Building ActiveX Controls

Objectives

- Learn the steps in creating a simple ActiveX control.
- Investigate the events that occur in a UserControl object's life cycle.
- Use the ActiveX Control Interface Wizard to create most of a control's code.

TIP: All code examples for this chapter can be found in the text file ACTIVEX.TXT. A finished version of the control can be found in the sample project PRGMETER.VBP, and you'll find a test project as well (TESTPM.VBP).

What Is an ActiveX Control, Anyway?

If you've developed applications in almost any Microsoft development environment, you've used ActiveX controls. These user-interface elements (previously called **OLE controls)** provide extensions to the native environment. Some pertinent factoids:

- ActiveX controls can only be instantiated in a host application.
- An ActiveX control is basically an in-process Automation server, with an optional user interface.
- A single OCX file (the disk file that contains ActiveX controls) can contain many controls.
- If you're intending to provide functionality without a user interface, you may be better off using an ActiveX DLL. There's a price to pay for the interface elements of an ActiveX control (first of all, they require a form to be loaded in order to "come to life").

The remainder of this chapter walks you through the process of creating and using a simple ActiveX control: a continuous progress meter. Figure 1 shows the control in design view, and Figure 2 shows the control in use on a form.

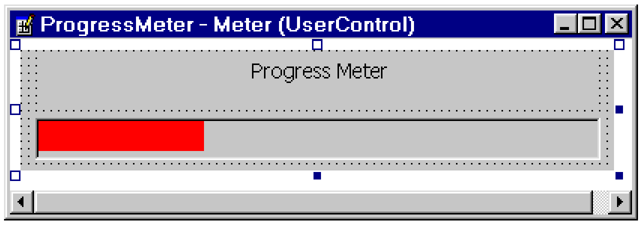


Figure 1. The Progress Meter control in design view.

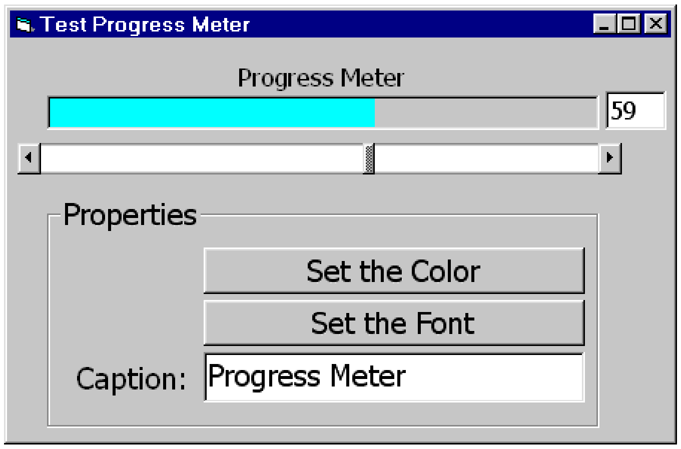


Figure 2. The sample ProgressMeter test form.

TIP: As we work through this chapter, we'll build the entire control from scratch. You needn't work through the chapter, however – the finished control is included on your sample diskette, as the project **PRGMETER.VBP,** and the **TESTPM.VBG** project group file to load both the control and the test project.

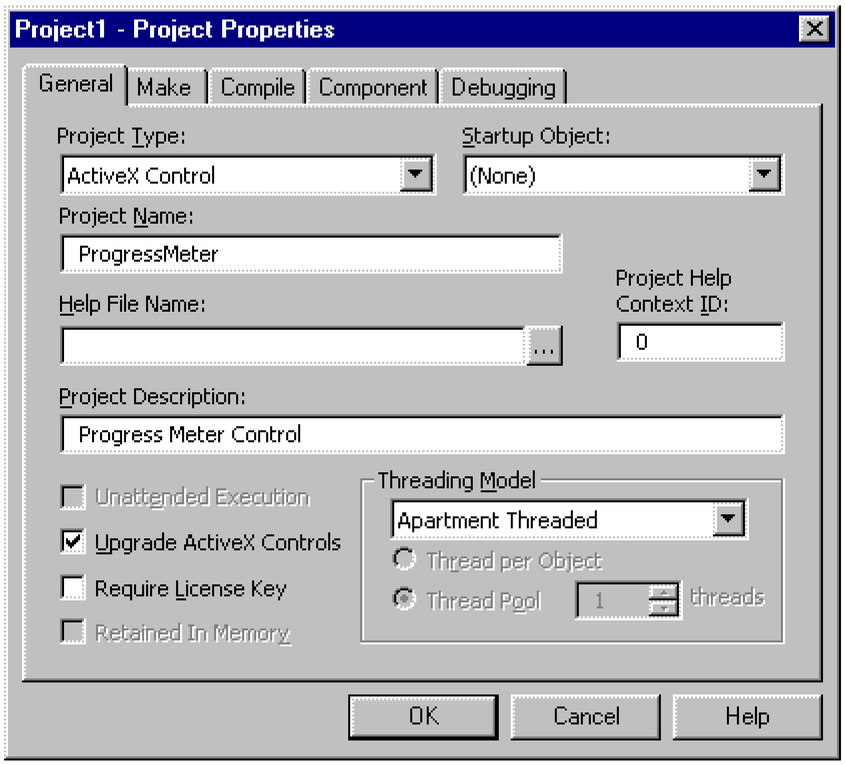
NOTE Unlike most other chapters, this one consists entirely of a set of steps, with explanation. If you can, you'll benefit the most from working through these steps, either during class or back at your home/office.

Creating the ProgressMeter Project

Although you can include an ActiveX control in any type of project, it'll only be available outside the project, as an ActiveX control that you can insert into other projects, if you create the project as an **ActiveX Control** project, and compile it to an OCX file.

1. **Create a new ActiveX Control project.** At this point, Visual Basic creates a UserControl object, presented to you in the UserControl designer, named UserControl1.

2. **Choose the Project|Project1 Properties**, and fill in the properties as shown below, in Figure 3, and then dismiss the dialog box:



Project Type: ActiveX Control

Startup Object: (None)

Project Name: ProgressMeter

Help File Name:

Project Help Context ID: 0

Project Description: Progress Meter Control

☐ Unattended Execution

☑ Upgrade ActiveX Controls

☐ Require License Key

☐ Retained In Memory

Threading Model: Apartment Threaded

○ Thread per Object

● Thread Pool 1 threads

Figure 3. Set the project properties before doing anything else.

What do these properties do?

- The **Project Name** property specifies the name of the type library for the compiled .OCX file.
- The **Project Description** property specifies the name users will see when they display the list of installed ActiveX components.

3. In the Properties sheet for the new UserControl object, set the **Name** property to be **Meter**. This becomes the class name for the control (just like CommandButton, or CommonDialog).

4. **Resize** the control designer so it's rectangular in shape. Thi sets the default size of the control, and we'll need a wide, short design area.

5. **Save the project:** from the File menu, choose the Save Project item. Save the control as Meter (Visual Basic adds the .CTL extension) and the project as PrgMeter (with the .VBP extension).

Adding the TestMeter Project

In order to test your control, you'll need a form. To make it simple for you to test ActiveX controls as they're being built, Visual Basic allows you to add a second project to your workspace. You can add a form in this second project, and run the form in both design and run mode with the original ActiveX control project in run mode.

You can save the two projects together as a project group (.VBG) file. This section shows how to add the second project, and how to save the project group file.

1. **Add the new project**: From the File menu, choose the **Add Project** menu item, and choose the **Standard EXE** project type. The form that's part of the new project (Form1) will become your test form. You should now be able to see both projects in the Project window, as shown in Figure 4.

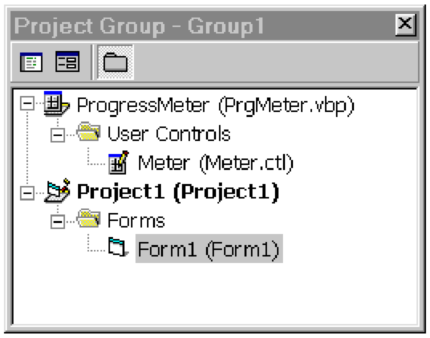


Figure 4. Once you add a second project, the new project becomes the startup project.

2. **Save the project group**: From the File menu, choose the Save Project Group menu item. Save the form as TestPM.FRM, the project as TestPM.VBP, and the group file as TestPM.VBG.

Running the ProgressMeter at Design Time

Unlike any other Visual Basic component type, ActiveX controls have three stages in their "life cycle" (most other components have only two: design-time and run-time):

- **Design-time for the control**—at this stage, you're setting the properties and behaviors the control will exhibit when placed into a host.
- **Run-time for the control/Design-time for the host**—at this stage, the control's project is running. It can react to events, but the host form is in design view.
- **Run-time for the control/Run-time for the host**—at this stage, the control is running, and the host is running. Both the control and the host can react to events.

Between each stage, Visual Basic destroys and recreates the control, and (this is the important part) triggers various events, depending on the stage of the control's "life."

This section will demonstrate how the control project can be running (that is, executing code) while the host project can be in design mode.

1. In the Project window, **double-click the Meter control** to make it the active designer.
2. Add a **Timer** control to the design surface, and set the control's Interval property to be 1000.
3. Add a **Label** control to the design surface, and make it large enough to read a short text string.
4. **Double-click** the Timer control, to add some code.
5. **Add code** so that the procedure looks like this:

```
' Code example #1.
Private Sub Timer1_Timer()
    Label1.Caption = Time
End Sub
```

6. **Close all the designer windows**. Doing this causes Visual Basic to run the ActiveX control's project, and its icon should now be available in the

Toolbox. Figure 5 shows the tool tip and the control available in the Toolbox.

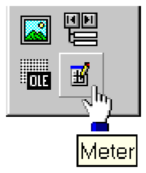


Figure 5. Once you close all the designer windows, your control is available in the Toolbox.

7. **Open Form1 in design view** and place an instance of the Meter control on the form. If you wait a second, you'll notice that the new control's label displays the current time. Figure 6 shows the control running on the form that's in Design view.

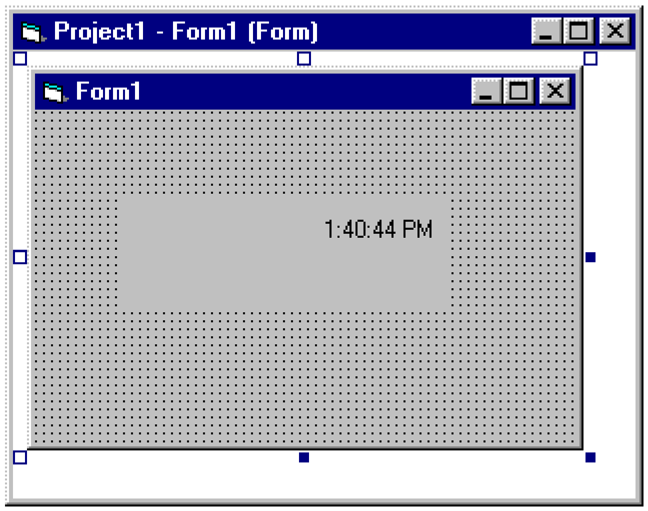


Figure 6. Even though the form isn't running, the code in the new ActiveX control is.

8. To see what happens when the control is open in design view, **double click on the Meter control in the Project window**. This will open the designer for the control, and will disable the control in the Toolbox and on the sample form. Figure 7 shows this situation.

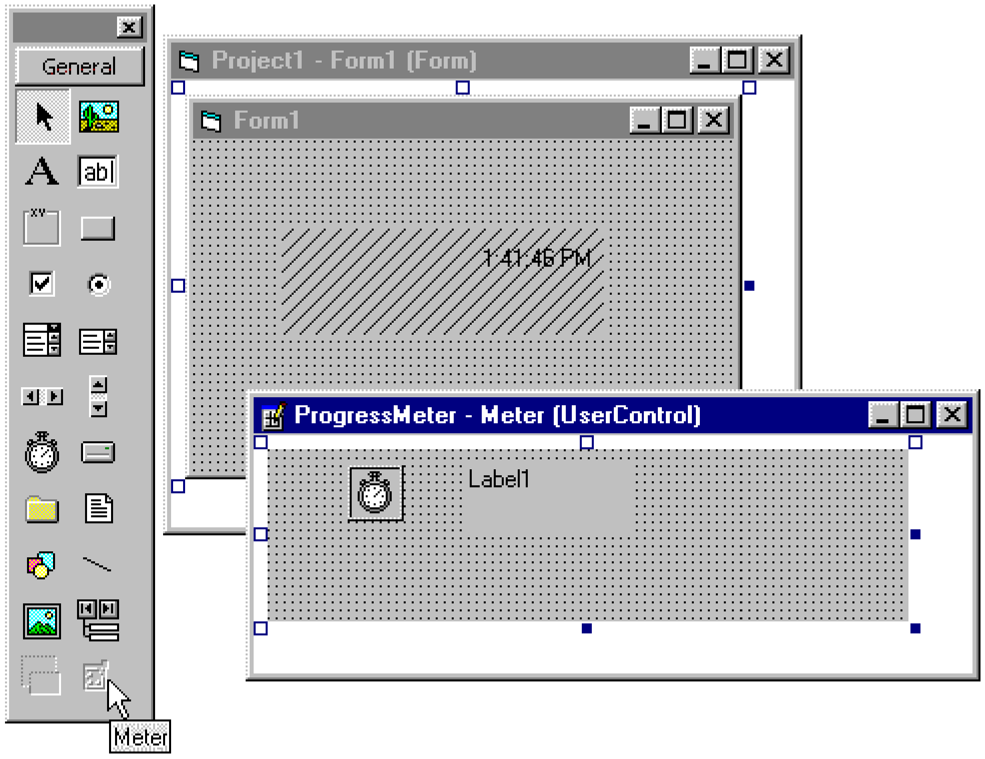


Figure 7. Once the control's open in design view, it's not available on the Toolbox, nor on the sample form.

9. When the control's open in design view, it doesn't react to events. This is easy to prove: while the control is "hatched out", the clock isn't ticking. The moment you close the control's design surfaces, the hatching goes away, and the ticking resumes.

10. **Delete the Meter control from Form1**, in preparation for the next step.

11. **Remove the Timer and Label** controls from the Meter control**.** We won't need them anymore. (You can delete the event procedure for the Timer control, as well.)

Life and Times of a ProgressMeter Control

Like forms, the life cycles of UserControl objects hang on a framework of a series of events. These events occur as you change the state of the control, and include the following events:

- Initialize, Terminate
- InitProperties
- ReadProperties
- WriteProperties

This section investigates these properties and when they occur.

1. Add the following code to the Meter control's module:

```
' Code example #2.
Private Sub UserControl_Initialize()
    Debug.Print "Initialize"
End Sub

Private Sub UserControl_InitProperties()
    Debug.Print "InitProperties"
End Sub

Private Sub UserControl_ReadProperties( _
 PropBag As PropertyBag)
    Debug.Print "ReadProperties"
End Sub

Private Sub UserControl_Terminate()
    Debug.Print "Terminate"
End Sub
```

```
Private Sub UserControl_WriteProperties( _
 PropBag As PropertyBag)
    Debug.Print "WriteProperties"
End Sub
```

2. **Close all the windows** associated with the Meter control.

3. With the Immediate window open**, insert a Meter control on the form.** As you do this, you'll see three events occur:

   - **Initialize**: The Initialize event always occurs when you instantiate a new object.
   - **InitProperties:** The InitProperties event occurs when you first place a control on a form, allowing your code to set initial values of properties.

4. **Press F5 to run the project**. Now you'll see more events being registered in the Immediate window:

   - **WriteProperties**: This event occurs before Visual Basic destroys the design-time version of the running control. It allows you to save any changes made to properties at design-time (normally, changes made through the Properties sheet). These changes are stored with the form itself.
   - **Terminate**: Occurs as the control is being destroyed.

Then, as the control comes "back to life," you'll see three more events:

   - **Initialize**: Again, a new instance of the control has been created.
   - **ReadProperties**: This event occurs as the control instance is being created, giving your code a change to read the saved properties from the form's storage.

Why no InitProperties? That event only occurs when you first insert the control onto a form, when the form's in design view.

5. Close Form1. You'll see four more events occur:

   - **Terminate**: as the run-time instance of the control is destroyed
   - **Initialize**: as the new design-time instance of the control is created
   - **ReadProperties**: giving your code the chance to re-read the persisted properties

Why no WriteProperties? That only occurs when you destroy the design-time instance of the control.

6. **Remove all the event procedure code** from the UserControl's module.

Drawing the ProgressMeter

Finally, it's time to make the ProgressMeter control actually do something! In this section, we add constituent controls to the UserControl designer, and write code to react to the UserControl's Resize event.

1. In the Project window, **double click on the Meter control** to open its designer.

2. From the Toolbox, **add a label control** to the designer, and set its properties as follows:

Property	Value
Name	lblCaption
Alignment	2 - Center
Left	0
Top	0
Width	(About the same as the width of the UserControl)
Height	(About ½ the height of the UserControl)

3. From the Toolbox, **add a PictureBox control**, and set its properties as follows:

Property	Value
Name	picMeter
Align	2 – Align Bottom
ClipControls	False
Height	(About ½ the height of the UserControl)

4. From the Toolbox, **add a Shape control inside the PictureBox control**. Set its properties as follows:

Property	Value
Name	shpMeter
Left	0
Top	0
BorderStyle	0 – Transparent
FillColor	(Any color you like)
FillStyle	0 – Solid
Shape	0 – Rectangle

5. Close all the designer windows for the UserControl, and open Form1, if it’s not already open. Note that the changes you made to the control appear on Form1. **Resize the control**, and note that although the PictureBox control hugs the bottom of the UserControl, nothing else works right.

6. To make the Resize event do something**, add the following code**:

```
' Code example #3
Const conMessageHeight = 0.5

Private Sub UserControl_Resize()
    ' Set the width of the label control.
    ' Set the height to the chosen ratio of the
    ' control's height.
    lblCaption.Move 0, 0, _
     UserControl.ScaleWidth, _
     UserControl.ScaleHeight * conMessageHeight

    picMeter.Move 0, lblCaption.Height, _
     lblCaption.Width, _
     UserControl.ScaleHeight * (1 - conMessageHeight)

    shpMeter.Move 0, 0, shpMeter.Width, picMeter.Height
End Sub
```

7. Close all the UserControl’s designer windows, and **try resizing the control** on Form1. Now it should react reasonably.

Working with the ProgressMeter's Properties

So far we've managed to conglomerate some controls and have them create a new control type, but you can't use the new control to do anything, because it provides almost no properties.

By the time it's complete, this control will expose the following properties:

- **BackColor**: a standard color object, allowing you to set the color of the caption's background.
- **Caption**: a string, corresponding to the Caption property of the lblMessage control's Caption property.
- **Percent**: an integer, controls the amount of the progress meter that displays. It doesn't correspond to any built-in property.
- **FillColor**: a standard color object, allowing you to set the color of the progress meter. Corresponds to the FillColor property of the shpMeter control.
- **Font**: a Font object, including the standard font information, corresponding to the lblMessage control's Font.

In this section, we'll create the Caption property by hand, and will let the ActiveX Control Interface Wizard create the rest property.

> TIP: It's important that you understand how to create properties by hand, and to know what the issues are. But, in reality, you'll use the ActiveX Control Interface Wizard to write most of the code for you.

1. **Create the Caption property**: Double-click the UserControl's designer to load the code module. Add the following procedures:

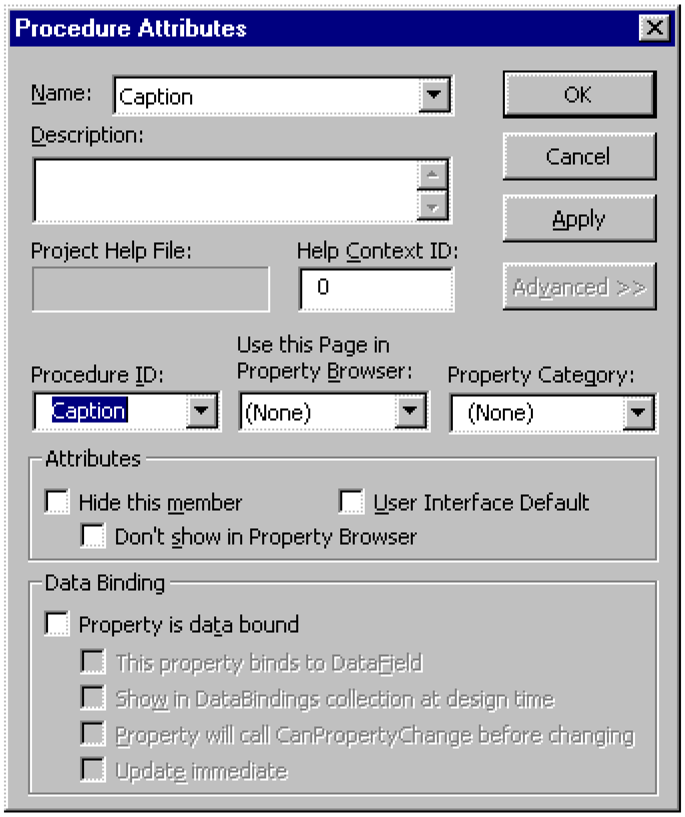
```
' Code example #4
Public Property Get Caption() As String
    Caption = lblMessage.Caption
End Property
```

```
Public Property Let Caption(ByVal NewCaption As String)
    lblMessage.Caption = NewCaption
    PropertyChanged "Caption"
End Property
```

Just as with other ActiveX components, a UserControl's module is a class module, and you use Property Get and Let procedures to retrieve and set property values. The only difference here is the use of the **PropertyChanged method**. This new statement tells the Properties window to update its display, and informs Visual Basic that the control's properties storage have changed.

2. Perhaps you've noticed that the Caption property for controls in Visual Basic has a special characteristic: As you type, the control reflects the text you're typing. To make that happen, open the **Tools|Procedure Attributes** dialog box for the UserControl object. The Caption procedure should already be selected in the **Name:** list.

3. Click the **Advanced** button, and, in the **Procedure ID:** list, select the **Caption** item. This choice indicates to Visual Basic that this procedure represents the Caption property for this object, and so it can add the appropriate Property window behavior. Click OK when you're done. Figure 8 shows the dialog box, once you've completed your mission.


Procedure Attributes

Name: Caption
Description:
Project Help File:
Help Context ID: 0
Procedure ID: Caption
Use this Page in Property Browser: (None)
Property Category: (None)

Attributes
- [ ] Hide this member
- [ ] User Interface Default
- [ ] Don't show in Property Browser

Data Binding
- [ ] Property is data bound
- [ ] This property binds to DataField
- [ ] Show in DataBindings collection at design time
- [ ] Property will call CanPropertyChange before changing
- [ ] Update immediate

Figure 8. Choose the Caption procedure ID value for the Caption property.

Persisting Property Values

1. In the Projects window, double-click on Form1, and if there's no control on the form, insert a Meter control on the form. Note that its Properties sheet includes the property you just created. **Change the value of the Caption property**. Now run the form. Note that your change wasn't preserved!

2. **Persist changed properties**: As you saw in the previous step, simply changing property values when your form is open in Design view doesn't save those properties. Because Visual Basic recreates the control every time it changes state, you need to persist the properties in the form's storage. This property information is stored, in memory, in the form's PropertyBag object. In your control's WriteProperties event, you'll need to save property values into the PropertyBag object. Remove any existing WriteProperties event code, and add the following code to your UserControl's module:

```
' Code example #5
Private Sub UserControl_WriteProperties( _
 PropBag As PropertyBag)
    'Write property values to storage
    Call PropBag.WriteProperty( _
     "Caption", lblMessage.Caption, "")
End Sub
```

The WriteProperty method of the PropertyBag object passed into the event procedure allows you to specify:

- **Property name**
- **Property value**
- **Default value** – why supply a default value for writing? If the value you're writing is the same as the default value you specify, Visual Basic simply skips writing the property value. There's no point saving property values that are the same as the default for the property, and this saves adding properties to the PropertyBag (and reducing the size and load time of the form).

3. **Load persisted properties**: Of course, when your control gets loaded back up, at run time for its host, you need to retrieve the saved properties from the PropertyBag and assign them to the control. This happens when the ReadProperties event occurs. Replace any existing code for the ReadProperties event with the following code:

```
' Code example #6
 Private Sub UserControl_ReadProperties( _
 PropBag As PropertyBag)
    lblCaption.Caption = PropBag.ReadProperty( _
     "Caption", lblCaption.Caption)
End Sub
```

Just like the WriteProperty method of the PropertyBag object, the ReadProperty method allows you to specify the property name. It also allows you to specify a default value to be used if there's no value specified in the PropertyBag.

Initializing Property Values

To initialize your control's properties, first remove the existing UserControl_InitProperties procedure (if it's still there), and add the following code to the UserControl's module. This code initializes properties when you insert an instance of the control onto its host, and runs only once for each instance:

```
' Code example #7
Private Sub UserControl_InitProperties()
    On Error Resume Next
    Me.Caption = Extender.Name
End Sub
```

Wait! What's that Extender object thing?

The Extender Object

To the user of the control, built-in properties such as Name, Top, Left and Visible appear to be part of your control. In fact, they're provided by the container provided by the host application, **the Extender object**. Visual Basic provides an Extender object that contains your control, as does any other application that supports ActiveX controls. (These are the properties you see in the Visual Basic Properties sheet when you create and insert an ActiveX control that supplies no properties of its own.)

WARNING! **Not all Extender objects are created equal**. That is, although you can be reasonably safe in assuming that all Extender objects provide a Name property, beyond that you'll need to be careful. If you use an Extender property in your control but the host doesn't provide the property, you're guaranteed to crash and burn. Of course, because your control runs in-process, there goes your host application as well. The moral of the story: use Extender properties carefully, and only with error handling. In addition, **not all hosts are created equally**. Some (such as Access 97 and IE) don't even make the Extender object available. Make sure and trap for errors!

The Ambient Object

The **Ambient object** corresponds to the form hosting your UserControl. It doesn't supply properties for your control, it simply hosts it. You can use the Ambient object to discern the properties of the form containing your control, and can write code to assimilate those properties. In this example, we'll later use the BackColor and Font properties of the host form when you insert a new control onto the form.

TIP: If you want to react to changes to Ambient properties (that is, suppose someone changes the form's BackColor property, and you want your control to always inherit that property), you can write code in the UserControl's **AmbientChanged** event procedure. This event procedure passes to you the name of the property that was changed. Generally, you'll write a Select Case statement, based on the name of the changed Ambient property, to correspondingly alter a property of your UserControl.

Why Not Use the Initialize Event?

Why was this particular code in the InitProperties event procedure, and not in the Initialize event procedure? Remember, the Initialize event occurs quite often. The InitProperties event occurs just once, and is the correct place to put code you only want executed once.

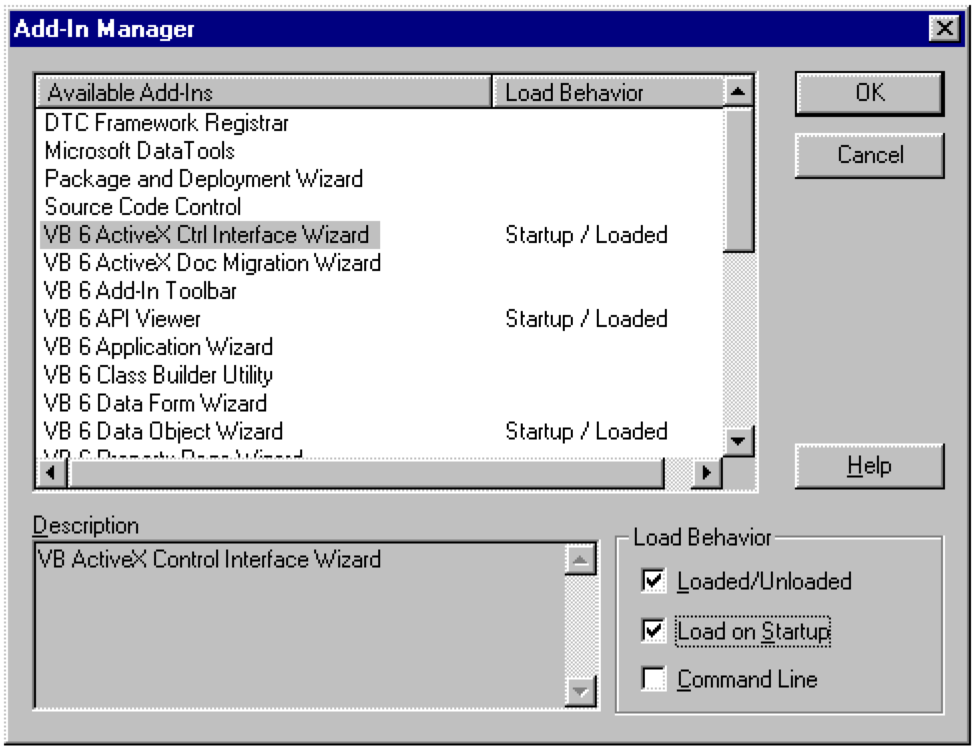
In addition, when the Initialize event occurs, the Extender and Ambient objects aren't yet available – that is, the control hasn't yet been "sited." You can't use the Extender and Ambient objects until the control has actually been placed into the host (that is, "sited" on the host), and so you must wait for the InitProperties event.

Using the ActiveX Control Interface Wizard

We still need to create BackColor, Percent, FillColor, and Font properties. Although these properties are no more difficult than the one you've already added, we'll use the ActiveX Control Interface Wizard to add these properties, just as you might use the wizard to add any number of properties to your UserControl, once you've laid out its user interface.

To use the ActiveX Control Interface Wizard to add one or properties to your UserControl, follow these steps:

1. **Ensure that the wizard is available**: pull down the Add-Ins menu. If you don't see "ActiveX Control Interface Wizard…" on the bottom of the menu, choose "Add-In Manager…" and select the wizard from the list. Figure 9 shows the Add-In Manager.



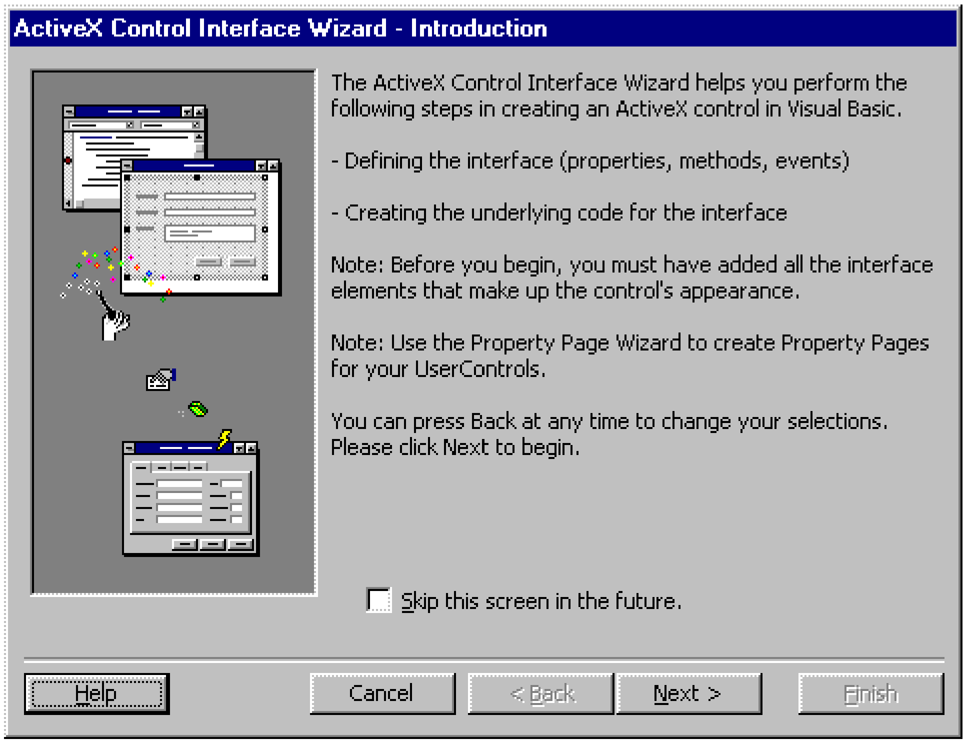
Available Add-Ins	Load Behavior
DTC Framework Registrar	
Microsoft DataTools	
Package and Deployment Wizard	
Source Code Control	
VB 6 ActiveX Ctrl Interface Wizard	Startup / Loaded
VB 6 ActiveX Doc Migration Wizard	
VB 6 Add-In Toolbar	
VB 6 API Viewer	Startup / Loaded
VB 6 Application Wizard	
VB 6 Class Builder Utility	
VB 6 Data Form Wizard	
VB 6 Data Object Wizard	Startup / Loaded

Description: VB ActiveX Control Interface Wizard

Load Behavior
- [✓] Loaded/Unloaded
- [✓] Load on Startup
- [ ] Command Line

Figure 9. The Add-In manager allows you to add wizards for the current session, and for future sessions as well.

2. **Start the Wizard**: you'll see the introductory page (unless you've told it not to show you this page, previously). Figure 10 shows the first page of the wizard.


ActiveX Control Interface Wizard - Introduction

The ActiveX Control Interface Wizard helps you perform the following steps in creating an ActiveX control in Visual Basic.

- Defining the interface (properties, methods, events)

- Creating the underlying code for the interface

Note: Before you begin, you must have added all the interface elements that make up the control's appearance.

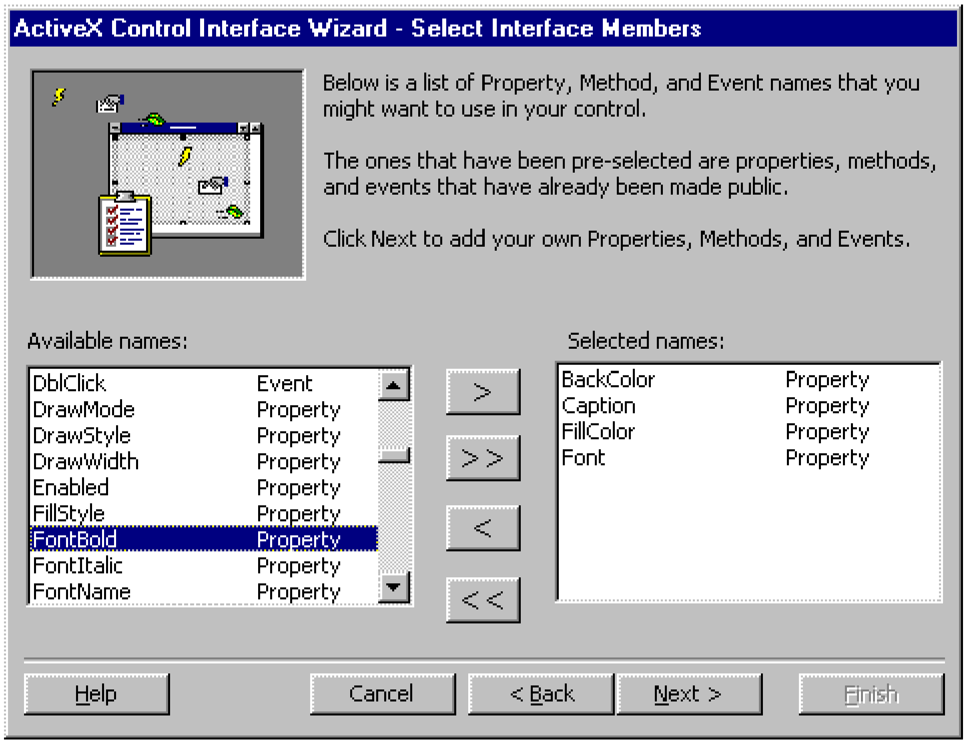
Note: Use the Property Page Wizard to create Property Pages for your UserControls.

You can press Back at any time to change your selections. Please click Next to begin.

☐ Skip this screen in the future.

Figure 10. Introduction for the ActiveX Control Interface Wizard.

3. **Select Interface Members**: On the next page, select the BackColor, FillColor, and Font properties from the list of available properties. Figure 11 shows the wizard after selecting all the necessary properties from the list.



ActiveX Control Interface Wizard - Select Interface Members

Below is a list of Property, Method, and Event names that you might want to use in your control.

The ones that have been pre-selected are properties, methods, and events that have already been made public.

Click Next to add your own Properties, Methods, and Events.

Available names:

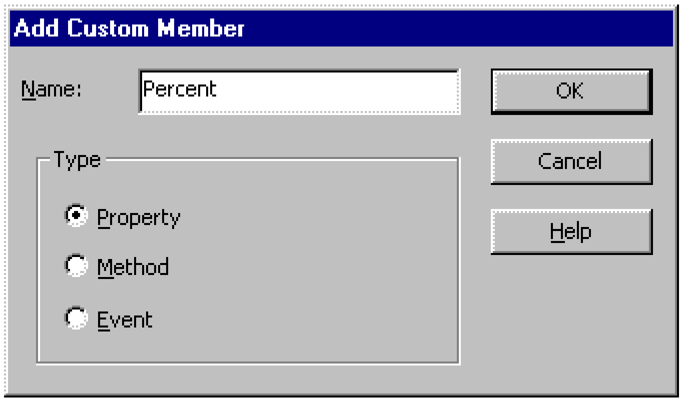
DblClick	Event
DrawMode	Property
DrawStyle	Property
DrawWidth	Property
Enabled	Property
FillStyle	Property
FontBold	Property
FontItalic	Property
FontName	Property

Selected names:

BackColor	Property
Caption	Property
FillColor	Property
Font	Property

Figure 11. Select new properties from the list of built-in properties. (Caption was there already – you choose the rest).

4. **Add custom items**: On the next page, you can add custom properties, methods, or events. In this case, add a new property named **Percent**.. Make sure you indicate that it's a property, as shown in Figure 12.



Add Custom Member

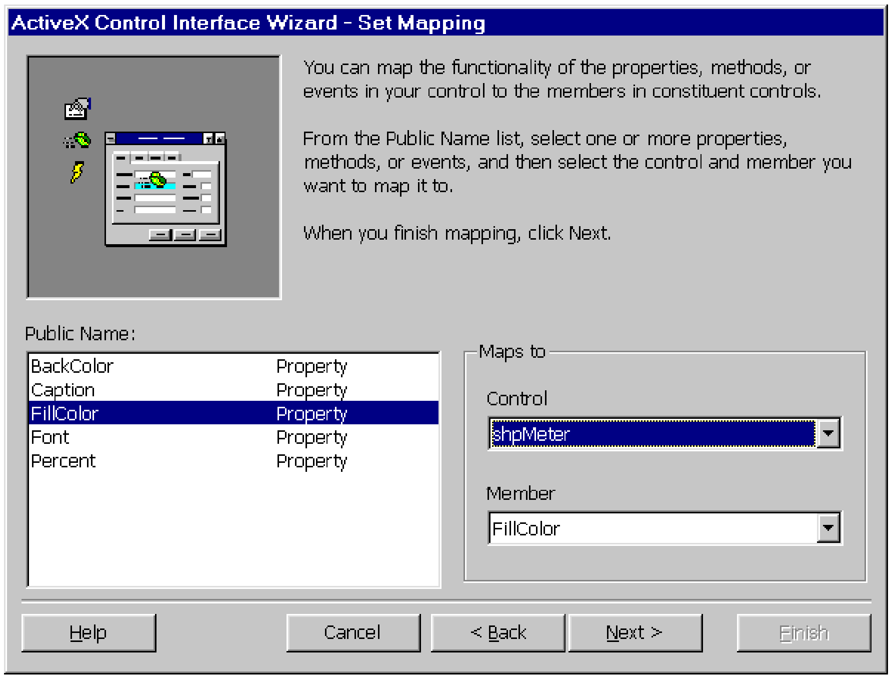
Name: Percent

Type: [✓ Property] Method Event

Figure 12. Add a new element with this dialog box.

5. **Set property mappings**: The next page allows you to map your control's properties to existing control properties. Figure 13 shows the wizard after mapping the new FillColor property to the FillColor property of shpMeter. Select mappings as in the following table:

Public Name	Maps to Control	Member
BackColor	lblCaption	BackColor
Caption	lblCaption	Caption
FillColor	shpMeter	FillColor
Font	lblCaption	Font
Percent	(None)	


ActiveX Control Interface Wizard - Set Mapping

You can map the functionality of the properties, methods, or events in your control to the members in constituent controls.

From the Public Name list, select one or more properties, methods, or events, and then select the control and member you want to map it to.

When you finish mapping, click Next.

Public Name:

BackColor	Property
Caption	Property
FillColor	Property
Font	Property
Percent	Property

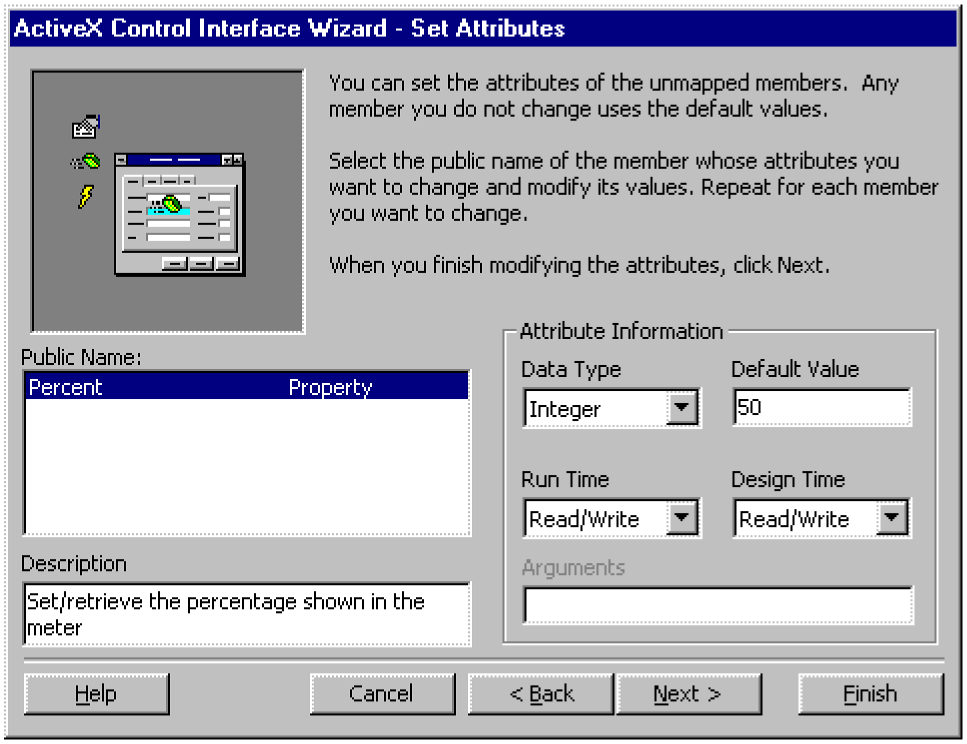
Maps to

Control: shpMeter

Member: FillColor

Figure 13. Use this page to map new properties to existing properties of constituent controls.

6. **Set attributes**: This page allows you set attributes for properties that aren’t directly mapped to built-in properties in this session. In this example, you need to set properties of the Percent property as shown in Figure 14.



You can set the attributes of the unmapped members. Any member you do not change uses the default values.

Select the public name of the member whose attributes you want to change and modify its values. Repeat for each member you want to change.

When you finish modifying the attributes, click Next.

Public Name: Percent Property

Description: Set/retrieve the percentage shown in the meter

Data Type: Integer

Default Value: 50

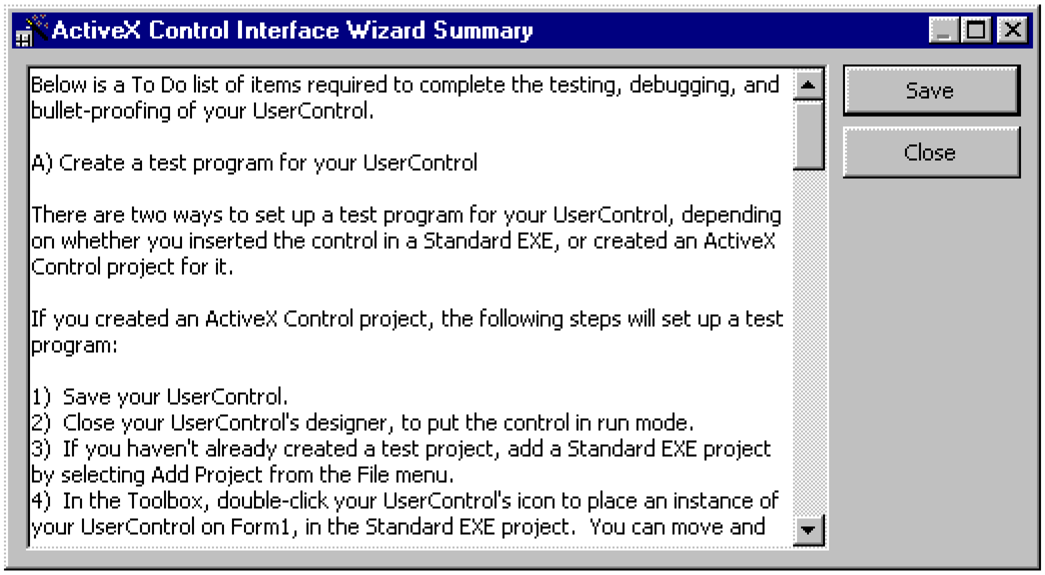
Run Time: Read/Write

Design Time: Read/Write

Arguments

Figure 14. Set attributes of unmapped properties.

7. **Finish up**: Once you've made all your changes, Visual Basic writes the modified code to your module. Before you get a chance to dig in, the wizard displays information about what you should do next. It's tempting to disregard this page, but don't. Figure 15 shows the form containing the suggestions.



Below is a To Do list of items required to complete the testing, debugging, and bullet-proofing of your UserControl.

A) Create a test program for your UserControl

There are two ways to set up a test program for your UserControl, depending on whether you inserted the control in a Standard EXE, or created an ActiveX Control project for it.

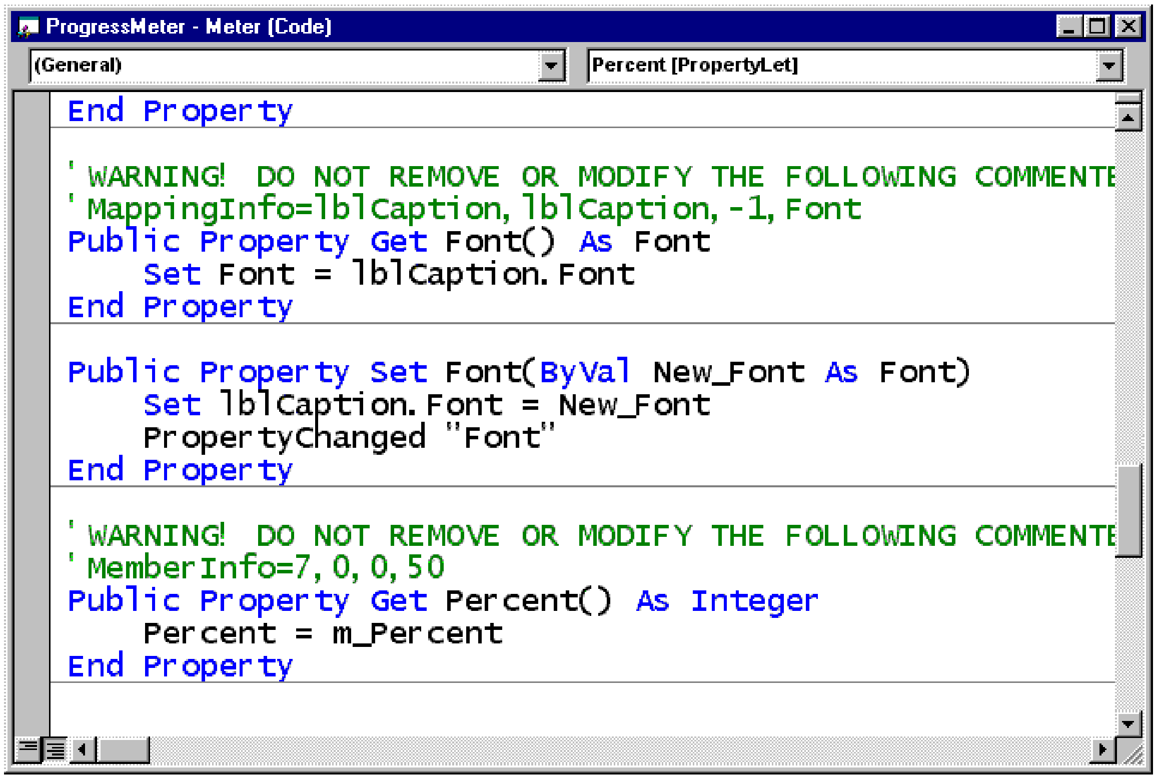
If you created an ActiveX Control project, the following steps will set up a test program:

1) Save your UserControl.
2) Close your UserControl's designer, to put the control in run mode.
3) If you haven't already created a test project, add a Standard EXE project by selecting Add Project from the File menu.
4) In the Toolbox, double-click your UserControl's icon to place an instance of your UserControl on Form1, in the Standard EXE project. You can move and

Figure 15. The Wizard makes some good suggestions. Follow them!

8. **Check out the code changes**: The wizard will add the appropriate Property Let and Property Get procedures, and will update the code in your ReadProperties and WriteProperties events. Figure 16 shows the Property Set and Get procedures for the new Font property. Note the comments the wizard added – it uses those the next time it runs, so it knows how to map the properties to built-in properties.

WARNING! If you look carefully, you'll also notice that the Wizard commented out the code you'd already added for the Caption property. Because you set the mapping in the Wizard, it assumed you no longer wanted to use the existing code. In this case, that's not a problem, but be aware that the wizard will touch your existing code.



```
End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTE
'MappingInfo=lblCaption,lblCaption,-1,Font
Public Property Get Font() As Font
    Set Font = lblCaption.Font
End Property

Public Property Set Font(ByVal New_Font As Font)
    Set lblCaption.Font = New_Font
    PropertyChanged "Font"
End Property

'WARNING! DO NOT REMOVE OR MODIFY THE FOLLOWING COMMENTE
'MemberInfo=7,0,0,50
Public Property Get Percent() As Integer
    Percent = m_Percent
End Property
```

Figure 16. The Wizard adds the appropriate code for you.

9. Test out the changes: Put away all the UserControl designers, and check out the FillColor property on Form1. You should be able to set it in design view, and have it persisted at run time.

NOTE The **OLE_COLOR data type** is a special data type, used for choosing colors. It indicates to Visual Basic that it should display a color-picker in the Properties sheet. All color properties should use OLE_COLOR as their data type. The same sort of behavior exists for the Font property type, and several others.

Finish the Control

In order to add necessary final touches to the control, you'll need to take a few more steps.

1. Add code to the InitProperties event procedure, pulling properties in from the Extender and Ambient objects (add to the code the Wizard put in the event procedure, replacing the code you had put there originally):

```
' Code example #8
Private Sub UserControl_InitProperties()
    On Error Resume Next
    m_Percent = m_def_Percent
    Me.Caption = Extender.Caption
    Set Me.Font = Ambient.Font
    Me.BackColor = Ambient.BackColor
End Sub
```

2. Add a procedure to handle setting the width of the progress meter indicator:

```
' Code example #9
Private Sub SetPercent()
    shpMeter.Width = picMeter.Width * Me.Percent / 100
End Sub
```

3. Add some unobtrusive error handling to the Percent Property Let procedure. If the value entered by a user is too high or too low, set it to a reasonable value. To do that, modify the procedure so it looks like the code below. Also, add a call to the SetPercent procedure, so the display matches the property value:

```
' Code example #10
Public Property Let Percent(ByVal New_Percent As Integer)
    If NewPercent < 0 Then NewPercent = 0
    If NewPercent > 100 Then NewPercent = 100
    m_Percent = New_Percent
    Call SetPercent
    PropertyChanged "Percent"
End Property
```

4. Add a call to SetPercent from the ReadProperties event procedure, so the meter gets updated when you read properties from storage.

Giving the ProgressMeter a Property Page

Not all host applications will provide a Property window for your control. In addition, it’s become expected for every ActiveX control to supply its own set or Property pages (and they’ll be required if the host doesn’t supply a Properties sheet that can include properties you’ve created for your control).

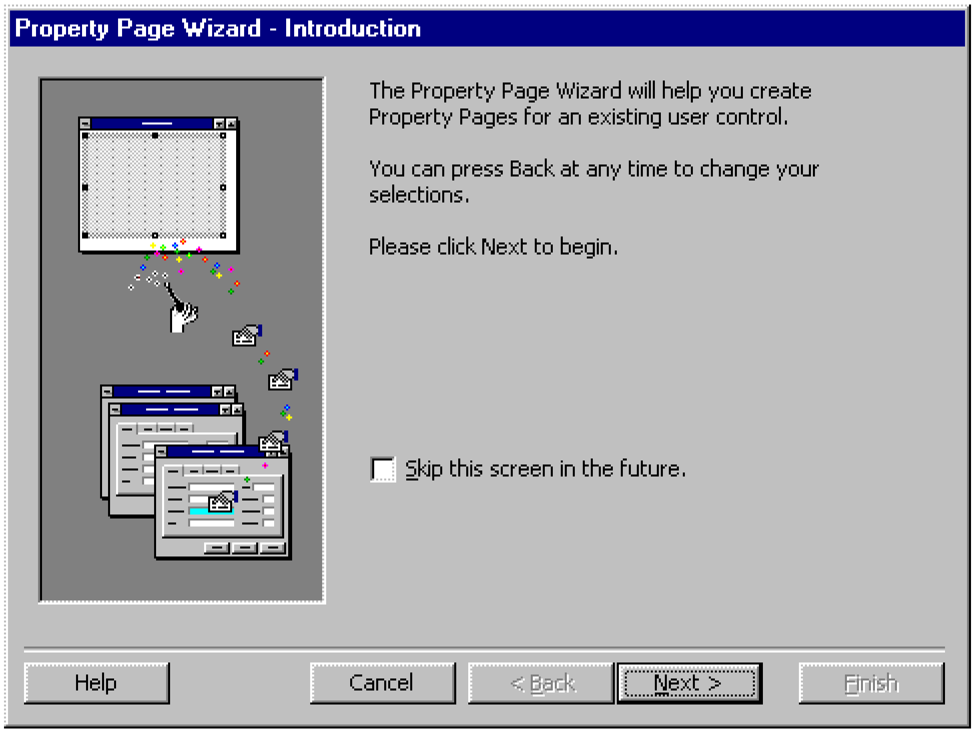
Visual Basic makes it simple to create and include property pages with your control. The Property Page Wizard does most of the work for you, and once you’ve set up your property pages, Visual Basic handles their display and navigation. Visual Basic even provides common property pages (fonts and colors) for free! Although you can create property pages from scratch (choose the **File|Add Property Page** menu), don’t. The Wizard makes this trivial.

Using the Property Page Wizard

1. **Double click on the Meter control** in the Project window to select it.

2. Pull down the Add-Ins menu. If you don’t see **Property Page Wizard…** on the menu, use the Add-In Manager to add it.

3. Choose the **Add-Ins|Property Page Wizard…** menu item. You’ll be greeted with an introductory page (unless you’ve told the wizard not to show you this page). Figure 17 shows the introductory page.



The Property Page Wizard will help you create Property Pages for an existing user control.

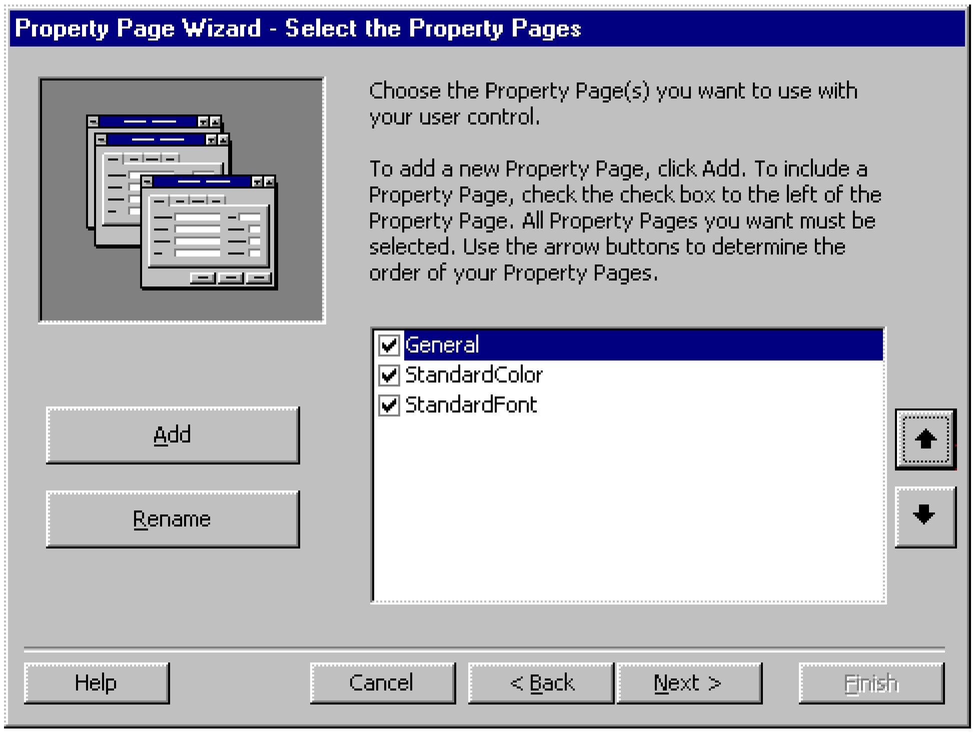
You can press Back at any time to change your selections.

Please click Next to begin.

☐ Skip this screen in the future.

Figure 17. Introduction to the Property Page wizard.

4. **Add custom pages**: The second page proposes two standard property pages, StandardFont and StandardColor. Because our control uses font and color properties, leave both enabled. **Click the Add button** to add a new page, named **General**, and use the up and down arrows to move it to the top of the list. Figure 18 shows the wizard page once you've added the new page.


Property Page Wizard - Select the Property Pages

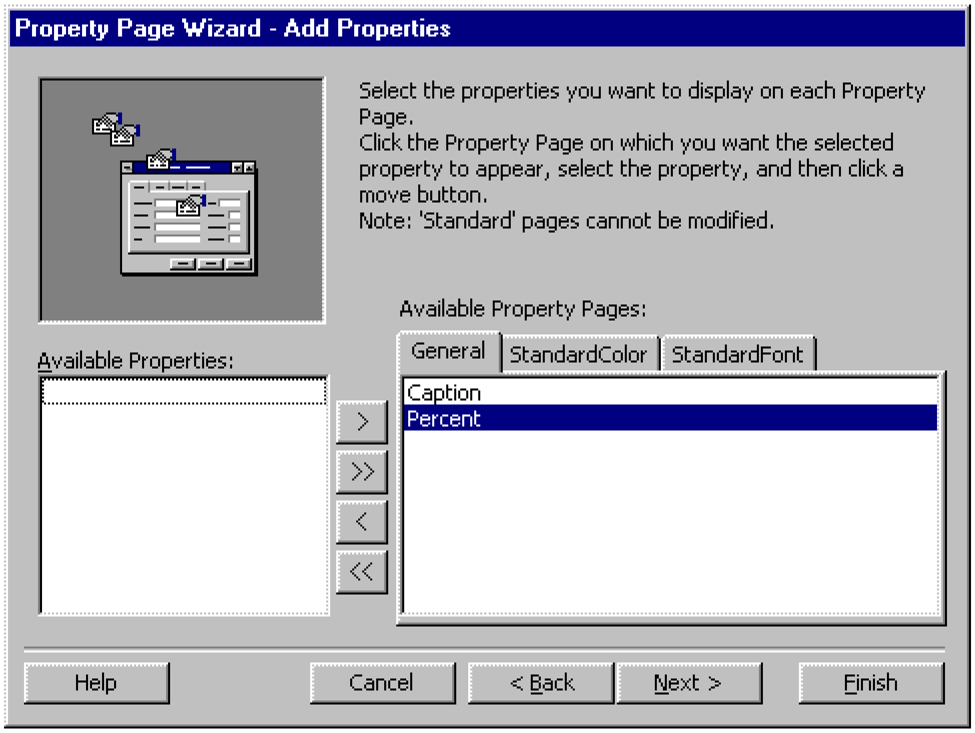
Choose the Property Page(s) you want to use with your user control.

To add a new Property Page, click Add. To include a Property Page, check the check box to the left of the Property Page. All Property Pages you want must be selected. Use the arrow buttons to determine the order of your Property Pages.

- [✓] General
- [✓] StandardColor
- [✓] StandardFont

Figure 18. Use the Property Page Wizard to use existing and add new property pages.

5. **Assign properties to Property Pages**: On the next page, select properties and assign them to the appropriate property page. The wizard does its best to guess which page is the correct one. In this case, add **Percent** and **Caption** to the **General** page. Once you're done, the wizard page should look like the page shown in Figure 19.



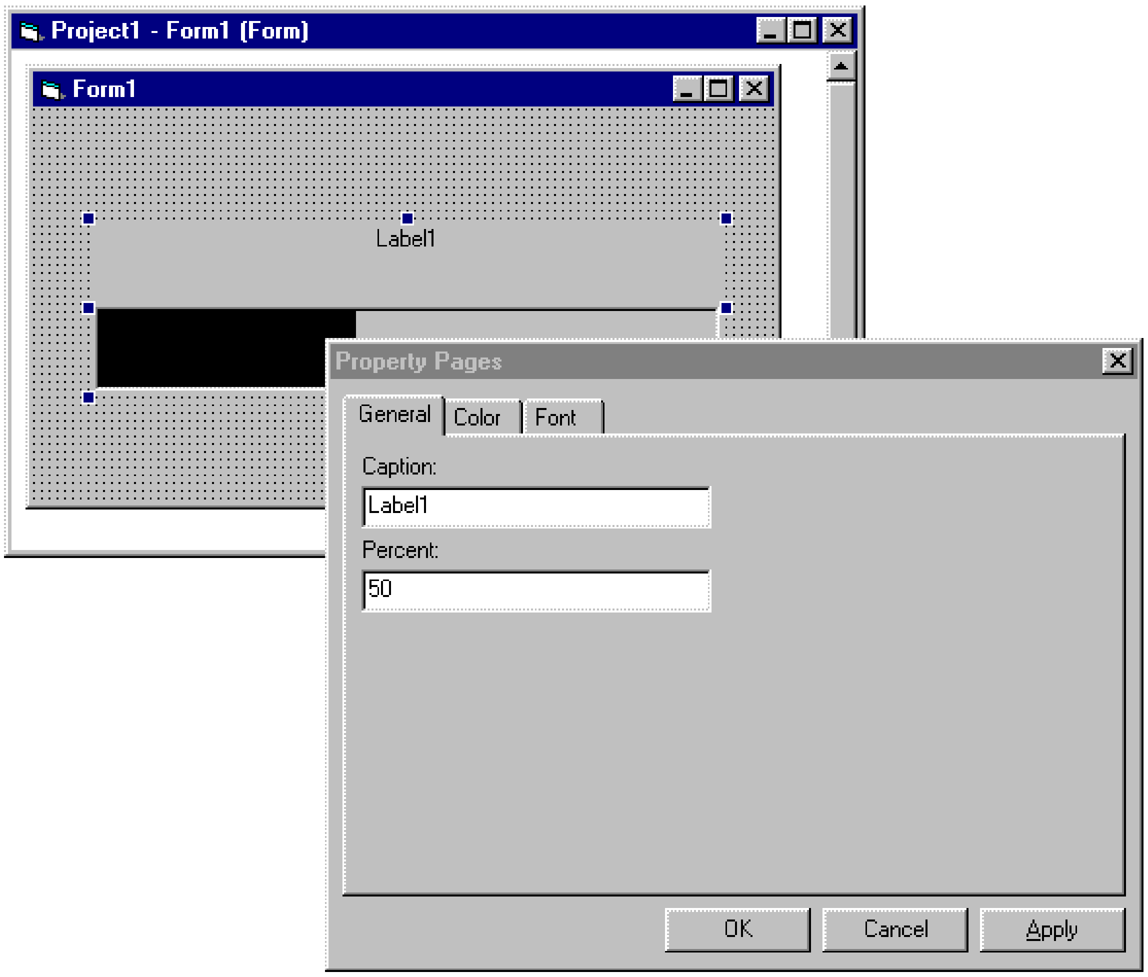
Select the properties you want to display on each Property Page.
Click the Property Page on which you want the selected property to appear, select the property, and then click a move button.
Note: 'Standard' pages cannot be modified.

Caption
Percent

Figure 19. Use the wizard to assign properties to pages.

6. **Click the Finish button**: Complete the Wizard's duties by clicking the Finish button. This will insert a new PropertyPage object into your project.

7. **Test the Property Pages**: Ensure that all the designer windows for the UserControl are closed, then right-click on a Meter control on Form1, and choose the Properties item. You should see a Property Page as shown in Figure 20.



Caption:
Label1
Percent:
50

Figure 20. A user-created Property Page in action.

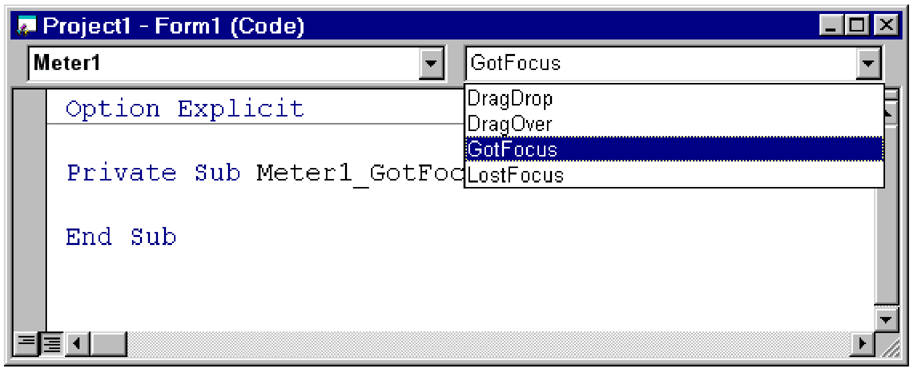
8. **Save the project**: When you save the project, the property pages are stored as .PAG files. Save your project now.

NOTE The use of property pages with multiple controls complicates the issues somewhat. Check out the Visual Basic Books Online for more information on this advanced topic.

Adding Events to the ProgressMeter Control

So now you've got your control working moderately well, and you'd like to use it on a form. On the form, you'd like to display the current percent value for the meter in a text box. You peruse the available events for the control, and notice there's no Change event. You certainly need one, in order to solve your problem.

If you'd like to write code to react to events generated by your ActiveX control, you'll be sorely disappointed. Double-clicking on the control while it's on a form in design view will give you a few meager options, but if you want more interesting events, you'll need to write the code yourself. Figure 21 shows the events you'll be offered, until you take matters into your own hands and add your own events.



```
Option Explicit

Private Sub Meter1_GotFoc

End Sub
```

Figure 21. UserControls don't supply many interesting events, by default.

In this section, we'll add two events to your control, so that programmers using the control can react to both Click and Change events. Raising events from your controls is no different than raising events from any other ActiveX component, and requires two steps:

- Declare the event, using the Event keyword.
- Raise the event, using the RaiseEvent keyword.

Follow these steps to add the two events to the Meter control:

1. **Declare the events**: Add the following code to the Declarations area of your UserControl's module, to declare the events for later use:

```
' Code example #11
Public Event Click()
Public Event Change()
```

2. **Add the code to raise the Click event**: Add the following to the control's module:

```
' Code example #12
Private Sub lblCaption_Click()
    RaiseEvent Click
End Sub

Private Sub picMeter_Click()
    RaiseEvent Click
End Sub
```

3. **Add code to raise the Change event**: Add the RaiseEvent statement to the existing SetPercent procedure, so that it looks like the code below, or replace the existing procedure with the one below:

```
' Code example #13
Private Sub SetPercent()
    shpMeter.Width = picMeter.Width * Me.Percent / 100
    RaiseEvent Change
End Sub
```

That's all it takes to add the two events. Time to test it out! To do so, follow these steps:

1. If you don't have an instance of the Meter control on Form1, add one (Meter1).
2. Add a **Horizontal scroll bar** (Hscroll1) to the form, and **set its Max property to be 100**.
3. Add a text box (Text1) to the form, and delete the default value from its Text property.
4. Double click on the Meter control, choose Change from the list of events, and add code so that the procedure looks like this:

```
' Code example #14
Private Sub Meter1_Change()
    Text1 = Meter1.Percent
End Sub
```

5. Add code to react to the Change event of HScroll1:

```
' Code example #15
Private Sub HScroll1_Change()
    Meter1.Percent = HScroll1.Value
End Sub
```

6. Give it a try! As you change the scroll bar's value, the meter ought to progress, and the value in the text box should reflect the percentage in the progress meter.

You've done it!

TIP: You can also use the ActiveX Control Interface Wizard to add events to your controls, but getting the RaiseEvent calls in the right place can be tricky, and limiting, with the Wizard. That's why we opted to add events by hand in this example.

Compiling the ProgressMeter Control

Before you can distribute your ActiveX control, you must compile its project into an OCX file. To do this, and test out the compiled version, follow these steps:

1. Make sure the TestPM project isn't running.

2. In the Project window, select your ActiveX control, so that its project becomes the active project.

3. From the File menu, choose **Make PrgMeter.OCX**. Supply a path for the OCX file. Make sure and save the project, as well.

4. Remove the ActiveX control project from the project group, so you can test the compiled OCX. To do this, right click on the project, and choose **Remove Project**. When you do that, Visual Basic will remind you that another project is using the control, and verify that you want to remove it. Choose Yes. When you remove the project, Visual Basic looks in the registry for the information about your compiled OCX, and replaces TestPM's reference to the control in the project with the new OCX reference.

5. Run the TestPM project. It'll now be using the compiled OCX.

For even more fun, try using PrgMeter.OCX in other applications, such as Microsoft Excel, as well.

Distributing the ProgressMeter Control

To distribute your ActiveX control to users who may not already have Visual Basic installed on their computers, you'll need to run the Package and Deployment Wizard, and build a set of distribution diskettes. If you're going to be distributing your control via the Internet, on a web page, you'll need to follow the same steps.

The process is simple, and requires almost no intervention that's specific to ActiveX controls. The only question happens near the end, when the Wizard asks if your ActiveX control will be used anywhere besides in the Visual Basic environment. If so, you must include the property page DLL. You'll need to answer that question, then finish the steps.

Once you've answered all the questions, the wizard creates a full installation set for your control, including the Visual Basic run-time library.

If you're planning on distributing your ActiveX control for use on web pages, there are a few more steps. We'll discuss those options in the next section.

TIP: To avoid creating multiple CLSID entries in the registry, as you build and test your component, turn on Binary Compatibility (and point to the OCX file you've already built). That way, every time you build a new version, you'll reuse the existing GUIDs. Of course, if you modify public interfaces, you'll need to turn off compatibility and recreate the component, using new GUIDs.

ActiveX Controls and the Internet

Using ActiveX Controls on Web Pages

So, you've written an ActiveX that you'd like to use as part of a web page. A few questions arise:

- How do you insert it into the web page?
- How do you get the control onto clients' machines, so they can view the control on the web page? (At this point, all ActiveX controls used in web pages must reside on the client machine.)
- How do you convince the browser to load your control, given all the issues dealing with security?

The following sections will demonstrate setting up a control for web page usage, and will, along the way, answer these questions.

Preparing for Internet Download

You'd like to use the ProgressMeter control on a web site. The first step, then, is to package the control for download. Web browsers that support ActiveX controls understand how to download components they need, and only the components they need. But you've got to help out by packaging the component and placing it somewhere the browser can find it (on the web server).

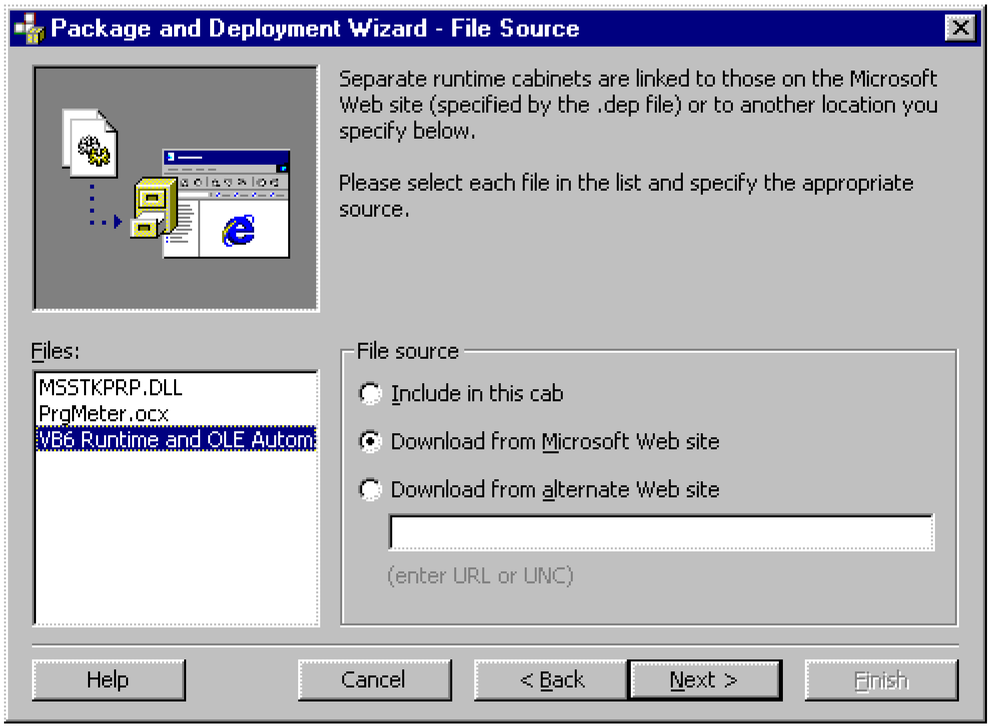
To package an Internet component for download, follow these steps:

1. From within the Visual Basic development environment, with the ProgressMeter project loaded, run the Package and Deployment Wizard. After choosing the **Package** portion, choose the **Internet Package** option (as opposed to the normal Standard Setup Package option).

2. Follow the same path as for a normal package.

3. On the File Source page (see Figure 22), choose the download location for run-time components. Unless you're in an Intranet environment, you don't want people downloading Microsoft's components from your web site (they have a lot more bandwidth than you do!). In addition, if you have people download them from Microsoft's site, they're guaranteed to get the most current versions.

NOTE Only necessary files will be downloaded. That is, if a user already has the Visual Basic run-time DLL on the machine, visiting a web site that uses your ActiveX control won't cause them to download the (rather large) file again.



Separate runtime cabinets are linked to those on the Microsoft Web site (specified by the .dep file) or to another location you specify below.

Please select each file in the list and specify the appropriate source.

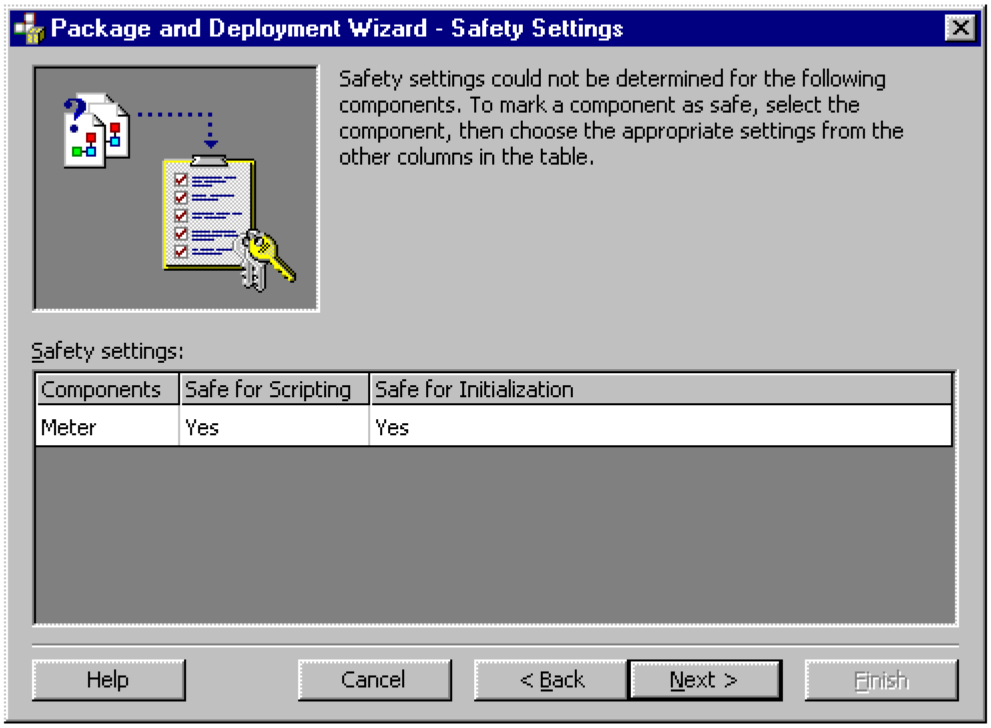
Files:
MSSTKPRP.DLL
PrgMeter.ocx
VB6 Runtime and OLE Autom

File source
( ) Include in this cab
(•) Download from Microsoft Web site
( ) Download from alternate Web site
(enter URL or UNC)

Figure 22. Let users download run-time components from Microsoft.

4. On the Safety Settings page, make sure the **Safe for Initialization** and **Safe for Scripting** options are both set to Yes for your control (see Figure 23).

WARNING!	By checking these options, you're only specifying that you think your control is safe. You're not providing any guarantees, and Internet Explorer will only display the control if the **browser's Safety Level option setting is set to Medium.**



Safety settings could not be determined for the following components. To mark a component as safe, select the component, then choose the appropriate settings from the other columns in the table.

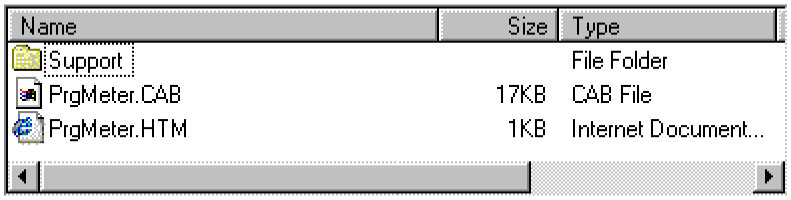
Safety settings:

Components	Safe for Scripting	Safe for Initialization
Meter	Yes	Yes

Figure 23. Is it safe?

5. Click **Next** to open the Finished! page. At this point, you can save the information you've entered as a template for the next time, if you like.

6. Check out the files placed into the output directory. Figure 24 shows the output files, the CAB file containing the download for your ActiveX control, and a sample HTM file demonstrating how to insert your control onto a web page.



Name	Size	Type
Support		File Folder
PrgMeter.CAB	17KB	CAB File
PrgMeter.HTM	1KB	Internet Document...

Figure 24. The Package and Deployment Wizard creates a CAB file and a sample HTM file.

Testing the Control

To test the control in its web page, follow these steps:

1. Make sure the control isn't registered on your machine. To do this, and open Explorer, navigate to the directory containing your OCX. From the

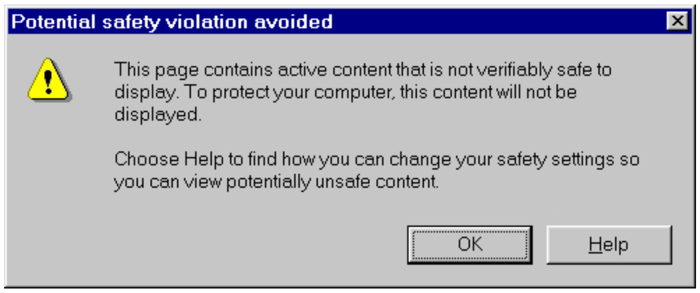
Start menu, choose Run, and type (replacing the general name with your own):

```
REGSVR32 /U C:\YourPath\YourFile.OCX
```

2. Start Internet Explorer 3.0 or 4.0. In IE3, choose the **View|Options|Security** dialog box, and select the **Safety Level** command button. Select the High setting. Close all the dialogs. In IE4, choose the **View|Internet Options|Security** dialog, select the zone, and select the High setting for the correct zone.

3. To test the security:

   - **In IE3**: From Windows Explorer, double-click on PRGMETER.HTM to load it into your browser. It should fail, complaining about security. In IE3, the dialog looks like that shown in Figure 25.
   - **In IE4**: You can't simply click on the file to load it in IE4 and expect security settings to work. You must actually move the files to a directory managed by your web server, and use the HTTP protocol to load the page.


Potential safety violation avoided

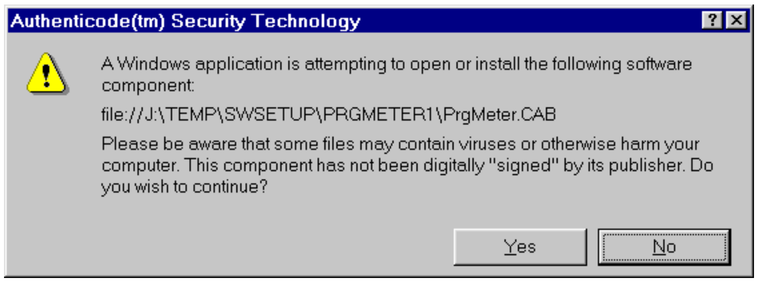
This page contains active content that is not verifiably safe to display. To protect your computer, this content will not be displayed.

Choose Help to find how you can change your safety settings so you can view potentially unsafe content.

Figure 25. High security settings won't work for your unsigned ActiveX control.

4. Go back to Internet Explorer, and change the **Safety Level** setting to be **Medium**.

5. Again, load PRGMETER.HTM. This time, you'll get a different dialog box, but your control will get loaded. (See Figure 26 for IE3's dialog.)


Authenticode(tm) Security Technology

A Windows application is attempting to open or install the following software component:

file://J:\TEMP\SWSETUP\PRGMETER1\PrgMeter.CAB

Please be aware that some files may contain viruses or otherwise harm your computer. This component has not been digitally "signed" by its publisher. Do you wish to continue?

Figure 26. Isn't paranoia a wonderful thing?

What's in that HTML?

If you investigate the sample web page the Setup Wizard creates for you, you'll find this important chunk of HTML code:

```
<OBJECT ID="Meter" WIDTH=395 HEIGHT=80
CLASSID="CLSID:C9BC50B0-F241-11D0-82BA-00AA00698579"
CODEBASE="PrgMeter.CAB#version=1,0,0,0">
</OBJECT>
```

The important parts of this HTML tag are described in the following table.

CLASSID	Contains the GUID (ClassID) for the ActiveX control. This is the only value in this table that you must not change. (This may be different on your machine.)
ID	Name of the control, necessary when using scripting to manipulate the control.
CODEBASE	Location and minimum version number of the control. You'll need to modify the location to match your web site.
WIDTH	Width of the control's space on the web page, in pixels.
HEIGHT	Height of the control's space on the web page, in pixels.

If you want to try setting parameters in the HTML, add a PARAM tag, like these (setting values for the Caption and Percent properties):

```
<OBJECT ID="Meter" WIDTH=395 HEIGHT=80
CLASSID="CLSID:C9BC50B0-F241-11D0-82BA-00AA00698579"
CODEBASE="PrgMeter.CAB#version=1,0,0,0">
<PARAM NAME="Caption" VALUE="This is a progress meter">
<PARAM NAME="Percent" VALUE="72">
</OBJECT>
```

Summary

Rather than summarizing the chapter, this section presents a list of steps you should follow when creating an ActiveX control:

1. Determine the features your control will provide.

4. Design the appearance of your control.

5. Design the interface for your control — that is, the properties, methods, and events your control will expose.

6. Create a project group consisting of your control project and a test project.

7. Implement the appearance of your control by adding controls and/or code to the UserControl object.

8. Implement the interface and features of your control.

9. As you add each interface element or feature, add features to your test project to exercise the new functionality.

10. Design and implement property pages for your control.

11. Compile your control component (.ocx file) and test it with all potential target applications.

Questions

1. What makes an ActiveX control and an ActiveX DLL similar?
2. What's the simplest way to add properties to an ActiveX control?
3. How can you enable the icon for the new ActiveX control in the toolbox?
4. How do you raise events from your new ActiveX control?

Answers

1. What makes an ActiveX control and an ActiveX DLL similar?
   They're both run in-process.

2. What's the simplest way to add properties to an ActiveX control?
   Run the ActiveX Control Interface Wizard.

3. How can you enable the icon for the new ActiveX control in the toolbox?
   Make sure all the designer windows for the control are closed.

4. How do you raise events from your new ActiveX control?
   Use the Event declaration, and the RaiseEvent statement.