The MainWindowHandle property is just a guess based on heuristics

devblogs.microsoft.com/oldnewthing/20220124-00

January 24, 2022

Raymond Chen

A customer had a program written in Windows Forms that wanted the following behavior:

- When the user minimizes the app, it hides the window.
- When the user relaunches the app, the second instance finds the existing (hidden) window and makes it visible again.

They got the "Hide when minimized" part working, but were not having success with the "Find the existing window and make it visible again." Here's their code:

```
Private Sub RestoreHiddenInstance
    For Each process As Process In Process.GetProcesses()
        If process.ProcessName.StartsWith("Contoso") Then
            If proceess.StartTime <> Process.GetCurrentProcess.StartTime Then
                ShowWindow(process.MainWindowHandle, SW_RESTORE)
                ShowWindow(process.MainWindowHandle, SW_SHOW)
                ShowWindow(process.MainWindowHandle, SW_SHOWDEFAULT)
                SetForegroundWindow(process.MainWindowHandle)
            End If
        End If
    Next
End Sub
```

They didn't provide any debugging details about what "didn't work." All they said was that it "didn't work."

We noted some time ago that the `MainWindowHandle` property is just a guess based on heuristics. There is no formal definition of a "main window" for a process. It's a synthetic property driven by enumerating all the top-level of the windows that belong to the process and trying to guess which one is the main one.

And if you go to the reference source, you'll see how the BCL decides whether a window is the main window:

```
bool IsMainWindow(IntPtr handle)
{
    if (NativeMethods.GetWindow(new HandleRef(this, handle),
                                NativeMethods.GW_OWNER) != (IntPtr)0 ||
        !NativeMethods.IsWindowVisible(new Handleref(this, handle)))
        return false;

    return true;
}
```

According to the BCL heuristics, any unowned visible window is a candidate for being the "main" window.

Since the Contoso program hid all of its windows, there are no "main" windows as far as the `MainWindowHandle` property is concerned. The `process.MainWindowHandle` property is `null`, and naturally that means that the code doesn't actually do anything with the main window of the previous instance.

You need to move away from the heuristic-based window-detection and design something more deterministic. Here are some ideas.

- Give the main window a unique class name like `Contoso_MainWindow`. Enumerate the top-level windows owned by the previous instance and look for one that has the correct class name. (This solution won't work for this particular customer because the class name for Windows Forms windows cannot be customized.)
- Register a custom message like `IsContosoMainWindow`. Have your main window respond `TRUE` to that message, and have the second instance send this message to each candidate, and see which one returns `TRUE`.
- Create a named shared memory block and put the window handle in it.
- Find some other shared data storage that you can use to hold the window handle.

This list is far from exhaustive, but gives you an idea of the sort of thinking you need to engage in.

Raymond Chen

Follow