extent = pMxDoc.ActiveView.FullExtent
  pMxDoc.ActiveView.Refresh
  pMxDoc.UpdateContents
End Sub

Private Sub UIComboBoxControl1_SelectionChange _
           (ByVal newIndex As Long)
  Dim intChoice As Integer
  intChoice = newIndex + 1  'ComboBox index begins at 0

  Select Case intChoice
  Case 1        '1 = Europe
    AddLayersFromFolder "C:\Student\IPAO\Data\Europe"
  Case 2        '2 = North America
    AddLayersFromFolder "C:\Student\IPAO\Data\North America"
  Case 3        '3 = South America
    AddLayersFromFolder "C:\Student\IPAO\Data\South America"
  Case Else
    MsgBox "Unrecognized Choice!", vbCritical
    Exit Sub
  End Select
End Sub
```

EXERCISE B12: WRITE ALL MAP LAYERS TO DISK

In this exercise, you will create a new UIButtonControl for ArcMap that writes each layer in the active data frame to its own layer file (*.lyr) on disk.

- Difficulty: █░░░

- Objects used: GxLayer, Map, MxDocument
- Interfaces used: IEnumLayer, ILayer, IGxFile, IGxLayer, IMap, IMxDocument
- Relevant OMDs: ArcCatalog, ArcMap

BASIC STEPS

☐ Start ArcMap. Use the Customize dialog to add a new UIButtonControl to a toolbar.

☐ In the button's click event procedure, write code to get all the layers from the activated data frame (focused map).

☐ Loop through the enum of layers. For each layer, create a new GxLayer. Set the GxLayer's pathname to store the layer in your student directory using the layer name + *.lyr* as the filename (C:\Student\Streets.lyr, for example). Set the Layer property to point to the layer. Call the Save method on the GxLayer to write it to disk. Finally, end your loop.

☐ When the loop has finished, tell the user that all layers have been successfully written to disk.

☐ Your control should only be enabled if there are one or more layers in the active data frame.

EXERCISE END

SOLUTION B12: WRITE ALL MAP LAYERS TO DISK

```
Private m_pMxDoc As IMxDocument

Private Sub UIButtonControl1_Click()
    Set m_pMxDoc = ThisDocument
    Dim pEnumLayers As IEnumLayer
    Set pEnumLayers = m_pMxDoc.FocusMap.Layers

    Dim pLayer As ILayer
    Set pLayer = pEnumLayers.Next
    Dim pGxLayer As IGxLayer
    Dim pGxFile As IGxFile

    Do Until pLayer Is Nothing
        Set pGxFile = New GxLayer
        Set pGxLayer = pGxFile
        pGxFile.Path = "C:\Student\" & pLayer.Name & ".lyr"
        Set pGxLayer.Layer = pLayer
        pGxFile.Save
        Set pLayer = pEnumLayers.Next
    Loop
    MsgBox "All layers have been saved"
End Sub

Private Function UIButtonControl1_Enabled() As Boolean
 Set m_pMxDoc = ThisDocument
 UIButtonControl1_Enabled = m_pMxDoc.FocusMap.LayerCount > 0
End Function
```

EXERCISE B13: WORK WITH SPATIAL REFERENCE

In this exercise, you will make a simple form that allows the user to quickly change the spatial reference (projection) for the activated data frame.

- Difficulty:
- Objects used: MxDocument, ProjectedCoordinateSystem, SpatialReferenceEnvironment
- Interfaces used: IActiveView, IMxDocument, IProjectedCoordinateSystem, ISpatialReferenceFactory
- Relevant OMDs: ArcMap, SpatialReference

BASIC STEPS

☐ Start ArcMap. Create a new map. Add the AfricanCountries shapefile from the IPAO training database.

☐ Start the Visual Basic Editor. Create a new user form.

☐ Add three option buttons to the form. The option buttons will allow the user to change the data frame's projection to: Albers, Lambert, or Sinusoidal. Name the controls and give them captions that describe their function.

☐ Open the code module for your form. Begin a new sub procedure as defined below:

```
Private ChangePrj (aNewPrj As esriSRProjCSType)
```

NOTE: esriSRProjCSType is an enumeration that identifies the predefined ArcMap projected coordinate systems.

☐ Begin the procedure by getting the IMxDocument interface on the current document.

☐ Make a new SpatialReferenceEnvironment object, pointing to the ISpatialReferenceFactory interface.

☐ Call the CreateProjectedCoordinateSystem method, passing in the aNewPrj parameter. Store the returned object in a variable that points to the IProjectedCoordinateSystem interface.

☐ Set the focused map's spatial reference with the new projected coordinate system.

☐ Zoom the active view's extent to the full extent. Refresh the active view.

☐ Add code to each of the option button's click event procedures to call the ChangePrj procedure. Pass in one of these identifiers for the appropriate projection:

```
esriSRProjCS_WGS1984AfricaAlbers, esriSRProjCS_WGS1984AfricaLambert,
esriSRProjCS_WGS1984AfricaSinusoidal.
```

☐ Run your form. Try clicking each of the option buttons. Does the projection change?

EXERCISE END

SOLUTION B13: WORK WITH SPATIAL REFERENCE

```
Public Sub ChangePrj(aNewPrj As esriSRProjCSType)
  Dim pMxDoc As IMxDocument
  Set pMxDoc = ThisDocument

  Dim pSRFactory As ISpatialReferenceFactory
  Set pSRFactory = New SpatialReferenceEnvironment

  Dim pPrjCoordSys As IProjectedCoordinateSystem
  Set pPrjCoordSys = _
         pSRFactory.CreateProjectedCoordinateSystem(aNewPrj)

  Set pMxDoc.FocusMap.SpatialReference = pPrjCoordSys
  pMxDoc.ActiveView.Extent = pMxDoc.ActiveView.FullExtent
  pMxDoc.ActiveView.Refresh
End Sub

Private Sub optAlbers_Click()
  ChangePrj esriSRProjCS_WGS1984AfricaAlbers
End Sub

Private Sub optLambert_Click()
  ChangePrj esriSRProjCS_WGS1984AfricaLambert
End Sub

Private Sub optSinusoidal_Click()
  ChangePrj esriSRProjCS_WGS1984AfricaSinusoidal
End Sub
```

EXERCISE B14: ADD AN ATTRIBUTE DOMAIN

This exercise is a modification to the code you wrote in 14B, this time adding an attribute domain. You will set some of its properties, such as the minimum and maximum value, then add it to the personal geodatabase. You will add to your existing code to associate this domain with your PopPerMH field.

☐ Return to the Visual Basic Editor in ArcCatalog. You will continue coding the AddField procedure.

☐ Move to the portion of the procedure that makes the new Field object. You will begin your code immediately *above* the line "Dim pField As IFieldEdit" (between opening the feature class and creating the new field).

☐ Dimension the following variables to reference a new RangeDomain object. Both of these variables will point to the same object (QueryInterface).

```
Dim pDomain As IDomain
Dim pRDomain As IRangeDomain
```

The Domain class is an abstract class that has two creatable subclasses: RangeDomain and CodedValueDomain. The IDomain interface, which is defined on the abstract class, can be used on either of these CoClasses. This interface defines basic properties such as Name, Description, and FieldType. The IRangeDomain interface is used to specify the MinValue and MaxValue properties.

☐ Initialize the pRDomain variable by setting it equal to a new RangeDomain object.

☐ Use QueryInterface to point the pDomain variable to the same object.

☐ Set the following properties on the RangeDomain object (on the proper interface).

Property	**Value**
Name	"PopPerMH_MinMax"
Description	"Range of reasonable values for people per mobile home"
FieldType	esriFieldTypeInteger
MinValue	1
MaxValue	1000000

The domain above will restrict values in the PopPerMH field to fall between 1 and 1 million. It is assumed that there would never be fewer than one person per mobile home and that there would never be more than one million (according to this rule, California, which has over 30 million people could not have less than 30 mobile homes in the entire state, or more than 30 million, for example).

Before assigning the domain to the PopPerMH field, it is necessary to add it to the workspace. Remember that (using the interface) domains are added as properties of the geodatabase. Once a domain is defined in a particular database, all datasets inside the database can use the domain.

☐ To add the domain to the Access database, you need to use the IWorkspaceDomains interface. Write the code below to QI to this interface.

```
Dim pWSDomains As IWorkspaceDomains
Set pWSDomains = pAccessWorkspace
```

☐ Now call the AddDomain method on pWSDomains to add pDomain to the workspace.

☐ Scroll down in the code to the portion of the procedure that sets properties for pField.

☐ Associate the domain to the field by setting pField's Domain property. (Hint: This is a property put by reference.)

☐ Using ArcCatalog, navigate to the *ManyMobileHomes* feature class. Right-click the feature class and choose *Properties*.

☐ In the Properties dialog, click the *Fields* tab. Scroll down and select the *PopPerMH* field. Press `Delete` on your keyboard to delete the field.

☐ Click *OK* in the Properties dialog.

☐ Return to the Visual Basic Editor and run the *AddField* procedure.

☐ Again, navigate to the dataset in ArcCatalog. Verify that your domain was added to the Access database and is associated with the PopPerMH field.

Congratulations. You have written a lot of code for working with datasets on disk. You wrote code to convert from one format to another, to make a subset of data, to add and calculate attribute fields, and to define attribute domains.

If you have time (or the energy), you might want to try the last challenge step. Otherwise, export your module to the student directory.

SOLUTION B14: ADD AN ATTRIBUTE DOMAIN

```
Public Sub AddField()
  Dim pAccessFactory As IWorkspaceFactory
  Dim pAccessWorkspace As IFeatureWorkspace
  Dim pAccessFClass As IFeatureClass
  Set pAccessFactory = New AccessWorkspaceFactory
  Dim pProps As IPropertySet
  Set pProps = New PropertySet
  pProps.SetProperty "DATABASE", "C:\Student\IPAO\Data\States.mdb"
  Set pAccessWorkspace = pAccessFactory.Open(pProps, Application.hWnd)
  Set pAccessFClass = pAccessWorkspace.OpenFeatureClass("ManyMobileHomes")
  Dim pRDomain As IRangeDomain
  Dim pDomain As IDomain
  Set pRDomain = New RangeDomain
  Set pDomain = pRDomain
  pDomain.FieldType = esriFieldTypeInteger
  pDomain.Name = "PopPerMH_MinMax"
  pDomain.Description = "A range of reasonable values for people per mobile home"
  pRDomain.MaxValue = 1000000
  pRDomain.MinValue = 0
  Dim pWSDomains As IWorkspaceDomains
  Set pWSDomains = pAccessWorkspace
  pWSDomains.AddDomain pDomain
'**
  Dim pField As IFieldEdit
  Set pField = New Field
  pField.Name = "PopPerMH"
  pField.Type = esriFieldTypeInteger
  pField.Length = 16
  pField.AliasName = "People Per Mobile Home"
  Set pField.Domain = pDomain '****Adding the domain
  pAccessFClass.AddField pField

  Dim pUpdateFeatures As IFeatureCursor
  Set pUpdateFeatures = pAccessFClass.Update(Nothing, True)
  Dim intNewFld As Integer
  Dim intPopFld As Integer
  Dim intHomeFld As Integer

  intNewFld = pUpdateFeatures.FindField("PopPerMH")
  intPopFld = pUpdateFeatures.FindField("Pop1999")
  intHomeFld = pUpdateFeatures.FindField("Mobilehome")

  Dim pFeature As IFeature
  Dim lngPop As Long
  Dim lngHomes As Long

  Set pFeature = pUpdateFeatures.NextFeature
  Do Until pFeature Is Nothing
      lngPop = pFeature.Value(intPopFld)
      lngHomes = pFeature.Value(intHomeFld)
      pFeature.Value(intNewFld) = lngPop / lngHomes
      pUpdateFeatures.UpdateFeature pFeature

      Set pFeature = pUpdateFeatures.NextFeature
  Loop
  MsgBox "Update complete"
End Sub
```

I N D E X