The managed way to retrieve text under the cursor (mouse pointer)

devblogs.microsoft.com/oldnewthing/20130408-00

April 8, 2013

Raymond Chen

Today’s Little Program is a managed version of the text-extraction program from several years ago. It turns out that it’s pretty easy in managed code because the accessibility folks sat down and wrote a whole framework for you, known as UI Automation.

(Some people are under the mistaken impression that UI Automation works only for extracting data from applications written in managed code. That is not true. Native code can also be a UI Automation provider. The confusion arises because the name *UI Automation* is used both for the underlying native technology as well as for the managed wrappers.)

```
using System;
using System.Windows;
using System.Windows.Forms;
using System.Windows.Automation;


class Program
{
 static Point MousePos {
  get { var pos = Control.MousePosition;
        return new Point(pos.X, pos.Y); }
 }


 public static void Main()
 {
  for (;;) {
   AutomationElement e = AutomationElement.FromPoint(MousePos);
   if (e != null) {
    Console.WriteLine(“Name: {0}”,
     e.GetCurrentPropertyValue(AutomationElement.NameProperty));
    Console.WriteLine(“Value: {0}”,
     e.GetCurrentPropertyValue(ValuePattern.ValueProperty));
    Console.WriteLine();
   }
   System.Threading.Thread.Sleep(1000);
  }
 }
}
```

We use the `FromPoint` method to locate the automation element under the current mouse position and print its name and value.

Well that was pretty simple. I may as well do something a little more challenging. Since the feature is known as UI *Automation*, I’ll try automating the *Run* dialog by programmatically entering some text and then clicking OK.

```
using System.Windows.Automation;


class Program
{
 static AutomationElement FindById(AutomationElement root, string id)
 {
  return root.FindFirst(TreeScope.Children,
   new PropertyCondition(AutomationElement.AutomationIdProperty, id));
 }


 public static void Main()
 {
  var runDialog = AutomationElement.RootElement.FindFirst(
   TreeScope.Children,
   new PropertyCondition(AutomationElement.NameProperty, “Run”));
  if (runDialog == null) return;


  var commandBox = FindById(runDialog, “12298”);
  var valuePattern = commandBox.GetCurrentPattern(ValuePattern.Pattern)
                     as ValuePattern;
  valuePattern.SetValue(“calc”);


  var okButton = FindById(runDialog, “1”);
  var invokePattern = okButton.GetCurrentPattern(InvokePattern.Pattern)
                     as InvokePattern;
  invokePattern.Invoke();
 }
}
```

The program starts by looking for a window named *Run* by performing a children search on the root element for an element whose *Name* property is equal to `“Run”`.

Assuming it finds it, the program looks for a child element whose automation ID is `“12298”`. How did I know that was the automation ID to use? The documentation for UI Automation suggests using a tool like UI Spy to look up the automation IDs.

Mind you, since I am automating something outside my control, I have to accept that the automation ID may change in future versions of Windows. (It’s not like they check with me before making changes.) But this is a Little Program, not a production-level program, so that’s a limitation I will accept, since I’m the only person who’s going to use this program, and if it stops working, I know who to talk to (namely, me).

Anyway, afer we find the command box, I ask for its Value pattern. Automation elements can support *patterns* which expose additional properties and methods specific to particular uses. In our case, the Value pattern lets us get and set the value of an editable object, so we use the

`SetValue` method to set the text in the Run dialog to `calc`.

Next, we look for the OK button, which UI Spy told me had automation ID 1. We ask for the Invoke pattern on the button and then call the `Invoke` method. The Invoke pattern is the pattern for objects that do just one thing, and `Invoke` means "Do that thing that you do."

Open the *Run* dialog and run this program. It should programmatically set the command line to `calc`, then click OK. Hopefully, this will run the Calculator.

Just for fun, here's another program that just dumps the automation properties and patterns for whatever object is under the mouse cursor:

```
using System;
using System.Windows;
using System.Windows.Forms;
using System.Windows.Automation;


class Program
{
 static Point MousePos {
  get { var pos = Control.MousePosition;
        return new Point(pos.X, pos.Y); }
 }


 public static void Main()
 {
  for (;;) {
   AutomationElement e = AutomationElement.FromPoint(MousePos);
   if (e != null) {
    foreach (var prop in e.GetSupportedProperties()) {
     object o = e.GetCurrentPropertyValue(prop);
     if (o != null) {
      var s = o.ToString();
      if (s != “”) {
       var id = o as AutomationIdentifier;
       if (id != null) s = id.ProgrammaticName;
       Console.WriteLine(“{0}: {1}”, Automation.PropertyName(prop), s);
      }
     }
    }
    foreach (var pattern in e.GetSupportedPatterns()) {
     Console.WriteLine(“Pattern: {0}”, Automation.PatternName(pattern));
    }
    Console.WriteLine();
   }
   System.Threading.Thread.Sleep(1000);
  }
 }
}
```

Raymond Chen

Follow