Enumerating notification icons via UI Automation

devblogs.microsoft.com/oldnewthing/20141013-00

October 13, 2014

Raymond Chen

Today's Little Program uses accessibility to enumerate the current notification icons (and possibly click on one of them). This could be done manually via `IAccessible`, but the BCL folks conveniently created the `System.Windows.Automation` namespace which contains classes that take a lot of the grunt work out of walking the accessibility tree.

While it's true that the `System.Windows.Automation` namespace takes a lot of the grunt work out of accessibility, it is still rather verbose, so I'm going to start with some helper functions. They're all one-liners since they simply pass their parameters through with a little bit of extra typing.

```
using System.Collections.Generic;
using System.Windows.Automation;
using System.Linq;
static class AutomationElementHelpers
{
 public static AutomationElement
 Find(this AutomationElement root, string name)
 {
  return root.FindFirst(
   TreeScope.Descendants,
   new PropertyCondition(AutomationElement.NameProperty, name));
 }
```

The `AutomationElement.Find` extension method searches for a descendant of an accessible element with a particular name.

```
 public static IEnumerable<AutomationElement>
 EnumChildButtons(this AutomationElement parent)
 {
  return parent == null ? Enumerable.Empty<AutomationElement>()
                        : parent.FindAll(TreeScope.Children,
    new PropertyCondition(AutomationElement.ControlTypeProperty,
                          ControlType.Button)).Cast<AutomationElement>();
 }
```

The `AutomationElement.EnumChildButtons` extension method enumerates the button controls which are immediate children of a parent element.

```
 public static bool
 InvokeButton(this AutomationElement button)
 {
  var invokePattern = button.GetCurrentPattern(InvokePattern.Pattern)
                      as InvokePattern;
  if (invokePattern != null) {
   invokePattern.Invoke();
  }
  return invokePattern != null;
 }
}
```

The `AutomationElement.InvokeButton` extension method checks if the element is invokable (which buttons are), and if so invokes its default action.

Okay, given those helpers, we can write the actual enumerator.

```
class Program {
 public static IEnumerable<AutomationElement> EnumNotificationIcons()
 {
  foreach (var button in AutomationElement.RootElement.Find(
                 "User Promoted Notification Area").EnumChildButtons()) {
   yield return button;
  }
  foreach (var button in AutomationElement.RootElement.Find(
               "System Promoted Notification Area").EnumChildButtons()) {
   yield return button;
  }
  var chevron = AutomationElement.RootElement.Find("Notification Chevron");
  if (chevron != null && chevron.InvokeButton()) {
   foreach (var button in AutomationElement.RootElement.Find(
                      "Overflow Notification Area").EnumChildButtons()) {
    yield return button;
   }
  }
 }
```

Okay, here's what's going on.

First, we enumerate all the buttons that are children of an object called *User Promoted Notification Area*.

Next, we enumerate all the buttons that are children of an object called *System Promoted Notification Area*. This object is usually empty, but it may contain an icon if a demoted icon is temporarily promoted because it is showing a balloon.

Finally, if we are asked to enumerate hidden icons, we also find the *Notification Chevron* button and push it. That pops up a dialog called *Overflow Notification Area*, and we enumerate all the buttons from that dialog as well.

Okay, let's take this function out for a spin.

```
 public static void Main()
 {
  foreach (var icon in EnumNotificationIcons())
  {
   var name = icon.GetCurrentPropertyValue(AutomationElement.NameProperty)
              as string;
   System.Console.WriteLine(name);
   System.Console.WriteLine("---");
  }
 }
}
```

When you run this program, it should print the names of all the icons in your notification area, including the hidden ones (for which it needs to open the overflow dialog).

You may have noticed that it takes a long time to generate the icons in the *System Promoted Notification Area*; that's because the accessibility system is going crazy looking for something that usually doesn't exist. Let's speed things up by reducing the scope of the search. Once we find the *User Promoted Notification Area*, we will search for the *System Promoted Notification Area* inside the same window. That should save a lot of time.

```
// in static class AutomationElementHelpers
 static public AutomationElement
 GetTopLevelElement(this AutomationElement element)
 {
  AutomationElement parent;
  while ((parent = TreeWalker.ControlViewWalker.GetParent(element)) !=
       AutomationElement.RootElement) {
   element = parent;
  }
  return element;
 }
```

The `AutomationElement.GetTopLevelElement` extension method walks up the control view and returns the ancestor element that is a direct child of the root.

```
public static IEnumerable<AutomationElement> EnumNotificationIcons()
{
 var userArea = AutomationElement.RootElement.Find(
                 "User Promoted Notification Area");
 if (userArea != null) {
  foreach (var button in userArea.EnumChildButtons()) {
   yield return button;
  }
  foreach (var button in userArea.GetTopLevelElement().Find(
                "System Promoted Notification Area").EnumChildButtons()) {
    yield return button;
  }
 }
 var chevron = AutomationElement.RootElement.Find("Notification Chevron");
 if (chevron != null && chevron.InvokeButton()) {
  foreach (var button in AutomationElement.RootElement.Find(
                     "Overflow Notification Area").EnumChildButtons()) {
   yield return button;
  }
 }
}
```

Of course, what's the point of enumerating the icons if you can't also click them? Let's go look for the volume control icon and click it.

```
public static void Main()
{
 foreach (var icon in EnumNotificationIcons())
 {
  var name = icon.GetCurrentPropertyValue(AutomationElement.NameProperty) as string;
  if (name.StartsWith("Speakers:")) {
   icon.InvokeButton();
   break;
  }
 }
}
```

So far, this program relies on the accessible tree structure used by Windows 7 and Windows 8. If you run the program on earlier versions of Windows (or later ones), it may not work. That's because the accessible tree changed. The accessibiliy tree is not part of the API. It's something that is exposed for accessibility purposes to the end user, and the fact that there is a programmatic interface to it is an artifact of the accessibility.

In Windows Vista, there is not a separate overflow area for notification icons. Instead, you use the chevron button to expand or contract the notification area. Let's tweak our program to work on Windows Vista instead of Windows 7:

```
public static IEnumerable<AutomationElement> EnumNotificationIcons()
{
 var userArea = AutomationElement.RootElement.Find("Notification Area");
 if (userArea != null) {
  // If there is a chevron, click it. There may not be a chevron if no
  // icons are hidden.
  var chevron = userArea.GetTopLevelElement().Find("NotificationChevron");
  if (chevron != null) {
   chevron.InvokeButton();
  }
  foreach (var button in userArea.EnumChildButtons()) {
   yield return button;
  }
 }
}
```

The name of the notification area in Windows Vista is simply *Notification Area*, and the name of the chevron is *NotificationChevron* with no space. (Somebody was apparently trying to save two bytes.)

Okay, now that you see the general idea, I'll leave Windows XP, Windows 2000, and Windows NT support as an exercise.

Raymond Chen

Follow