Walkthrough: Debugging IronPython for ASP.NET

Introduction

Visual Studio provides you with tools to help find errors in your ASP.NET Web pages. In this walkthrough, you will work with the debugger, which allows you to step through the page's code line by line and examine the values of variables.

In the walkthrough, you will create a Web page that contains a simple calculator that squares a number. After creating the page (which will include a deliberate error), you will use the debugger to examine the page as it is running.

Tasks illustrated in this walkthrough include:

- Setting breakpoints.
- Invoking debugger from a Web page in a file system Web site.

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, with Visual Studio Integration or Visual Web Developer Integration, as appropriate

This walkthrough assumes that you have a general understanding of working in Visual Web Developer or Visual Studio. For an introduction, see Walkthrough: Creating a Basic Page in Visual Web Developer.

Creating the Web Site

In the first part of the walkthrough, you will create a page that you can debug.

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

To create a file system Web site and an ASP.NET Web page

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

   The **New Web Site** dialog box appears.
3. In the **Language** list, select **IronPython** as the default for your Web site.

   Note You can include statically compiled languages in the same Web application by creating pages and components in different programming languages.
4. Under **Visual Studio installed templates**, click **ASP.NET Web Site**.
5. In the **Location** box, click **File System** and then type the name of the folder where you want to keep the pages of your Web site.

   For example, type the folder name **C:\DebugWebSite**.
6. Click **OK**. Visual Studio creates the folder and a new page named Default.aspx.

Creating a Page to Debug

For this walkthrough, it is important that you create a new page as specified in the following procedure.

To add a page to the Web site

1. Close the Default.aspx page.
2. In Solution Explorer, right-click the name of your Web site (for example, **C:\DebugWebSite**) and choose **Add New Item**.
3. Under **Visual Studio installed templates**, choose **Web Form**.
4. In the **Name** box, enter **DebugPage.aspx**. The language defaults to **IronPython**.
5. Be sure that the **Place code in separate file** check box is checked.

   In this walkthrough, you are creating a page with the code in a separate file. The code for ASP.NET pages can be located either in the page or in a separate class file.

   Note In this release, there are small differences in the way breakpoints and watches can be set, depending on whether the code is in the page or in a separate file. The walkthrough calls out these differences..
6. Click **Add**.

Adding Controls and Code for Debugging

You can now add some controls to the page and then add code. The code will be simple, but enough to allow you to add breakpoints later.

To add controls and code for debugging

1. Switch to Design view, and then from the **Standard** folder of the **Toolbox**, drag the following controls onto the page and set their properties as indicated:

Control	Properties
Label	ID: **CaptionLabel** Text: (empty)
TextBox	ID: **NumberTextBox** Text: (empty)
Button	ID: **SquareButton** Text: Square
Label	ID: **ResultLabel** Text: (empty)

   Note For this walkthrough, the layout of the page is not important.

2. Right-click the page and click **View Code**.

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

3. Add an event handler for the button's **Click** event, with logic to call a function named **Square** to square the number entered by the user. The handler might look like the following example.

   Note The code example deliberately does not include error checking.

```
def SquareButton_Click(sender, args):
    number = float(NumberTextBox.Text)
    result = Square(number)
    ResultLabel.Text = '%s squared is %8.2f' % \
        (NumberTextBox.Text, result)
```

4. Switch to DebugPage.aspx and select Source view.
5. Bind the event handler to the `SquareButton` control by adding an `OnClick` attribute, as shown in the following example.

```
<asp:Button ID="SquareButton" runat="server" Text="Square"
    OnClick="SquareButton_Click" />
```

6. Right-click the page and click **View Code** to return to the code file
7. Create the `Square` function that squares the number. Include a bug in the code that adds the number to itself instead of multiplying it.

   The code might look like the following example.

```
def Square(number):
    return number + number
```

8. In the **Page_Load** method, add the parameters `sender` and `args` to the **Page_Load** event handler.

```
def Page_Load(sender, args):
    pass
```

   In IronPython, you can omit the arguments of the event handler. However, they are useful for debugging.
9. Above the **pass** statement, add code to set the text of the `CaptionLabel` control to **Enter a number:** if this is the first time the page is running, and thereafter to **Enter another number:**. The handler will look like the following.

```
def Page_Load(sender, args):
    postback = sender.IsPostBack
    if IsPostBack:
        CaptionLabel.Text = "Enter another number: "
    else:
        CaptionLabel.Text = "Enter a number: "
    pass
```

   You can use the **IsPostBack** property by itself, as shown in the **if** statement, or qualified by `sender`, as shown in the assignment statement. When it is used by itself, it is recognized as a property of the script page just as it is in C# and Visual Basic. In the assignment statement, `sender.IsPostBack` is used because later in the walkthrough it illustrates a limitation of debugging in the current release.

   Note In this case, `sender` and the implicit page reference happen to be the same, because **Page_Load** handles a page event.

   The Python **pass** statement is a placeholder that does nothing when executed. Normally, you would not use it in an actual method body, but in this walkthrough it helps demonstrate stepping through the **if** statement.

10. Save the page.
11. Press CTRL+F5 to run the page without debugging.
12. Enter a number (other than 2) and press the **Square** button.

    Note that the result is incorrect, because there is an intentional bug in the program.
13. Close the browser.

Debugging the Page

In this part of the walkthrough, you will use the debugger to examine the page code line by line as it is running, add breakpoints to the code, and then run the page in debug mode. You will start by setting breakpoints.

To set breakpoints

1. In **Source** view, set a breakpoint on the following line.

   ```
   postback = sender.IsPostBack
   ```

   Note When your code is in a separate file, you can toggle breakpoints by pressing F9, by right-clicking a line and choosing **Breakpoint**, or by clicking in the margin to left of the line. In this release, the only mechanism that works for code embedded in a Web page is clicking in the margin.
2. Set another breakpoint on the following line of the `SquareButton_Click` handler:

   ```
   result = Square(number)
   ```

You are now ready to run the debugger.

To run the debugger

1. From the **Debug** menu, choose **Start Debugging** (or press F5) to run the page in debug mode.

   Note The first time you debug, you will be prompted to modify the Web.config file to enable debugging. Debugging is disabled by default, for better performance.

   Because the breakpoint is in the **Page_Load** event handler, the page has not finished processing yet. The browser is open, but the page is not yet displayed.
2. Click the `postback` variable and press Shift+F9 to display its value in a **Quick Watch** window. The value is **null**.
3. Click `sender` and press Shift+F9 to display its value, which is the current script page. You can expand `sender`, then expand `base`, and view the properties of the page. Scroll down and note that the value of the **IsPostBack** property is **false**.
4. Click **IsPostBack** and press Shift+F9. The value cannot be displayed. Like all Python variables, `sender` is dynamically typed, and thus the properties of the object it contains cannot be resolved by the debugger.
5. Press F10 to execute the assignment statement, and check the value of `postback` to see that it is now **false**.
6. In the **Debug** menu, click **Windows** and then click **Locals**.

   This opens the **Locals** window, which displays the values of variables and objects that are in scope at the current line being executed. The value of `postback` is **false**. You can expand `sender`, and under `sender` expand `base`, to view the properties of the script page.

   Notice that `IsPostBack`, `CaptionLabel`, and `SquareButton` do not appear in the **Locals** window.

7. Press F10 to execute the **if** statement.

   The line of code after the **if** statement is highlighted, even though `IsPostBack` is **false**. The reason is that the debugger highlights the end of a skipped block before continuing to the next line of code to be executed. Python does not have **End** statements or curly braces to mark the end of a block, so the debugger pauses on the last line of code in the skipped block. When there is only one line of code in the block, it can be hard to determine which branch was executed. During development, you can add a **pass** statement to the end of single-line **if** and **else** blocks.
8. Press F10 and observe that the line of code after the **else** statement is now highlighted.
9. Press F10 again to execute the line of code.
10. In the **Immediate** window, use the question mark operator to examine the value of `postback`.

    The **Immediate** window display will look like the following.

    ```
    >? postback
    >false
    ```

    In the current release, the **Immediate** window readily displays variable values and simple computations, but it is not useful for examining object properties because variables are dynamically typed. In order to examine the **Text** property of the `CaptionLabel` control, for example, you would have to cast both `sender` and `CaptionLabel` to explicit types, using an expression like the following:

    ```
    ? ((Label)((ScriptPage)sender).FindControl("CaptionLabel")).Text
    ```
11. In the **Debug** menu, click **Windows**, click **Watch**, and then click **Watch 1**.

    Note If you are using Visual Studio Express Edition, the debugger offers only a single **Watch** window.
12. Right-click `postback`, then click **Add Watch** on the context menu to add the `postback` variable to the watch.

    Note In the current release, you cannot right-click to add a watch if the code is in the page instead of in a separate file. Instead, you can enter the name of the variable in the first cell of the **Name** column in the **Watch** window.

    In the current release, the **Watch** window is subject to the same limitations as the **Immediate** window. If you want to display the properties of variables containing objects, you must cast them to specific types.
13. Press F5 to continue execution and display the page.
14. Enter the value `7` into the text box and click the **Square** button.

    The debugger is displayed again, with the breakpoint in the `Page_Load` event handler. Press F10 to execute the line. The **Watch** window shows that the value of `postback` is **true**.
15. Press F5 to continue.

    The debugger processes the **Page_Load** event handler and enters the `SquareButton_Click` handler, where it stops on the second breakpoint you set.
16. Press F11 to step into the `Square` function.

    Note In the current release, pressing F11 sometimes displays a message that there is no source code available for the current location. In this case, you must set a breakpoint inside the function you want to enter, and then press F5 to continue to that breakpoint.

17. Continue stepping through the function until you return from it.

    The value of `result` still is not set.

18. Press F11 one more time, and note the incorrect value.

    In the current release, you have to stop the debugger in order to correct the code.

19. Press F5 to continue running the program.

See Also

Walkthrough: Creating a Basic Page in Visual Web Developer