Walkthrough: Creating a User Control with IronPython

Introduction

You can use IronPython for ASP.NET to create user controls that encapsulate the functionality of multiple server controls in a unit. In this walkthrough, you will create an ASP.NET user control that acts as a picker control. The picker control has two lists, with a set of choices in one list (the source). Users can select items in the `SourceList` list, and then add the items to the `TargetList` list.

Note In this release, you cannot use IronPython to create custom server controls.

Tasks illustrated in this walkthrough include the following:

- Creating a user control and adding ASP.NET server controls to the user control, with event handlers written in IronPython.
- Adding a user control to a host page.
- Adding properties and methods to the user control with IronPython.
- Coding properties in IronPython so that they can be used declaratively.
- Adding IronPython code to the host page to handle the user control.

Prerequisites

In order to complete this walkthrough, you will need:

- Microsoft Visual Studio 2005 or Microsoft Visual Web Developer
- IronPython for ASP.NET Community Technology Preview, including Visual Studio Integration or Visual Web Developer Integration, as appropriate

This walkthrough assumes that you have a general understanding of working with ASP.NET in Visual Studio, and a general knowledge of data binding in ASP.NET. For an introduction to ASP.NET in Visual Studio, see Walkthrough: Creating a Basic Page in Visual Web Developer.

Creating the Web Site

If you have already created a Web site in Visual Studio (for example, by working with the companion walkthrough "Creating a Basic Web Page with IronPython"), you can use that Web site and go to the next section. Otherwise, create a new Web site and page by following these steps.

To create a file system Web site

1. Open Visual Studio or Visual Web Developer.
2. On the **File** menu, point to **New**, and then click **Web Site**.

   The **New Web Site** dialog box appears.
3. Under **Visual Studio installed templates**, click **ASP.NET Web Site**.
4. In the right-most **Location** box, enter the name of the folder where you want to keep the pages of the Web site.

   For example, type the folder name C:\WebSites.
5. In the **Language** list, click **IronPython** to make IronPython the default language for the Web site.
6. Click **OK**.

   Visual Studio creates the folder and a new page named Default.aspx.

Creating the User Control

Creating a user control is similar to creating an ASP.NET Web page. In fact, a user control is a subset of an ASP.NET page and it can include most of the types of elements that you put on an ASP.NET page.

To create a user control

1. In Solution Explorer, right-click the name of the Web site, and then click **Add New Item**.
2. In the **Add New Item** dialog box, under **Visual Studio installed templates**, click **Web User Control**.
3. In the **Name** box, type **ListPicker**.
4. Make sure the **Place code in separate file** check box is checked.

   You can create user controls that do not have a separate code file. If you choose to do so for this walkthrough, switch to source view for ListPicker.ascx and put code in the `script` block whenever the instructions tell you to put code in ListPicker.ascx.py.
5. Click **Add**.
6. Switch to Design view, and on the **Layout** menu, click **Insert Table**.
7. Use the **Insert Table** dialog box to create a table with one row and three columns, and then click **OK**.

   You are creating the table to hold the controls; that is, a layout table.
8. In the table, in the left column, type **Available**, and then press ENTER to create a new line.
9. In the right column, type **Selected**, and then press ENTER to create a new line.
10. From the **Standard** group in the Toolbox, drag the following controls onto the table and set their properties as indicated.

Control	Properties
Drag a **ListBox** to the left column and place it under **Available**.	**Height**: **200px** **ID**: SourceList **Width**: **200px**
Drag a **Button** to the middle column.	**ID**: AddAll **Text**: >>
Drag a **Button** to the middle column.	**ID**: AddOne **Text**: SPACE>SPACE
Drag a **Button** to the middle column.	**ID**: **Remove** **Text**: SPACE**X**SPACE
Drag a **ListBox** to the right column and place it under **Selected**.	**Height**: **200px** **ID**: TargetList **Width**: **200px**

11. Click the `SourceList` list, and then in the **Properties** window, for the **Items** property, click the ellipsis (…) button to open the **ListItem Collection Editor** dialog box, and add three items.
12. Under **ListItem properties**, set the **Text** for the three items to three different values, for example **A**, **B**, and **C**.

Adding Code to Handle User Selections

Users will select items using the buttons that are in the middle column of the table. Therefore, most of the code for the control is in handlers for the **Click** events.

To add code to handle user selections

1. Open ListPicker.ascx.py and add the following event handler for the **Click** event of the **AddAll** button, which adds all items from the source list to the target list. If you chose to create the user control without a separate code file, switch to source view and put the code in the `script` block.

   ```
   def AddAll_Click(sender, args):
       TargetList.SelectedIndex = -1
       for li in SourceList.Items:
           AddItem(li)
   ```

   > **Note** In this release, IronPython event handlers must be coded and bound manually. You cannot create them by double-clicking a control in Design mode, or by selecting an event in the Properties window.

   The IronPython code loops through all the list items in the `SourceList` list. For each item, it calls the `AddItem` method and passes it the current item. Later in this procedure, you will write the code for the `AddItem` method.
2. Add the following event handler for the **Click** event of the **AddOne** button, which adds the selected item of the source list to the target list.

   ```
   def AddOne_Click(sender, args):
       if SourceList.SelectedIndex >= 0:
           AddItem(SourceList.SelectedItem)
   ```

   The IronPython code calls the `AddItem` method if there is a selection in the `SourceList` list.
3. Add the following event handler for the **Click** event of the **Remove** button, removes the selected item from the target list.

   ```
   def Remove_Click(sender, args):
       if TargetList.SelectedIndex >= 0:
           TargetList.Items.RemoveAt(TargetList.SelectedIndex)
           TargetList.SelectedIndex = -1
   ```

   The IronPython code first checks that the `TargetList` list contains a selection, and if so removes the selected item and cancels the selection.
4. Add the following **AddItem** method:

   ```
   def AddItem(li):
       TargetList.SelectedIndex = -1
       TargetList.Items.Add(li)
   ```

   The IronPython code unconditionally adds an item to the `TargetList` list. Later in this walkthrough, in "Adding Custom Properties and Methods to the User Control," you will improve this code by adding the option to determine whether there are duplicates.

5. Switch to ListPicker.ascx, and in **Source** view add **OnClick** attributes to bind the event handlers:

```
<asp:Button ID="AddAll" runat="server" Text=">>"
    OnClick="AddAll_Click"/>
<asp:Button ID="AddOne" runat="server" Text=" > "
    OnClick="AddOne_Click"/>
<asp:Button ID="RemoveOne" runat="server" Text=" X "
    OnClick="Remove_Click"/>
```

Testing the User Control

In this part of the walkthrough, you will add the user control to a page.

To add the user control to the page

1. Add a new page to the Web site, or use an existing page.
2. Switch to Design view.
3. From Solution Explorer, drag the user control file (ListPicker.ascx) onto the page.
4. Switch to Source view.
   - You will see the following `@ Register` directive at the top of the page, which looks the same as it would for a user control written in a statically-compiled language.

   ```
   <%@ Register Src="ListPicker.ascx" TagName="ListPicker"
      TagPrefix="uc1" %>
   ```

   - Similarly, you will see an element for the user control, which again looks like any user control:

   ```
   <uc1:ListPicker id="ListPicker1" Runat="server" />
   ```

5. Press CTRL+F5 to run the page, and verify that the buttons work as expected.

Code Discussion

Your user control now works, but it is not yet useful as a general-purpose control. A more practical version of the user control would let you do the following:

- Specify whether duplicate values are allowed in the `TargetList` list, either in code or declaratively in the host page.
- Specify the list of items to display in the `SourceList` list dynamically.
- Get the list of items that the user selected in the `TargetList` list.
- Provide a way for users to clear all items in the `TargetList` list quickly.

Performing these tasks requires that the host page can communicate with the user control, both to share values (set and read properties) and to issue commands (call methods).

Adding a Property to the User Control

In this part of the walkthrough you will add an `AllowDuplicates` property to the user control, and set its value declaratively in the host page on the `ListPicker1` element. You will also modify the `ListPicker` code to prevent it from adding duplicate elements when the property is set to **true**. In a later part of the walkthrough, you will set the value of the property in code.

To add code to define a property

1. Switch to the code file, ListPicker.ascx.py.
2. Add the following accessor methods for the property. In this release, the accessors must be named Get<propertyname> and Set<propertyname>.

   Note The Python language has its own syntax for properties in classes, but in this release Python-style properties do not work in user controls.

   ```
   def GetAllowDuplicates():
       return bool(ViewState["allowDuplicates"])
   def SetAllowDuplicates(value):
       ViewState["allowDuplicates"] = bool(value)
   ```

   The value of the property must be saved explicitly in view state so that it persists between round trips. Because there is initially no entry in view state for "allowDuplicates", the default setting is **false**.

If a property is declared using the pattern described in this walkthrough, you can set it declaratively. In this procedure, you will set the **AllowDuplicates** property to an initial value of **true**.

To set user control properties declaratively

1. Switch to or open the test page that contains the instance of your user control.
2. In Source view, set **AllowDuplicates** declaratively by using syntax similar to the following:

   ```
   <uc1:ListPicker id="ListPicker1" Runat="server"
       AllowDuplicates="1" />
   ```

   The "Set" prefix is inferred, so that the `SetAllowDuplicates` property accessor is called.

   Notice the use of "1" instead of "true". For Boolean properties, use 1 for **true** and 0 for **false**. The Python language does not have a **true** keyword, and any non-zero result is interpreted as true. Thus, `AllowDuplicates="true"` and `AllowDuplicates="false"` both set the property to **true**.

Next you will modify the existing code in the user control to take advantage of the **AllowDuplicates** property setting.

To modify existing code to use the AllowDuplicates property

1. Find the `AddItem` method that you wrote in "Adding Code to Handle User Selections," earlier in this walkthrough, and replace the contents with the following highlighted code:

   ```
   def AddItem(li):
       TargetList.SelectedIndex = -1
       if GetAllowDuplicates():
           TargetList.Items.Add(li)
       else:
           if TargetList.Items.FindByText(li.Text) < 0:
               TargetList.Items.Add(li)
   ```

   The code performs the same function as before (adding an item to the `TargetList` list), but now the code checks to determine whether the **AllowDuplicate** property is set to **true**. If the **AllowDuplicate** property is set to **true**, the code first looks for an existing item with the same value as the proposed new item, and if no existing item is found, adds the new item.
2. Run the test page and verify that the user control still allows duplicates. That is, verify that the default behavior was overridden by setting the property declaratively.
3. Change the declarative property value to 0 and run the page again to verify that duplicates cannot be added.

Adding Methods to the User Control

In this part of the walkthrough, you will add methods to populate the `SourceList` list dynamically, to get the list of items that the user selected in the `TargetList` list, and to provide a way for users to clear all items in the `TargetList` list quickly.

To add code to define custom methods

1. Switch to the code file, ListPicker.ascx.py.
2. Add the following `GetSelectedItems` method:

   ```
   def GetSelectedItems():
       return TargetList.Items
   ```

   Note The Python language supports read-only properties on classes, but in this release you cannot add properties to a user control using Python property syntax.
3. Add the following `AddSourceItem` method to populate the source list:

   ```
   def AddSourceItem(sourceItem):
       SourceList.Items.Add(sourceItem)
   ```

4. Add the following `ClearAll` method to clear items in the target list:

   ```
   def ClearAll():
       SourceList.Items.Clear()
   ```

Testing the User Control Methods

The final task in this walkthrough is to enhance the host page to be able to share values with the user control programmatically, by setting and retrieving the properties and calling the methods. To illustrate how to work with the user control programmatically, you will add some controls to the host page.

Note Because you will be setting the contents of the `SourceList` list using the AddSourceItem method, you can remove the test data that you entered in "To create a user control", earlier in this walkthrough.

To work with the user control programmatically

1. Go to the test page and switch to Design view.
2. From the **Standard** group in the Toolbox, drag the following controls onto the table on the host page, and then set the properties as indicated.

Control	Properties
TextBox	**ID**: NewItem **Text**: **(empty)**
Button	**ID**: AddItem **Text**: **Add Item**
Button	**ID**: LoadFiles **Text**: **File List**
Button	**ID**: ClearSelection **Text**: **Clear All**
CheckBox	**AutoPostBack**: True **Checked**: True **ID**: AllowDuplicates **Text**: **Allow duplicates**
Button	**ID**: ShowSelection **Text**: **Show Selection**
Label	**ID**: **Selection** **Text**: (empty)

3. Switch to the code file, ListPicker.ascx.py.
4. Add the following event handler for the **CheckedChanged** event of the **Allow Duplicates** check box:

```
def AllowDuplicates_CheckedChanged(sender, args):
    ListPicker1.SetAllowDuplicates(AllowDuplicates.Checked)
```

   The code transfers the new setting of the check box to the `AllowDuplicates` property of the `ListPicker` control.

5. Add the following event handler for the **Click** event of the **Add Item** button:

```
def AddItem_Click(sender, args):
    ListPicker1.AddSourceItem(Server.HtmlEncode(NewItem.Text))
```

   The code adds the item in the **NewItem** textbox to the source list.

6. Add an event handler for the **Click** event of the `LoadFiles` button, and then add the following highlighted code:

```
def LoadFiles_Click(sender, args):
    path = Server.MapPath(Request.ApplicationPath)
    files = System.IO.Directory.GetFiles(path)
    for filename in files:
        ListPicker1.AddSourceItem(filename)
```

   The code populates the source list with a list of files from the Web site root directory.

7. Add an event handler for the **Click** event of the `ShowSelection` button:

```
def ShowSelection_Click(sender, args):
    selectedItemString = ""
    for lItem in ListPicker1.SelectedItems:
        selectedItemString += lItem + " "
    Selection.Text = selectedItemString
```

   The code concatenates the items that have been selected in the `TargetList` of the `ListPicker`, and displays them in the `Selection` **Label** control.

8. Add the following event handler for the **Click** event of the **Clear Selection** button:

```
def ClearSelection_Click(sender, args):
    ListPicker1.ClearAll()
```

   This code invokes the `ClearAll` method for the user control to remove all items from the `TargetList` list.

9. In the Web page, switch to **Source** view and bind the event handlers to the controls:

```
<asp:Button ID="AddItem" runat="server" Text="Add Item"
    OnClick="AddItem_Click" />
<asp:Button ID="LoadFiles" runat="server" Text="File List"
    OnClick="LoadFiles_Click" />
<asp:Button ID="ClearSelection" runat="server" Text="Clear All"
    OnClick="ClearSelection_Click" />
<asp:CheckBox ID="AllowDuplicates" runat="server"
    AutoPostBack="True" Checked="True" Text="Allow Duplicates"
        OnCheckedChanged="AllowDuplicates_CheckedChanged" />
<asp:Button ID="ShowSelection" runat="server" Text="Show Selection"
    OnClick="ShowSelection_Click" />
```

10. Press CTRL+F5 to run the page. Verify that the user control works as expected.

Next Steps

For more information on the features of user controls, see "ASP.NET User Controls Overview" in the MSDN Online Library.

See Also

Walkthrough: Creating a Basic Page in Visual Web Developer