15 Working with the Actions Pane

Introduction to the Actions Pane

Developing a solution that runs within an Office application provides considerable benefits because you can take advantage of the functionality that already exists in Office. Sometimes, however, it is hard to design a user interface that meets your needs, as most of the user interface space is controlled by the Office application. Office 2003 and VSTO introduce a number of new user interface capabilities, including the ability to use Windows Forms controls on the document. (See Chapter 14, "Using Windows Forms in VSTO," for more information on this capability.)

Placing a control on the document is not always the right paradigm for the user interface of your application. Putting a control onto the document can often lead to issues with layout when the controls are laid out relative to a range or paragraph, for example. If you use a button on a Word document, by default, it will be inline with the text. This means that when you reformat the document, the button will move with the text. Obviously, being able to move a control with the text is something that you would want if you are developing a flow-based user interface. But this model quickly becomes difficult when you are developing more traditional user interfaces. Things get even more complex if you start to consider what type of behavior you want when the user prints a document. Do you want your

Windows Forms controls to be printed with the rest of the document, for example?

To address these user interface challenges, Office 2003 introduced the ability to put your own custom user interface into the Document Actions task pane of Word and Excel. The task pane is designed to provide a contextual user interface that is complementary to the document. Word, for example, provides a task pane that shows the styles and formats available in the current document and displays the style of the current selection in the document, as shown in Figure 15.1. To display the task pane, choose Task Pane from the View menu.

The active task pane can be changed by making a selection from the drop-down list of available task panes at the top of the task pane, as shown in Figure 15.2. The active task pane is a per-document setting. You can have only one task pane visible at a time per document. The drop-down list shows several task panes that are built into Office. The task pane acts

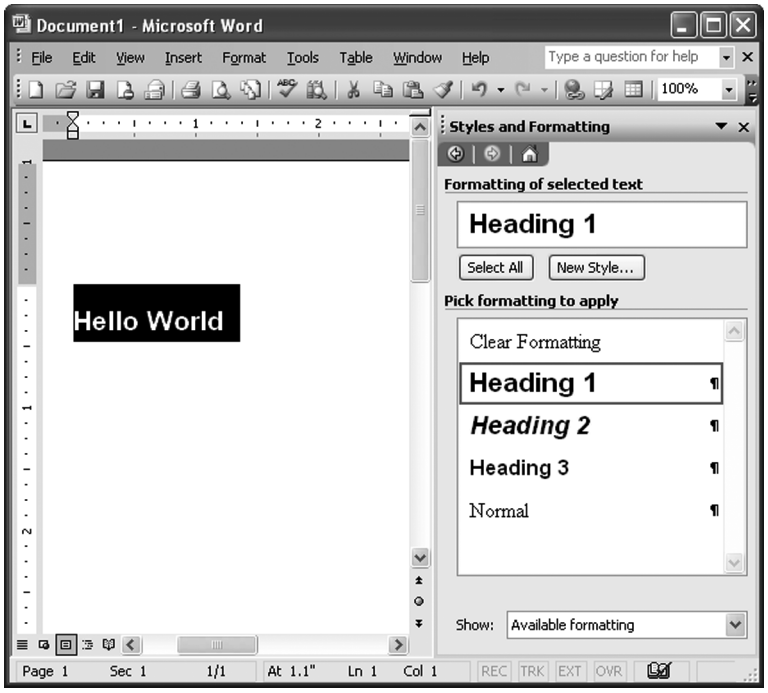

FIGURE 15.1: The Styles and Formatting task pane in Word.

FIGURE 15.2: Selecting a task pane in Word.

like a toolbar when you drag it to move it to another location. It can float above the document. It can also be docked to the left, top, right, or bottom of the application window space.

Figure 15.2 lists several of the built-in task panes available in Word, including Getting Started, Help, and Clip Art. The task pane in the list that is customizable by your VSTO Word or Excel application is called the Document Actions task pane. In VSTO and in this book, we often refer to the Document Actions task pane as the actions pane, as kind of a contraction between the Document Actions and the task pane. ActionsPane is the name of the control in the VSTO programming model that you will use to put your own content in the Document Actions task pane. Note that the Document Actions task pane is listed as an available task pane for a document that has a VSTO customization associated with it that uses the ActionsPane control.

Listing 15.1 shows a simple VSTO Excel customization that displays a Windows Forms button control in the Document Actions task pane. In Excel, the ActionsPane control is a member of the ThisWorkbook class. Because this code is written in Sheet1, we use the Globals object to access

the ThisWorkbook class and, from the ThisWorkbook class, to access the ActionsPane control. The ActionsPane control has a Controls collection that contains the controls that will be shown in the Document Actions task pane. We add to this collection of controls a Windows Forms button control we created previously. Note that just the action of adding a control to the Controls collection causes the Document Actions task pane to be shown at startup.

LISTING 15.1: A VSTO Excel Customization That Adds a Button to the Actions Pane

```
Public Class Sheet1

  Public myButton As New Button

  Private Sub Sheet1_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup

    myButton.Text = "Hello World"
    Globals.ThisWorkbook.ActionsPane.Controls.Add(myButton)

  End Sub

End Class
```

Figure 15.3 shows the result of running Listing 15.1. The Document Actions task pane is shown with a Windows Forms button displayed in the pane.

Listing 15.2 shows a similar VSTO Word customization that displays a Windows Forms Button control in the Document Actions task pane. In Word, the ActionsPane control is a member of the ThisDocument class.

LISTING 15.2: A VSTO Word Customization That Uses the Actions Pane

```
Public Class ThisDocument

  Public myButton As New Button

  Private Sub ThisDocument_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup

    myButton.Text = "Hello World"
    ActionsPane.Controls.Add(myButton)

  End Sub

End Class
```

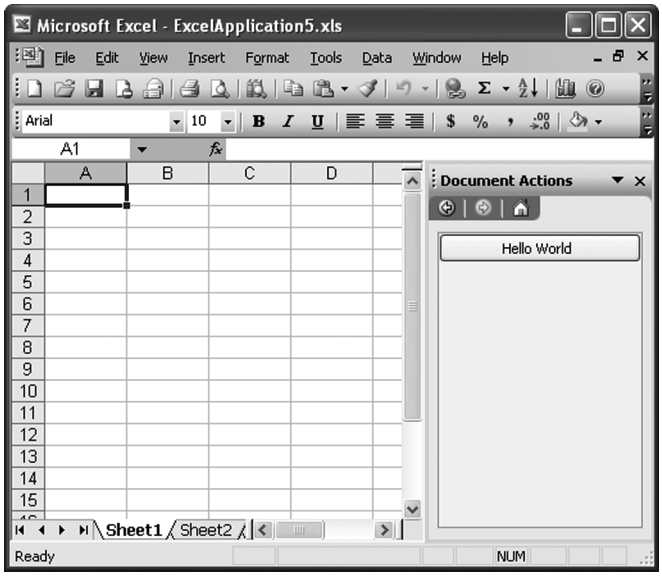

FIGURE 15.3: The result of running Listing 15.1.

The Document Action task pane is actually part of a larger application development platform provided in Office 2003 called Smart Documents. The vision was that Smart Documents would integrate the new XML features available in Word and Excel and in the Document Actions task pane. This combination of XML and the Document Actions task pane provides an application development platform that makes it easier to build documents that are "smart" about their content and provide the appropriate user interface.

Smart Documents were designed primarily for the COM world. So although Smart Documents provided a powerful platform, they did not fit easily into the .NET development methodology. Why?

1. The way you create a Smart Document is first to create a component that implements the ISmartDocument interface. This interface is rather COM-centric.
2. To use a Smart Document, you must have XML schema mapped in your document. Although XML mapping provides considerable functionality to your application programming (see Chapter 21,

"Working with XML in Excel," and Chapter 22, "Working with XML in Word"), not all documents need or want to use XML mapping.

3. The Document Actions task pane supports only a small set of built-in controls and ActiveX controls. To use a Windows Forms control, you would have to register it as an ActiveX control and then attempt to get that to work within the Document Actions task pane. This requires COM registration and COM interop.
4. The Smart Documents infrastructure requires you to create an expansion pack, which includes the following:
   - Manifest.xml, which contains links to all the components within the expansion pack
   - Document to be used
   - Schema for the Smart Document
   - Configuration XML file, which contains the definition of the controls to be used

VSTO provides the ActionsPane control to give you access to all the features provided by Smart Documents with a much more .NET development experience. You do not have to implement the ISmartDocument interface or use schema mapping in the document. You do not have to register Windows Forms controls in the registry so that they can act as ActiveX controls. You do not have to create an expansion pack. Because using the ActionsPane control is so much simpler than using Smart Documents and provides all the benefits, this book does not consider building Smart Documents in the old COM way.

The ActionsPane feature of VSTO is actually implemented under the covers as a specialized Smart Document solution; when you look at a customized VSTO document and examine the attached XML schemas, you will see that a schema called ActionsPane is attached automatically. This schema provides the plumbing to connect VSTO's ActionsPane control to the Smart Document platform. When you install the VSTO runtime (see Chapter 20, "Deployment"), the ActionsPane schema is also installed and registered with Excel and Word, enabling the ActionsPane control to access the Document Actions task pane.

Working with the ActionsPane Control

A first step in understanding how VSTO's ActionsPane control works is delving a little into the architecture of VSTO's ActionsPane support.

The ActionsPane Architecture

The Document Actions task pane is a window provided by Office that can host ActiveX controls, as shown in Figure 15.4. VSTO places a special invisible ActiveX control in the Document Actions task pane that in turn hosts a single Windows Forms UserControl. This UserControl is represented in the VSTO programming model by the ActionsPane control—accessible in Word via Document.ActionsPane and accessible in Excel via Globals.ThisWorkbook.ActionsPane.

Although the Document Actions task pane can host multiple ActiveX controls, VSTO needs to put only a single ActiveX control and a single UserControl in the Document Actions task pane window, because the UserControl can host multiple Windows Forms controls via its Controls collection (ActionsPane.Controls). You can add Windows Forms controls to the ActionsPane by using the ActionsPane.Controls.Add method.

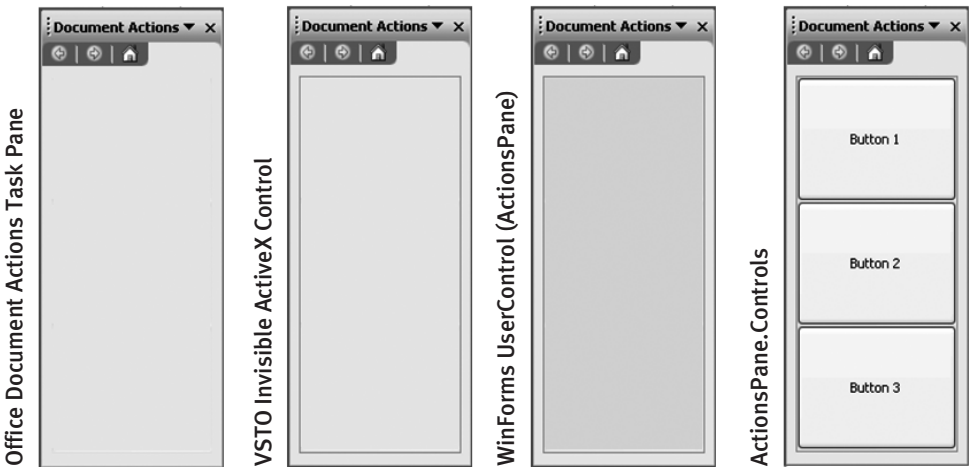

FIGURE 15.4: The four layers of the ActionsPane architecture.

The UserControl placed in the ActionsPane window is set to expand to fit the area provided by the ActionsPane window. If the area of the Document Actions task pane is not big enough to display all the controls hosted by the UserControl, it is possible to scroll the UserControl by setting the AutoScroll property of ActionsPane to `True`.

The ActionsPane control is a wrapper around System.Windows .Forms.UserControl with most of the properties, methods, and events of a UserControl. It also adds some properties, events, and methods specific to ActionsPane. When you understand the architecture in Figure 15.4, you will not be too surprised to know that some properties from UserControl that are exposed by ActionsPane—such as position-related properties, methods, and events—do not do anything. Because the position of the ActionsPane UserControl is forced to fill the space provided by the Actions-Pane window, for example, you cannot reposition the UserControl to arbitrary positions within the Document Actions task pane window.

Adding Windows Forms Controls to the Actions Pane

The basic way you add your custom UI to the actions pane is to add Windows Forms controls to the actions pane's Controls collection. Listing 15.1 illustrates this approach. First, it declares and creates an instance of a System.Windows.Forms.Button control. Then this control is added to the actions pane by calling the Add method of the Controls collection associated with the actions pane and passing the button instance as a parameter to the Add method.

The actions pane is smart about arranging controls within the Actions-Pane. If multiple controls are added to the Controls collection, the actions pane can automatically stack and arrange the controls. The stacking order is controlled by the ActionsPane.StackOrder property, which is of type `Microsoft.Office.Tools.StackStyle`. It can be set to `None` for no automatic positioning, or it can be set to `FromTop`, `FromBottom`, `From-Left`, or `FromRight`. Figure 15.5 shows the effects of the various Stack-Order settings.

Listing 15.3 shows some code that adds and positions controls in the actions pane when StackOrder is set to `StackStyle.FromBottom` and automatically positioned or set to `StackStyle.None` and manually positioned.

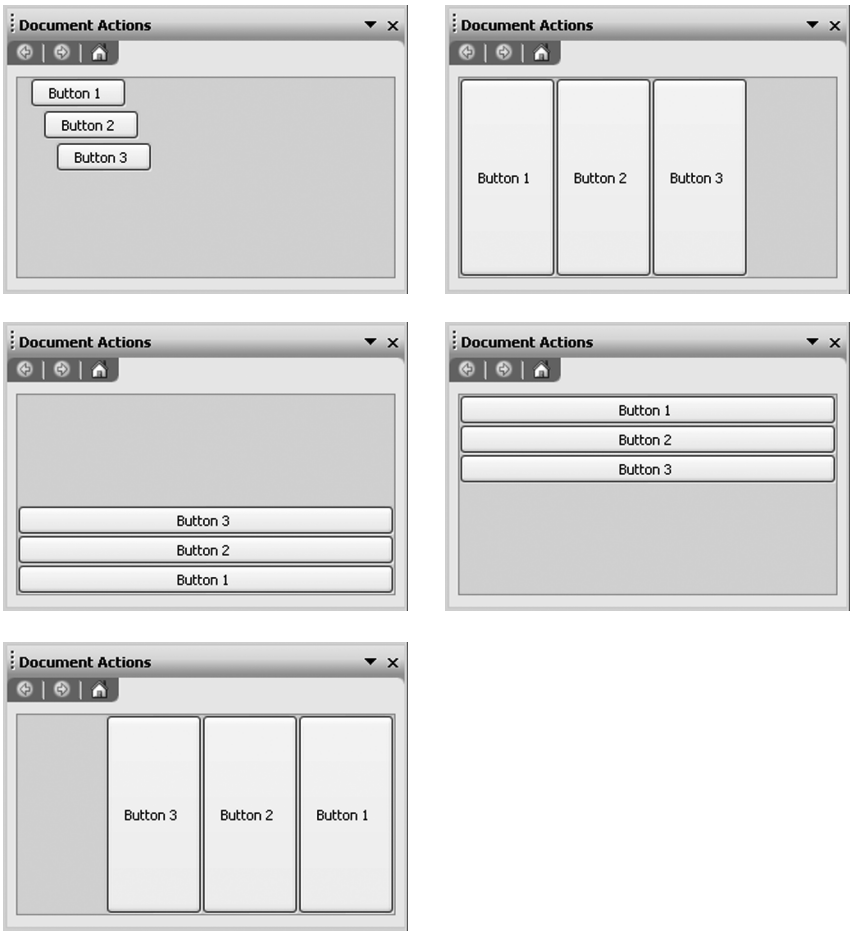

FIGURE 15.5: The results of changing the ActionsPane StackOrder setting, from top left: `None`, `FromLeft`, `FromBottom`, `FromTop`, **and** `FromRight`.

LISTING 15.3: A VSTO Excel Customization That Adds and Positions Controls with Either StackStyle.None or StackStyle.FromBottom

```
Public Class Sheet1

  Public button1 As New Button
  Public button2 As New Button
  Public button3 As New Button

  Private Sub Sheet1_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup
```

```
    button1.Text = "Button 1"
    button2.Text = "Button 2"
    button3.Text = "Button 3"

    Globals.ThisWorkbook.ActionsPane.BackColor = Color.Aquamarine

    Globals.ThisWorkbook.ActionsPane.Controls.Add(button1)
    Globals.ThisWorkbook.ActionsPane.Controls.Add(button2)
    Globals.ThisWorkbook.ActionsPane.Controls.Add(button3)

    If MsgBox("Do you want to auto-position the controls?", _
       MsgBoxStyle.YesNo, "StackStyle") = MsgBoxResult.Yes Then

      Globals.ThisWorkbook.ActionsPane.StackOrder = _
        Microsoft.Office.Tools.StackStyle.FromBottom
    Else
      Globals.ThisWorkbook.ActionsPane.StackOrder = _
        Microsoft.Office.Tools.StackStyle.None

      button1.Left = 10
      button2.Left = 20
      button3.Left = 30

      button1.Top = 0
      button2.Top = 25
      button3.Top = 50

    End If

  End Sub

End Class
```

Adding a Custom User Control to the Actions Pane

A more visual way of designing your application's actions pane user interface is to create a user control and add that user control to the ActionsPane's control collection. Visual Studio provides a rich design-time experience for creating a user control. To add a user control to your application, click the project node in Solution Explorer, and choose Add User Control from Visual Studio's Project menu. Visual Studio will prompt you to give the User Control a filename, such as UserControl1.vb. Then Visual Studio will display the design view shown in Figure 15.6.

The design area for the user control has a drag handle in the bottom-right corner that you can drag to change the size of the user control. Con-

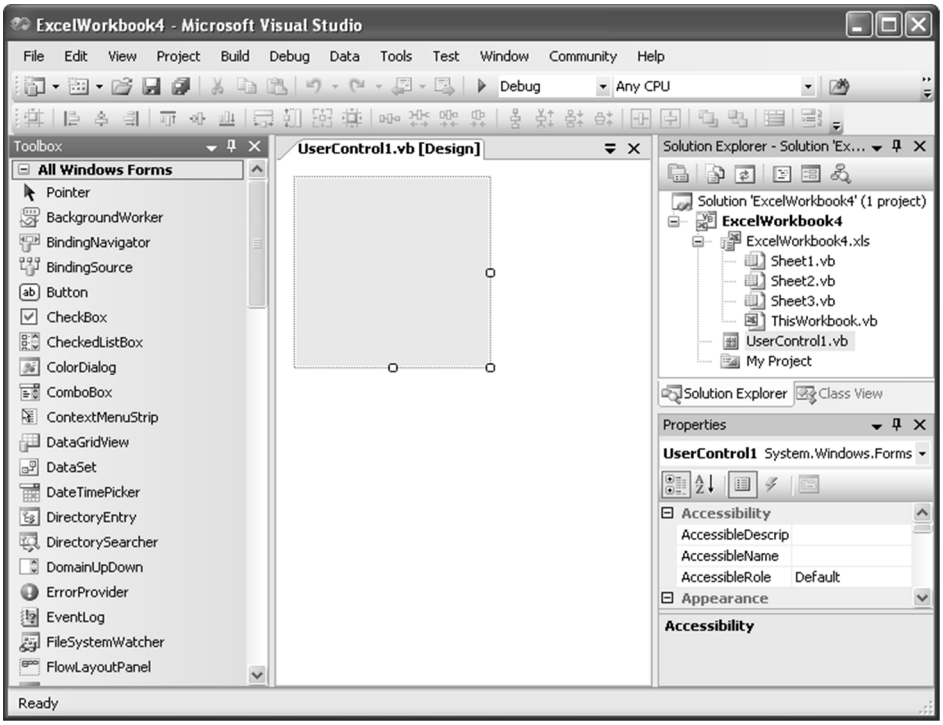

FIGURE 15.6: The design view for creating a custom user control.

trols from the toolbox can be dragged onto the user control design surface and positioned as desired. Figure 15.7 shows a completed user control that uses check boxes, text boxes, and labels.

Listing 15.4 shows a VSTO Excel customization that adds this custom user control to the Document Actions task pane. The user control created in Figure 15.7 is a class named UserControl1. Listing 15.4 creates an instance of

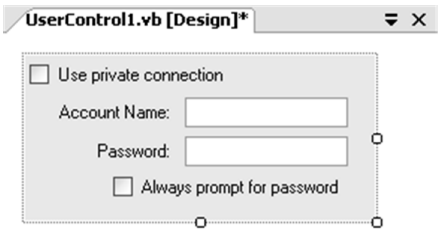

FIGURE 15.7: A custom user control.

UserControl1 and adds it to ActionPane's Controls collection using the Add method.

LISTING 15.4: A VSTO Excel Customization That Adds a Custom User Control to the Task Pane

```
Public Class Sheet1

  Public myUserControl As New UserControl1

  Private Sub Sheet1_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup

    Globals.ThisWorkbook.ActionsPane.Controls.Add(myUserControl)

  End Sub

End Class
```

Figure 15.8 shows the Document Actions task pane that results when Listing 15.4 is run.

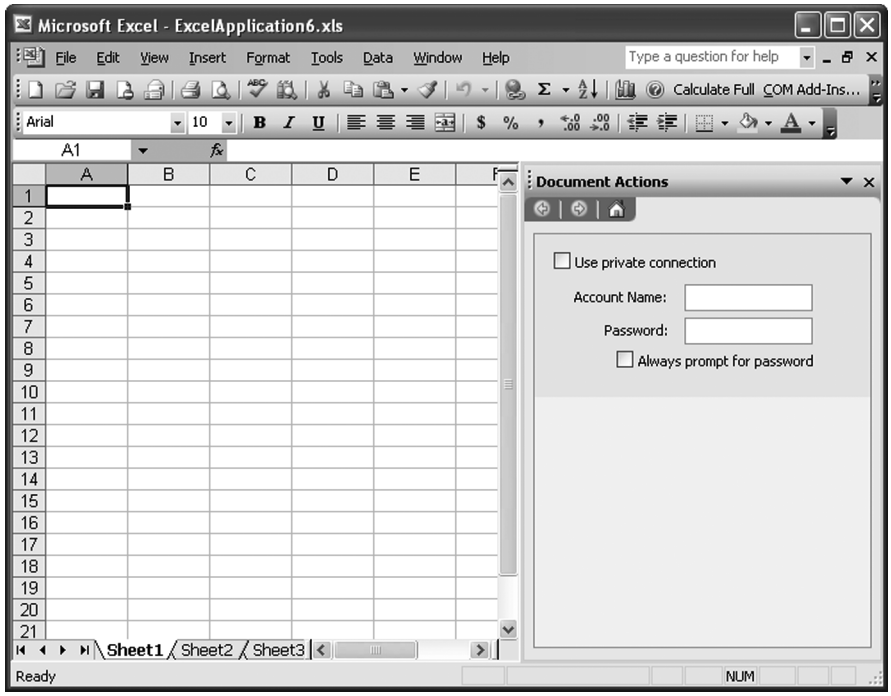

FIGURE 15.8: The result of running Listing 15.4.

Contextually Changing the Actions Pane

A common application of the ActionsPane is providing commands in the Document Actions task pane that are appropriate to the context of the document. In an order-form application, for example, the Document Actions task pane might display a button for selecting a known customer when filling out the customer information section of the document. When the user is filling out the order part of the document, the Document Actions task pane might display a button for examining available inventory.

Listing 15.5 shows a VSTO Excel customization in which two named ranges have been defined. One, called `orderInfo`, is a range of cells where the contents of an order are placed. The other, called `customerInfo`, is a range of cells specifying the customer information for the customer placing the order. Listing 15.5 contextually adds and removes an `inventoryButton` when the `orderInfo` range is selected and a `customerButton` when the `customerInfo` range is selected or deselected. It does this by handling NamedRange.Selected and NamedRange.Deselected events. When the Selected event indicating the `customerInfo` range of cells is selected, Listing 15.5 adds a `customerButton` that, when clicked, would allow the user to pick an existing customer. Listing 15.5 removes the `customerButton` when the customerInfo.Deselected event is raised. It calls ActionsPane.Controls.Remove to remove the `customerButton` from the actions pane.

Listing 15.5 is written in such a way that if the `customerInfo` range and the `orderInfo` range are selected at the same time, both the `customerButton` and the `inventoryButton` would be visible in the Document Actions task pane.

LISTING 15.5: A VSTO Excel Customization That Changes the Actions Pane Based on the Selection

```
Public Class Sheet1

  Public customerButton As New Button
  Public inventoryButton As New Button

  Private Sub Sheet1_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup

    customerButton.Text = "Select a customer..."
    inventoryButton.Text = "Check inventory..."
```

```
  End Sub

  Private Sub orderInfo_Selected( _
    ByVal Target As Microsoft.Office.Interop.Excel.Range) _
    Handles orderInfo.Selected

    Globals.ThisWorkbook.ActionsPane.Controls.Add( _
      inventoryButton)

  End Sub

  Private Sub orderInfo_Deselected( _
    ByVal Target As Microsoft.Office.Interop.Excel.Range) _
    Handles orderInfo.Deselected

    Globals.ThisWorkbook.ActionsPane.Controls.Remove( _
      inventoryButton)

  End Sub

  Private Sub customerInfo_Selected( _
    ByVal Target As Microsoft.Office.Interop.Excel.Range) _
    Handles customerInfo.Selected

    Globals.ThisWorkbook.ActionsPane.Controls.Add(customerButton)

  End Sub

  Private Sub customerInfo_Deselected( _
    ByVal Target As Microsoft.Office.Interop.Excel.Range) _
    Handles customerInfo.Deselected

    Globals.ThisWorkbook.ActionsPane.Controls.Remove( _
      customerButton)

  End Sub

End Class
```

You can also change the contents of the Document Actions task pane as the selection changes in a Word document. One approach is to use bookmarks and change the contents of the Document Actions task pane when a particular bookmark is selected. A second approach is to use the XML mapping features of Word and VSTO's XMLNode and XMLNodes controls (described in Chapter 22, "Working with XML in Word") and to

change the contents of the Document Actions task pane when a particular XMLNode or XMLNodes is selected in the document.

Detecting the Orientation of the Actions Pane

ActionsPane has all the UserControl events documented in the .NET class libraries documentation and one additional event: OrientationChanged. This event is raised when the orientation of the actions pane is changed. The actions pane can be in either a horizontal or a vertical orientation. Figure 15.3 earlier in this chapter shows an actions pane in a vertical orientation. Figure 15.9 shows a horizontal orientation.

Listing 15.6 shows a VSTO Excel customization that adds several buttons to the ActionsPane's Controls collection. Listing 15.6 also handles the OrientationChanged event and displays the orientation of the ActionsPane in a dialog box. It determines the orientation of the actions pane by checking the ActionsPane.Orientation property. The Orientation property returns a member of the `System.Windows.Forms.Orientation` enumeration: either `Orientation.Horizontal` or `Orientation.Vertical`.

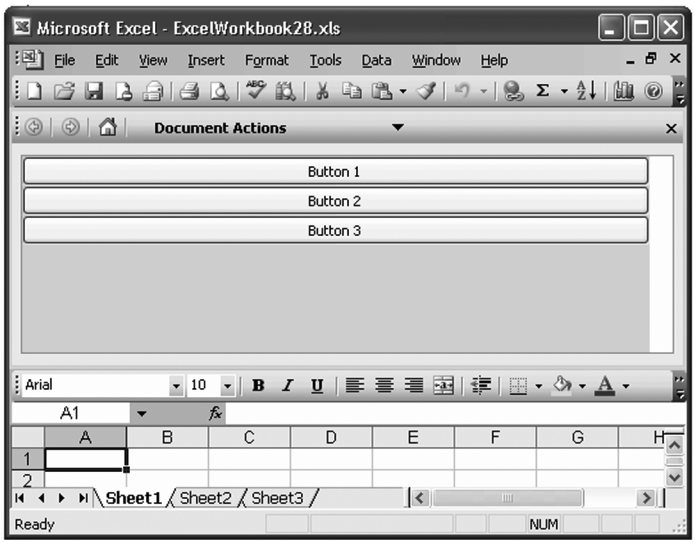

FIGURE 15.9: The actions pane in a horizontal orientation.

LISTING 15.6: A VSTO Excel Customization That Handles ActionsPane's OrientationChanged Event

```
Public Class Sheet1

  Public button1 As New Button
  Public button2 As New Button
  Public button3 As New Button

  Private Sub Sheet1_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup

    button1.Text = "Button 1"
    button2.Text = "Button 2"
    button3.Text = "Button 3"

    Globals.ThisWorkbook.ActionsPane.StackOrder = _
      Microsoft.Office.Tools.StackStyle.FromTop

    Globals.ThisWorkbook.ActionsPane.Controls.Add(button1)
    Globals.ThisWorkbook.ActionsPane.Controls.Add(button2)
    Globals.ThisWorkbook.ActionsPane.Controls.Add(button3)

    Globals.ThisWorkbook.ActionsPane.BackColor = Color.Aquamarine

    AddHandler _
      Globals.ThisWorkbook.ActionsPane.OrientationChanged, _
      AddressOf ActionsPane_OrientationChanged

  End Sub

  Private Sub ActionsPane_OrientationChanged( _
    ByVal sender As Object, _
    ByVal e As EventArgs)

    Dim orientation1 As Orientation = _
      Globals.ThisWorkbook.ActionsPane.Orientation()
    MsgBox(String.Format("Orientation is {0}.", _
     orientation1.ToString()))

  End Sub

End Class
```

Scrolling the Actions Pane

The AutoScroll property of the ActionsPane gets or sets a `Boolean` value indicating whether the actions pane should display a scroll bar when the size of the Document Actions task pane is such that not all the controls can

be shown. The default value of AutoScroll is `True`. Figure 15.10 shows a Document Actions task pane with ten buttons added to it. Because AutoScroll is set to `True`, a scroll bar is shown when not all ten buttons can be displayed, given the size of the Document Actions task pane.

Showing and Hiding the Actions Pane

The actions pane is shown automatically when you add controls to ActionsPane's Controls collection using the Add method. To show and hide the actions pane programmatically, you need to use the Excel or Word object model. In Excel, set the Application.DisplayDocumentActionTaskPane property to `True` or `False`. In Word, set the property Application.TaskPanes[WdTaskPanes.wdTaskPaneDocumentActions].Visible property to `True` or `False`.

You might be tempted to call ActionsPane.Hide or set ActionsPane.Visible to `False` to hide the ActionsPane. These approaches do not work, because you are actually hiding the UserControl shown in Figure 15.4 that is hosted by the Document Actions task pane, rather than just the Document Actions task pane. You should use the object model of Excel and Word to show and hide the actions pane.

Listing 15.7 shows a VSTO Excel customization that shows and hides the actions pane on the BeforeDoubleClick event of the Worksheet by toggling the state of the Application.DisplayDocumentActionTaskPane property. Note that the DisplayDocumentActionTaskPane property is an application-level property that is applicable only when the active document has a

FIGURE 15.10: The actions pane when AutoScroll is set to `True`.

Document Actions task pane associated with it. If the active document does not have a Document Actions task pane associated with it, accessing the DisplayDocumentActionTaskPane property will raise an exception.

LISTING 15.7: A VSTO Excel Customization That Shows and Hides the Actions Pane When Handling the BeforeDoubleClick Event

```
Public Class Sheet1

  Private isVisible As Boolean = True

  Private Sub Sheet1_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup

    Dim i As Integer
    For i = 1 To 10
      Dim myButton As New Button()
      myButton.Text = String.Format("Button {0}", i)
      Globals.ThisWorkbook.ActionsPane.Controls.Add(myButton)
    Next

  End Sub

  Private Sub Sheet1_BeforeDoubleClick( _
    ByVal Target As Microsoft.Office.Interop.Excel.Range, _
    ByRef Cancel As System.Boolean) Handles Me.BeforeDoubleClick

    ' Toggle the visibility of the ActionsPane on double-click.
    isVisible = Not isVisible
    Me.Application.DisplayDocumentActionTaskPane = isVisible

  End Sub

End Class
```

Listing 15.8 shows a VSTO Word application that shows and hides the actions pane on the BeforeDoubleClick event of the Document by toggling the state of the Application.TaskPanes[WdTaskPanes.wdTaskPaneDocumentActions].Visible property.

LISTING 15.8: VSTO Word Customization That Shows and Hides the Actions Pane in the BeforeDoubleClick Event Handler

```
Public Class ThisDocument

  Private Sub ThisDocument_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup
```

```
    Dim i As Integer
    For i = 1 To 10
      Dim myButton As New Button()
      myButton.Text = String.Format("Button {0}", i)
      ActionsPane.Controls.Add(myButton)
    Next

  End Sub

  Private Sub ThisDocument_BeforeDoubleClick( _
    ByVal sender As System.Object, _
    ByVal e As Microsoft.Office.Tools.Word.ClickEventArgs) _
    Handles Me.BeforeDoubleClick

    If Me.Application.TaskPanes( _
      Word.WdTaskPanes.wdTaskPaneDocumentActions).Visible Then

      Me.Application.TaskPanes( _
        Word.WdTaskPanes.wdTaskPaneDocumentActions).Visible _
        = False
    Else
      Me.Application.TaskPanes( _
        Word.WdTaskPanes.wdTaskPaneDocumentActions).Visible _
        = True

    End If

  End Sub

End Class
```

Attaching and Detaching the Actions Pane

Sometimes you will want to go beyond just hiding the actions pane and actually detach the actions pane from the document or workbook. You might also want to control whether the user of your document is allowed to detach the actions pane from the document or workbook. Recall from earlier in this chapter that the actions pane is actually a Smart Document solution, and as such, it can be attached or detached from the document or workbook via Excel and Word's built-in dialog boxes for managing attached Smart Document solutions.

When the actions pane is detached from the document, this means that the Document Actions task pane will not be in the list of available task panes when the user drops down the list of available task panes, as shown in Figure 15.2 earlier in this chapter. To detach the actions pane from the

document programmatically, call the ActionsPane.Clear method. Doing so detaches the actions pane solution from the document and hides the Document Actions pane. Calling ActionsPane.Show reattaches the actions pane and makes it available again in the list of available task panes. Note that in Word, when you call ActionsPane.Clear, you must follow the call with a second call to the Word object model: Document.XMLReferences["ActionsPane"].Delete.

If you want to allow the user of your document to detach the actions pane solution by using the Templates and Add-ins dialog box in Word, shown in Figure 15.11, or the XML Expansion Packs dialog box in Excel, shown in Figure 15.12, you must set the ActionsPane.AutoRecover property to `False`. By default, this property is set to `True`, which means that even when the user tries to detach the actions pane solution by deselecting it in these dialog boxes, VSTO will recover and automatically reattach the actions pane solution.

After an actions pane solution is attached to the document, and the user saves the document, the next time the user opens the document, the actions pane will be available and can be selected at any time during the session. If

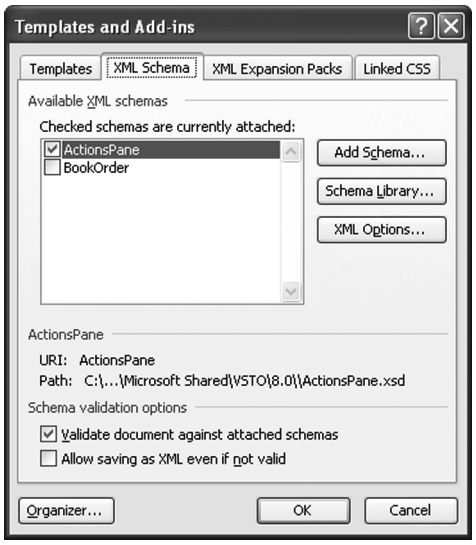

FIGURE 15.11: The ActionsPane solution attached to a Word document is visible in Word's Templates and Add-Ins dialog box and can be removed if ActionsPane.AutoRecover is not set to `True`.

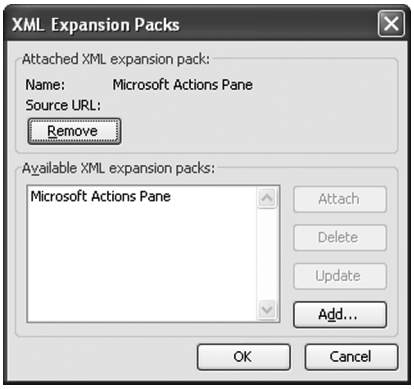

FIGURE 15.12: The ActionsPane solution attached to an Excel workbook is visible in Excel's XML Expansion Packs dialog box and can be removed if ActionsPane.AutoRecover is not set to `True`.

your code does not add controls to the actions pane until some time after startup, you might want to call the ActionsPane.Clear method in the Startup handler of your VSTO customization to prevent the user from showing the actions pane before your VSTO customization has added controls to the ActionsPane control.

Some Methods and Properties to Avoid

As mentioned earlier, the ActionsPane is a user control that has a fixed location and size that are controlled by VSTO. As such, you should avoid using a number of position-related properties and methods on the ActionsPane control, as listed in Table 15.1.

TABLE 15.1: Methods and Properties of ActionsPane to Avoid

Left	Top	Width
Height	Right	Location
Margin	MaximumSize	MinimumSize
Size	TabIndex	AutoScrollMargin
AutoScrollMinSize		

Conclusion

The chapter covered the ActionsPane control in VSTO and how it enables custom UI in Office's Document Actions task pane. The chapter examined the properties, methods, and events unique to the ActionsPane control. You also learned the basic architecture of ActionPane and how ActionsPane has the properties, methods, and events found on a Windows Forms user control.