Adventures in automation: Dismissing all message boxes from a particular application but only if they say that the operation succeeded

devblogs.microsoft.com/oldnewthing/20140303-00

March 3, 2014

Raymond Chen

Suppose you have a program that is part of your workflow, and it has the annoying habit of showing a message box when it is finished. You want to automate this workflow, and part of that automation is dismissing the message box.

Let's start by writing the annoying program:

```
#include <windows.h>
int WINAPI WinMain(
    HINSTANCE hinst, HINSTANCE hinstPrev,
    LPSTR lpCmdLine, int nCmdShow)
{
  Sleep(5000);
  MessageBox(nullptr, GetTickCount() % 1000 < 800 ?
             "Succeeded!" : "Failed!", "Annoying", MB_OK);
  return 0;
}
```

This annoying program pretends to do work for a little while, and then displays a message box saying whether or not it succeeded. (Let's say it succeeds 80% of the time.)

Our Little Program will automate this task and respond based on whether the operation succeeded. This is just a small extension of our previous program which logs the contents of every message box, except we are paying attention only to one specific message box.

To the rescue: UI Automation.

```
using System.Windows.Automation;
using System.Diagnostics;
using System.Threading;
class Program
{
 [System.STAThread]
 public static void Main(string[] args)
 {
  int processId = 0;
  bool succeeded = false;
  var resultReady = new ManualResetEvent(false);
  Automation.AddAutomationEventHandler(
   WindowPattern.WindowOpenedEvent,
   AutomationElement.RootElement,
   TreeScope.Children,
   (sender, e) =>
   {
    var element = sender as AutomationElement;
    if ((int)element.GetCurrentPropertyValue(
              AutomationElement.ProcessIdProperty) != processId)
    {
     return;
    }
    var text = element.FindFirst(TreeScope.Children,
      new PropertyCondition(AutomationElement.AutomationIdProperty,
                            "65535"));
    if (text != null && text.Current.Name == "Succeeded!")
    {
     succeeded = true;
     var okButton = element.FindFirst(TreeScope.Children,
       new PropertyCondition(AutomationElement.AutomationIdProperty,
                             "2"));
     var invokePattern = okButton.GetCurrentPattern(
       InvokePattern.Pattern) as InvokePattern;
     invokePattern.Invoke();
    }
    resultReady.Set();
   });
  // Start the annoying process
  Process p = Process.Start("annoying.exe");
  processId = p.Id;
  // Wait for the result
  resultReady.WaitOne();
  if (succeeded)
  {
   Process.Start("calc.exe");
  }
  Automation.RemoveAllEventHandlers();
 }
}
```

Most of this program you've seen before.

We register an automation event handler for new window creation that ignores windows that belong to processes we don't care about. That keeps us from being faked out by windows that happen to be created while our annoying task is running.

For simplicity's sake, I've removed other sanity checks which verify that the window which appeared is the one we actually care about. In real life, we might check the window class or the window title. I left that out because it's not really relevant to the story.

Once we think we have the window we want, we suck out the text so we can see whether the message was a success or failure message. If it was a success, we dismiss the dialog box by pushing the OK button; otherwise we leave the error message on the screen so the user can see what happened. Either way, we signal the main thread that we have a result.

After registering the event handler, we run the annoying process, tell the event handler the process ID, and wait for the signal. Once the signal arrives, we see whether it declared the operation a success, and if so, we proceed to the next step of, um, say, launching the calculator. (I just picked something arbitrary.)

Raymond Chen

Follow