Tutorial – UI Automation Framework

Tu, Pin-Ying
Ver. 1.0, 21st September, 2010

Contents

Contents .........................................................................................................................1
About This Tutorial .........................................................................................................1
Tutorial – Coded UI Testing ............................................................................................2
Appendix A – System Requirement ..............................................................................7
Appendix B – Accessible Names ....................................................................................7
Appendix C – Deployment for GUI Testing .....................................................................8
Appendix D – Modification for Windows 7 ...................................................................9

About This Tutorial

Unit test framework is very useful for testing programs. In this tutorial, we will introduce automatic acceptance testing. To do acceptance testing, we need to use programs to “simulate” user’s actions and validate system responses. A manual approach requires a number of testers to operate the system and look at the results shown on the screen. It works, but is also very expensive. Announced with Visual Studio 2010, “UI Automation Framework” provides a method to do automatic GUI (acceptance) testing (see Appendix A – System Requirement). This tutorial introduces how to use the UI Automation Framework to test the calculator of Windows XP (for Windows 7, please see “Appendix D – Modifications for Windows 7”).
Tutorial – Coded UI Testing

Step 1 Create a Test Project

Open Visual Studio 2010 and then choose File → New → Project. In the dialog, choose Visual C# → Test → Test Project. Key in the project name and click OK button to create a test project.

Step 2 Add Coded UI Test

By default, the wizard loads unit testing framework references and creates a simple test class. Remove the class. Click right mouse button on the project in the Solution Explorer (Figure 1). In the context menu, choose Add → Coded UI Test. The wizard loads UI Automation Framework references, creates another test class, and then pops up a dialog (Figure 2). Click Cancel button, the usage of this dialog will be introduced in another tutorial.

Figure 1 Add a coded UI test

Figure 2 Generate Code for Coded UI Test Dialog
Step 3 Modify the Class

Step 2 loads necessary private (reserved for wizards) references, but the codes generated by the wizard are not needed. Replace all with the following codes.

```csharp

namespace GUITestTutorial {

    [CodedUITest]
    public class WinXPCalculatorTest {

        private const string FILE_PATH = "C:\Windows\system32\calc.exe";
        private const string MSAA_TECHNOLOGY = "MSAA";
        private const string WINDOW_TYPE = "Window";
        private const string BUTTON_TYPE = "Button";
        private const string TEXT_TYPE = "Edit";
        private const string CALCULATOR_TITLE = "小算盤";
        private const string TEXT_PROPERTY = "Text";
        private const string EXPECTED_VALUE = "444. ";

        private ApplicationUnderTest _aut;
        private UITestControl _window;
    }
}
```

Step 4 Launch the Calculator

Add two member methods Initialize() and FindWindow() with following codes to launch the calculator. You have to modify the value of FILE_PATH constant if the calculator does not exist at the specific location.

```csharp
/// <summary>
/// Launches the Calculator
/// </summary>
///
[TestInitialize()]
public void Initialize() {
    _aut = ApplicationUnderTest.Launch(FILE_PATH);
    _window = FindWindow(CALCULATOR_TITLE);
}
```
UI Automation Framework uses SearchProperties to filter GUI controls on the screen. Three properties are required: TechnologyName, ControlType, and Name to find a specific control. The Find() will throw UITestControlNotFoundException that cause the test to fail if the window doesn’t exist.

```csharp
private UITestControl FindWindow(string title) {
    UITestControl window = new UITestControl(_aut);
    PropertyExpressionCollection properties = new PropertyExpressionCollection();
    properties.Add(UITestControl.PropertyNames.TechnologyName, MSAA_TECHNOLOGY);
    properties.Add(UITestControl.PropertyNames.ControlType, WINDOW_TYPE);
    properties.Add(UITestControl.PropertyNames.Name, title);
    window.SearchProperties.AddRange(properties);
    window.Find();
    return window;
}
```

**Step 5 Close the Calculator**

In the unit testing tutorial, there is no Cleanup() method, because garbage collection is performed automatically after finishing tests. But in this tutorial, this method is required to close the launched calculator. Add a member method Cleanup() method with the following codes.

```csharp
[TestMethod]
public void Cleanup() {
    _aut.Close();
}
```

**Step 6 Click Buttons**

UI Automation Framework offers Mouse and Keyboard classes that can simulate the user’s actions. Add a member method ClickButton() with the following codes which moves the cursor to the specific button and clicks on it. In this case, the name is the same as the text displayed on the button. However, the name is not always the same as the text. Please see Appendix B – Accessible Names for more details.
/// <summary>
/// Finds and clicks on the specific button by its name
/// </summary>
/// <param name="name">The name of the button</param>
private void ClickButton(string name) {
    UITestControl button = new UITestControl(_aut);
    PropertyExpressionCollection properties = new PropertyExpressionCollection();
    properties.Add(UITestControl.PropertyNames.TechnologyName, MSAA_TECHNOLOGY);
    properties.Add(UITestControl.PropertyNames.ControlType, BUTTON_TYPE);
    properties.Add(UITestControl.PropertyNames.Name, name);
    button.SearchProperties.AddRange(properties);
    button.Find();
    Mouse.Click(button);
}

Step 7 Write a Test Script

Now, use the ClickButton() method to write a test script. Add a new member method RunScriptAdd() with the following codes. This method automatically clicks buttons just like that a user is using the calculator.

/// <summary>
/// Runs the script: 123 + 321 =
/// </summary>
private void RunScriptAdd() {
    ClickButton("C");
    ClickButton("1");
    ClickButton("2");
    ClickButton("3");
    ClickButton("+");
    ClickButton("3");
    ClickButton("2");
    ClickButton("1");
    ClickButton("=");
}

Step 8 Assert the Result

In order to make sure that the result is correct, add a member method GetResult() with the following codes to get the calculated result. Note that there is no Name search property because the calculator didn’t assign the text box an accessible name. This is not a good design for testability.
However, in this case, the calculator only has one text box so the method works. If there are more than one text box, this method may fail. Therefore, please make sure that every control you want to test has an accessible name in homework.

```csharp
///<summary>
/// Gets the result in the calculator
///</summary>
///<returns>The actual result</returns>
private string GetResult() {
    UITestControl text = new UITestControl(_aut);
    PropertyExpressionCollection properties = new PropertyExpressionCollection();
    properties.Add(UITestControl.PropertyNames.TechnologyName, MSAA_TECHNOLOGY);
    properties.Add(UITestControl.PropertyNames.ControlType, TEXT_TYPE);
    text.SearchProperties.AddRange(properties);
    text.Find();
    return text.GetProperty(TEXT_PROPERTY).ToString();
}

Now, add a member method with [TestMethod] annotation as follows:

```csharp
///<summary>
/// Tests that the result of 123 + 321 should be 444
///</summary>
[TestMethod]
public void TestAdd() {
    RunScriptAdd();
    Assert.AreEqual(EXPECTED_VALUE, GetResult());
}
```

Step 9 Run!!

Click “Run Tests in Current Context” button in the toolbar. While the test is running, please do not move your mouse or press any key. Any event generated manually will cause the test failed. Design your actions sequence for practice if you have time after finishing the tutorial, or leave the practice in your homework.

-- The End --
Appendix A – System Requirement

Not every version of Visual Studio provides UI Automation Framework. The computers in the CSIE Lab have been installed with Visual Studio 2010 Ultimate version. If you need to do your homework at home, you need to install either the Ultimate or Premium version.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Professional</th>
<th>Premium</th>
<th>Ultimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Testing</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>UI Automation Testing</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

Figure 3 Comparison of different Visual Studio versions

Appendix B – Accessible Names

How were the “names” used in the tutorial obtained? Programmers! For better testability, the programmer has to assign Control.AccessibleName property value during programming. Note that the name of each testable control must be unique! The name can be assigned by using the Windows Form Designer (Figure 2) or by coding. However, there are some exceptions. For example, the accessible name of tabs (not tab pages) in TabControl is assigned at runtime and its name is the same as the displayed text. Since a name may be different when a program is running on an OS with a different language, your design of test codes needs the ability to access the displayed text for different languages.

Figure 4 AccessibleName property
Appendix C – Deployment for GUI Testing

Since different computers use different paths, testing a program (project) developed under the same solution requires a special treatment when the program is launched. First, please check the project dependency. As shown in Figure 5, the test project “UniversityGUITest” depends on the production project “University.” That means University will always be built before UniversityGUITest. Second, double click on the “local.testsettings” file. In the Deployment page (Figure 6), enable the deployment option and click “Add File” button and choose the executable file. Every time, before running tests, the last executable file will be copied to the test working directory. Thus, the tester can always use a relative path to launch the program.

![Figure 5 Project Dependency](image)

![Figure 6 Deployment](image)
Appendix D – Modifications for Windows 7

If you do this tutorial in Windows 7 (Traditional Chinese version), the test case will fail, because Win7’s calculator is totally different from Windows XP’s calculator. You need to modify the following constants:

```csharp
private const string TEXT_TYPE = "Text";
private const string RESULT_CONTROL_NAME = "結果"; // Add for Windows 7
private const string TEXT_PROPERTY = "DisplayText";
private const string EXPECTED_VALUE = "444";
```

In addition, modify `GetResult()` and `RunScriptAdd()` methods.

```csharp
private string GetResult() {
    UITestControl text = new UITestControl(_aut);
    PropertyExpressionCollection properties = new PropertyExpressionCollection();
    properties.Add(UITestControl.PropertyNames.TechnologyName, MSAA_TECHNOLOGY);
    properties.Add(UITestControl.PropertyNames.ControlType, TEXT_TYPE);
    properties.Add(UITestControl.PropertyNames.Name, RESULT_CONTROL_NAME);
    text.SearchProperties.AddRange(properties);
    text.Find();
    return text.GetProperty(TEXT_PROPERTY).ToString();
}

private void RunScriptAdd() {
    ClickButton("清除");
    ClickButton("1");
    ClickButton("2");
    ClickButton("3");
    ClickButton("加");
    ClickButton("3");
    ClickButton("2");
    ClickButton("1");
    ClickButton("等於");
}