Why can’t FindWindowEx find another program’s window by name?

devblogs.microsoft.com/oldnewthing/20180511-00

May 11, 2018

Raymond Chen

A customer had a test app that did something. They then wanted to write a test harness for their test app. The idea is that the test harness launches the test app, finds the test app’s *Start* button, waits for it to become enabled, and then presses it, and then does some more stuff. The customer was stuck at the second step: Find the *Start* button.

```
Console.WriteLine($"Main window handle: {hwnd}");
IntPtr button = IntPtr.Zero;
while (true) {
  button = FindWindowEx(hwnd, IntPtr.Zero, null, "Start");
  Console.WriteLine($"Start button window handle: {button}");
  if (button != IntPtr.Zero) break;
  Thread.Sleep(TimeSpan.FromSeconds(1));
}
```

The retry loop is to cover the race condition where the test harness looks for the *Start* button before the test app creates it.

> The main window handle is the handle we expect from the test app. But even after a minute, with the *Start* button right there on the screen, the test harness can’t find it.

Okay, so what would prevent `FindWindowEx` from finding a window? Here are some possibilities:

- The thing you want isn’t actually a window. It could be a windowless control. (The fact that their test harness is written in C# suggests that maybe their test app is a WinForms or WPF program, both of which use windowless controls.)
- The window title isn’t exactly what you specified. Maybe there’s an accelerator on it: `&Start`.
- The thing you want isn’t a direct child of that window. Maybe it’s a grandchild.
- The thing you want isn’t a descendent of that window. Maybe you’re passing the wrong parent window.

Basically, all of these options boil down to "`FindWindowEx` is not finding your window because there is in fact no window that meets all the criteria you specified." `FindWindowEx` is working exactly as defined. You need to check that the window you want really does satisfy the criteria you passed.

Double-checking with Spy++ showed that the customer was in the last case: The wrong parent window was being passed. They got the main window from the `Process.MainWindowHandle` property. But that property is synthetic. Windows doesn't have a formal concept of a "main window"; a program can create multiple top-level visible windows, and as far as Windows is concerned, they are all of equal importance.

My guess as to what happened is that the test app created a splash screen, and that got detected as "the main window". Too bad the splash screen doesn't have a *Start* button in it.

The customer updated their test harness to perform an `EnumWindows` to find the top-level window of the test app whose title indicates that it is the window with the *Start* button.

But this is really a case of answering the question without solving the problem.

If you have a test harness that wants to communicate with a test app, you'd be better off having a formal mechanism for the test harness to tell the test app what to do, and a formal mechanism for the test app to report results. Because I suspect the customer's test harness also polls the test app waiting for the phrase *Test passed* or *Test failed* to appear.

For example, the test harness can pass a `/auto` command line option to the test app, and the test app goes into automated mode, where it auto-presses the *Start* button, auto-fills in the text boxes with the appropriate test values, and so on. You might even have `/auto:config.xml` that configures what the test app will do. And then the test app signals whether the test was successful, say by using special exit codes or printing a specific message to `stdout`.

Because what's going to happen sooner or later is that somebody's going to make changes to the test app, maybe rename or rearrange some buttons, or add a new text box that needs to be filled in, and inadvertently break the test harness.

The test harness and the test app are in cahoots. Make the most of it.

Raymond Chen

Follow