**AutomationElement**

Each piece of user interface in an application corresponds to an AutomationElement in the UI Automation Tree, and the UI Automation Tree is just a tree of AutomationElements that has the Desktop window as the root.

To traverse the tree you can use the ‘FindFirst’ and ‘FindAll’ methods from AutomationElement.  They allow you to specify a TreeScope that determines where it will search (some options are Descendents, Ancestors, etc.), and a PropertyCondition.

So for instance to find the first element with the name “play” in the descendents of AutomationElement ‘e’ you would do this:

|  |  |
| --- | --- |
|  | AutomationElement play = e.FindFirst(TreeScope.Descendents,  new PropertyCondition(AutomationElement.NameProperty, "play")); |

In order to actually get AutomationElements to do something you have to get their Control Pattern.

Controls are not limited to a single Control Pattern.  For instance, the ComboBox exposes the ExpandCollapsePattern, SelectionPattern, and ValuePattern depending on what state it is in.

Here you’ll find an extensive list of the various types of Controls and the Control Patterns they support: http://msdn.microsoft.com/en-us/library/ms750574.aspx

Button is the first one in that list and it exposes the InvokePattern conditionally.

|  |  |
| --- | --- |
| 1 | InvokePattern playPattern = play.GetCurrentPattern(InvokePattern.Pattern) as InvokePattern; |

Now all we need to do is actually invoke it:

|  |  |
| --- | --- |
| 1 | playPattern.Invoke(); |

That pushes the button.  The code uses helper methods I wrote to simplify things:

– finds the first AutomationElement in the descendents of ‘scope’ with AutomationId == id

|  |  |
| --- | --- |
| 1 | FindElementWithId(AutomationElement scope, string id) |

– finds the first AutomationElement in the descendents of ‘scope’ with Name == name

|  |  |
| --- | --- |
| 1 | FindElementWithName(AutomationElement scope, string name) |

– finds all AutomationElements in the descendents of ‘scope’ with ClassName == className

|  |  |
| --- | --- |
| 1 | FindAllElementsWithClassName(AutomationElement scope, string className) |

**Button**

|  |  |
| --- | --- |
| 1  2  3 | var button = FindElementWithId(scope, id);  InvokePattern p = button.GetCurrentPattern(InvokePattern.Pattern) as InvokePattern;  p.Invoke(); |

**CheckBox**

|  |  |
| --- | --- |
| 1  2  3 | var checkbox = FindElementWithId(scope, id);  TogglePattern p = checkbox.GetCurrentPattern(TogglePattern.Pattern) as TogglePattern;  p.Toggle(); |

**ComboBox**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var combo = FindElementWithId(scope, id);  ExpandCollapsePattern p = combo.GetCurrentPattern(ExpandCollapsePattern.Pattern) as ExpandCollapsePattern;  p.Expand();  var items = FindAllElementsWithClassName(combo, "ListBoxItem");  if (index &gt;= 0 &amp;&amp; index &lt; items.Count)  {  SelectionItemPattern itemp = items[index].GetCurrentPattern(SelectionItemPattern.Pattern) as  SelectionItemPattern;  itemp.Select();  }  p.Collapse(); |

**TextBox**

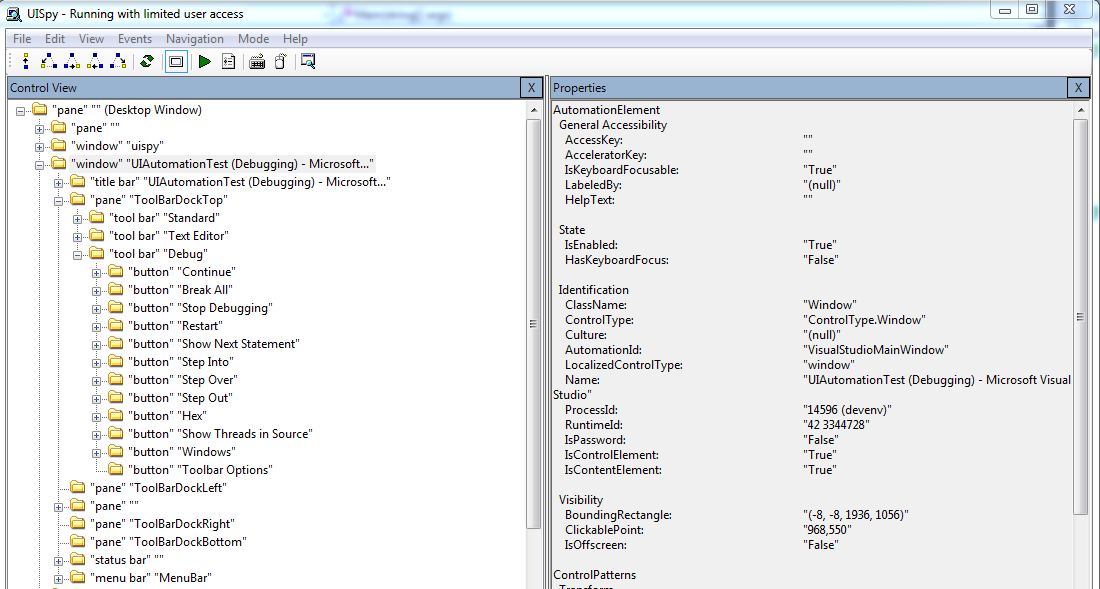
|  |  |
| --- | --- |
| 1  2  3 | var textbox = FindElementWithId(scope, id);  ValuePattern p = textbox.GetCurrentPattern(ValuePattern.Pattern) as ValuePattern;  p.SetValue(value); |

**RadioButton**

|  |  |
| --- | --- |
| 1  2  3 | var radiobutton = FindElementWithId(scope, id);  SelectionItemPattern p = radiobutton.GetCurrentPattern(SelectionItemPattern.Pattern) as SelectionItemPattern;  p.Select(); |

**Example**

As a simple complete example I’ll write a program that debugs itself via Visual Studio 2010.  First thing we need to do is go into UISpy:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | static void Main(string[] args)  {  Thread.Sleep(500);  var toolbar = AutomationHelper.FindFirstElementWithName(AutomationElement.RootElement, "ToolBarDockTop");  var breakall = AutomationHelper.FindFirstElementWithName(toolbar, "Break All");  AutomationHelper.PushButton(breakall);  Thread.Sleep(1000);  } |

So I’ve made a new command line application called “UI Automation Test” that just has this Main.

1. Thread.Sleep() Visual Studio has time to populate the toolbar with all the buttons after we start debugging.
2. Just to establish some scope so that the searching of the UI Automation Tree doesn’t take as long we’ll get the AutomationElement for the pane called “**ToolBarDockTop**”… this is the container that houses all the buttons at the top of VS2010.
3. Next I used the toolbar element as the scope for the search for the ‘Break All’ button.
4. Finally just push the button.
5. The Thread.Sleep(1000) just gives it a line to break on… otherwise it will say that it can’t pause execution.