C# Force Form Focus

Asked 16 years, 3 months ago Modified 6 years ago Viewed 57k times

25

So, I did search google and SO prior to asking this question. Basically I have a DLL that has a form compiled into it. The form will be used to display information to the screen. Eventually it will be asynchronous and expose a lot of customization in the dll. For now I just want it to display properly. The problem that I am having is that I use the dll by loading it in a Powershell session. So when I try to display the form and get it to come to the top and have focus, It has no problem with displaying over all the other apps, but I can't for the life of me get it to display over the Powershell window. Here is the code that I am currently using to try and get it to display. I am sure that the majority of it won't be required once I figure it out, this just represents all the things that I found via google.

```
CLass Blah
{
        [DllImport("user32.dll", EntryPoint = "SystemParametersInfo")]
        public static extern bool SystemParametersInfo(uint uiAction, uint
uiParam, uint pvParam, uint fWinIni);

        [DllImport("user32.dll", EntryPoint = "SetForegroundWindow")]
        public static extern bool SetForegroundWindow(IntPtr hWnd);

        [DllImport("User32.dll", EntryPoint = "ShowWindowAsync")]
        private static extern bool ShowWindowAsync(IntPtr hWnd, int cmdShow);
        private const int WS_SHOWNORMAL = 1;

    public void ShowMessage(string msg)
    {
            MessageForm msgFrm = new MessageForm();
            msgFrm.lblMessage.Text = "FOO";
            msgFrm.ShowDialog();
            msgFrm.BringToFront();
            msgFrm.TopMost = true;
            msgFrm.Activate();

            SystemParametersInfo((uint)0x2001, 0, 0, 0x0002 | 0x0001);
            ShowWindowAsync(msgFrm.Handle, WS_SHOWNORMAL);
            SetForegroundWindow(msgFrm.Handle);
            SystemParametersInfo((uint)0x2001, 200000, 200000, 0x0002 |
0x0001);
    }
}
```

As I say I'm sure that most of that is either not needed or even flat out wrong, I just wanted to show the things that I had tried. Also, as I mentioned, I plan to have this be asynchronously displayed at some point which I suspect will wind up requiring a separate thread. Would splitting the form out into it's own thread make it easier to cause it to get focus over the Powershell session?

@Joel, thanks for the info. Here is what I tried based on your suggestion:

```
msgFrm.ShowDialog();
msgFrm.BringToFront();
msgFrm.Focus();
Application.DoEvents();
```

The form still comes up *under* the Powershell session. I'll proceed with working out the threading. I've spawned threads before but never where the parent thread needed to talk to the child thread, so we'll see how it goes.

Thnks for all the ideas so far folks.

Ok, threading it took care of the problem. @Quarrelsome, I did try both of those. Neither (nor both together) worked. I am curious as to what is evil about using threading? I am not using Application.Run and I have yet to have a problem. I am using a mediator class that both the parent thread and the child thread have access to. In that object I am using a ReaderWriterLock to lock one property that represents the message that I want displayed on the form that the child thread creates. The parent locks the property then writes what should be displayed. The child thread locks the property and reads what it should change the label on the form to. The child has to do this on a polling interval (I default it to 500ms) which I'm not real happy about, but I could not find an event driven way to let the child thread know that the proerty had changed, so I'm stuck with polling.

c# winforms

Share

Improve this question

Follow

edited Jul 14, 2015 at 7:59

Yeldar Kurmangaliyev
34.2k ● 13 ● 63 ● 103

asked Sep 5, 2008 at 15:25

EBGreen
37.6k ● 12 ● 68 ● 86

Can you just call Focus() on one of the controls in the form? – Chad Boyer Sep 5, 2008 at 15:50

Thanks for the responses. @Chad, just trying to Focus() a control on the form wound up with the same results that I was already getting. @Dean, I think to use your method I would need to split the form out into it's own thread. I planned to do that nyway at some point so I guess I'll start on that now. – EBGreen Sep 5, 2008 at 16:04

8 Answers

Sorted by: Highest score (default)

18

I also had trouble activating and bringing a window to the foreground. Here is the code that eventually worked for me. I'm not sure if it will solve your problem.

Basically, call ShowWindow() then SetForegroundWindow().

```
using System.Diagnostics;
using System.Runtime.InteropServices;

// Sets the window to be foreground
[DllImport("User32")]
private static extern int SetForegroundWindow(IntPtr hwnd);

// Activate or minimize a window
[DllImportAttribute("User32.DLL")]
private static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
private const int SW_SHOW = 5;
private const int SW_MINIMIZE = 6;
private const int SW_RESTORE = 9;

private void ActivateApplication(string briefAppName)
{
    Process[] procList = Process.GetProcessesByName(briefAppName);

    if (procList.Length > 0)
    {
        ShowWindow(procList[0].MainWindowHandle, SW_RESTORE);
        SetForegroundWindow(procList[0].MainWindowHandle);
    }
}
```

Share

Improve this answer

Follow

edited Apr 20, 2011 at 21:29
Calvin Fisher
4,701 ●5 ●37 ●47

answered Sep 5, 2008 at 15:49
Dean Hill
4,499 ●6 ●33 ●36

Where do you find the original definitions for all of these SW_ constants? – gonzobrains Feb 21, 2013 at 23:04

16

Here is some code that I've used on one form or another for a few years. There are a few gotchas to making a window in another app pop up. Once you have the window handle do this:

```
      if (IsIconic(hWnd))
        ShowWindowAsync(hWnd, SW_RESTORE);

      ShowWindowAsync(hWnd, SW_SHOW);

      SetForegroundWindow(hWnd);

      // Code from Karl E. Peterson, www.mvps.org/vb/sample.htm
      // Converted to Delphi by Ray Lischner
      // Published in The Delphi Magazine 55, page 16
      // Converted to C# by Kevin Gale
      IntPtr foregroundWindow = GetForegroundWindow();
      IntPtr Dummy = IntPtr.Zero;

      uint foregroundThreadId = GetWindowThreadProcessId(foregroundWindow,
Dummy);
      uint thisThreadId       = GetWindowThreadProcessId(hWnd, Dummy);

      if (AttachThreadInput(thisThreadId, foregroundThreadId, true))
      {
        BringWindowToTop(hWnd); // IE 5.5 related hack
        SetForegroundWindow(hWnd);
        AttachThreadInput(thisThreadId, foregroundThreadId, false);
      }

      if (GetForegroundWindow() != hWnd)
      {
        // Code by Daniel P. Stasinski
        // Converted to C# by Kevin Gale
        IntPtr Timeout = IntPtr.Zero;
        SystemParametersInfo(SPI_GETFOREGROUNDLOCKTIMEOUT, 0, Timeout, 0);
        SystemParametersInfo(SPI_SETFOREGROUNDLOCKTIMEOUT, 0, Dummy,
SPIF_SENDCHANGE);
        BringWindowToTop(hWnd); // IE 5.5 related hack
        SetForegroundWindow(hWnd);
        SystemParametersInfo(SPI_SETFOREGROUNDLOCKTIMEOUT, 0, Timeout,
SPIF_SENDCHANGE);
      }
```

I won't post the whole unit since since it does other things that aren't relevant but here are the constants and imports for the above code.

```
//Win32 API calls necesary to raise an unowned processs main window

[DllImport("user32.dll")]

private static extern bool SetForegroundWindow(IntPtr hWnd);
[DllImport("user32.dll")]
private static extern bool ShowWindowAsync(IntPtr hWnd, int nCmdShow);
[DllImport("user32.dll")]
private static extern bool IsIconic(IntPtr hWnd);
[DllImport("user32.dll", SetLastError = true)]
private static extern bool SystemParametersInfo(uint uiAction, uint uiParam,
IntPtr pvParam, uint fWinIni);
[DllImport("user32.dll", SetLastError = true)]
private static extern uint GetWindowThreadProcessId(IntPtr hWnd, IntPtr
lpdwProcessId);
[DllImport("user32.dll")]
private static extern IntPtr GetForegroundWindow();
[DllImport("user32.dll")]
private static extern bool AttachThreadInput(uint idAttach, uint idAttachTo,
bool fAttach);
[DllImport("user32.dll")]
static extern bool BringWindowToTop(IntPtr hWnd);

[DllImport("user32.dll")]
private static extern int GetWindowText(IntPtr hWnd, StringBuilder lpString,
Int32 nMaxCount);
[DllImport("user32.dll")]
```

```
private static extern int GetWindowThreadProcessId(IntPtr hWnd, ref Int32
lpdwProcessId);
[DllImport("User32.dll")]
public static extern IntPtr GetParent(IntPtr hWnd);

private const int SW_HIDE = 0;
private const int SW_SHOWNORMAL = 1;
private const int SW_NORMAL = 1;
private const int SW_SHOWMINIMIZED = 2;
private const int SW_SHOWMAXIMIZED = 3;
private const int SW_MAXIMIZE = 3;
private const int SW_SHOWNOACTIVATE = 4;
private const int SW_SHOW = 5;
private const int SW_MINIMIZE = 6;
private const int SW_SHOWMINNOACTIVE = 7;
private const int SW_SHOWNA = 8;
private const int SW_RESTORE = 9;
private const int SW_SHOWDEFAULT = 10;
private const int SW_MAX = 10;

private const uint SPI_GETFOREGROUNDLOCKTIMEOUT = 0x2000;
private const uint SPI_SETFOREGROUNDLOCKTIMEOUT = 0x2001;
private const int  SPIF_SENDCHANGE = 0x2;
```

Share

Improve this answer

Follow

edited Nov 28, 2011 at 13:06
Stefan Paul Noack
3,734 ● 1 ● 29 ● 38

answered Sep 18, 2008 at 17:23
Kevin Gale
4,428 ● 6 ● 34 ● 31

1 This is the only answer here that worked for me, thanks Gevin for sharing!
– Patrick from NDepend team Sep 4, 2010 at 11:24

2 Btw, it seems like a dangerous practice according to Raymond Chen this can provoke UI freeze: I warned you: The dangers of attaching input queues blogs.msdn.com/b/oldnewthing/archive/2008/08/01/8795860.aspx Did you ever meet this problem Kevin? – Patrick from NDepend team Sep 4, 2010 at 12:27

Been using this code for years at many customer sites in an applicatoin that runs 24x7 with no reported problems. That doesn't mean Chen isn't right but if it is a problem it doesn't happen very often. – Kevin Gale Sep 5, 2010 at 13:57

1 This works great in general, however it won't work if a CMD window has focus, I believe this is because CMD windows don't have a message loop so you can't AttachThreadInput.
– Maurice Flanagan May 6, 2012 at 20:25

1 Interesting about the CMD window. Never bumped into that one. It seems like nothing works completely. It's frustrating. As a user I hate windows that suddenly pop up and steal focus. But users of our software demand this all the time. We constantly get requirements like "Every hour pop a window up and force the user to enter some number". – Kevin Gale May 9, 2012 at 14:53

Doesn't **ShowDialog()** have different window behavior than just **Show()**?

What if you tried:

3

```
msgFrm.Show();
msgFrm.BringToFront();
msgFrm.Focus();
```

Share Improve this answer Follow

answered Sep 5, 2008 at 18:22

sieben
2,191 ● 4 ● 23 ● 31

3

TopMost = true; .Activate() ?

Either of those any good?

Splitting it out into its own thread is a bit evil as it wont work properly if you don't call it with Application.Run and that will swallow up the thread. In the worst case scenario I guess you could separate it out into a different process and communicate via the disk or WCF.

Share Improve this answer Follow

answered Sep 5, 2008 at 18:24

Quibblesome
25.4k ● 10 ● 62 ● 104

2

The following solution should meet your requirements:

1. Assembly can be loaded into PowerShell and main class instantiated
2. When ShowMessage method on this instance is called, a new window is shown and activated
3. If you call ShowMessage multiple times, this same window updates its title text and is activated
4. To stop using the window, call Dispose method

Step 1: Let's create a temporary working directory (you can naturally use your own dir)

```
(powershell.exe)
mkdir C:\TEMP\PshWindow
cd C:\TEMP\PshWindow
```

Step 2: Now let's define class that we will be interacting with in PowerShell:

```
// file 'InfoProvider.cs' in C:\TEMP\PshWindow
using System;
using System.Threading;
using System.Windows.Forms;
```

```
namespace PshWindow
{
    public sealed class InfoProvider : IDisposable
    {
        public void Dispose()
        {
            GC.SuppressFinalize(this);
            lock (this._sync)
            {
                if (!this._disposed)
                {
                    this._disposed = true;
                    if (null != this._worker)
                    {
                        if (null != this._form)
                        {
                            this._form.Invoke(new Action(() =>
this._form.Close()));
                        }
                        this._worker.Join();
                        this._form = null;
                        this._worker = null;
                    }
                }
            }
        }

        public void ShowMessage(string msg)
        {
            lock (this._sync)
            {
                // make sure worker is up and running
                if (this._disposed) { throw new
ObjectDisposedException("InfoProvider"); }
                if (null == this._worker)
                {
                    this._worker = new Thread(() => (this._form = new
MyForm(this._sync)).ShowDialog()) { IsBackground = true };
                    this._worker.Start();
                    while (this._form == null || !this._form.Created)
                    {
                        Monitor.Wait(this._sync);
                    }
                }

                // update the text
                this._form.Invoke(new Action(delegate
                {
                    this._form.Text = msg;
                    this._form.Activate();
                }));
            }
        }

        private bool _disposed;
        private Form _form;
        private Thread _worker;
        private readonly object _sync = new object();
    }
}
```

As well as the Form that will be shown:

```
// file 'MyForm.cs' in C:\TEMP\PshWindow
using System;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;

namespace PshWindow
{
    internal sealed class MyForm : Form
    {
        public MyForm(object sync)
        {
            this._sync = sync;
            this.BackColor = Color.LightGreen;
            this.Width = 200;
            this.Height = 80;
            this.FormBorderStyle = FormBorderStyle.SizableToolWindow;
        }

        protected override void OnShown(EventArgs e)
        {
            base.OnShown(e);
            this.TopMost = true;

            lock (this._sync)
            {
                Monitor.PulseAll(this._sync);
            }
        }

        private readonly object _sync;
    }
}
```

Step 3: Let's compile the assembly...

```
(powershell.exe)
csc /out:PshWindow.dll /target:library InfoProvider.cs MyForm.cs
```

Step 4: ... and load the assembly in PowerShell to have fun with it:

```
(powershell.exe)
[System.Reflection.Assembly]::LoadFile('C:\TEMP\PshWindow\PshWindow.dll')
$a = New-Object PshWindow.InfoProvider
$a.ShowMessage('Hello, world')
```

A green-ish window with title 'Hello, world' should now pop-up and be active. If you reactivate the PowerShell window and enter:

```
$a.ShowMessage('Stack overflow')
```

The Window's title should change to 'Stack overflow' and the window should be active again.

To stop working with our window, dispose the object:

```
$a.Dispose()
```

This solution works as expected in both Windows XP SP3, x86 and Windows Vista SP1, x64. If there are question about how this solution works I can update this entry with detailed discussion. For now I'm hoping the code if self-explanatory.

Share Improve this answer Follow

answered Jan 13, 2009 at 13:41

Milan Gardian

11.4k ● 5 ● 41 ● 45

1

Huge thanks people.
I think I've made it a bit shorter, here's what I put on a seperate thread and seems to be working ok.

```
private static void StatusChecking()
{
    IntPtr iActiveForm = IntPtr.Zero, iCurrentACtiveApp = IntPtr.Zero;
    Int32 iMyProcID = Process.GetCurrentProcess().Id, iCurrentProcID = 0;
    IntPtr iTmp = (IntPtr)1;

    while (bIsRunning)
    {
        try
        {
            Thread.Sleep(45);
            if (Form.ActiveForm != null)
            {
                iActiveForm = Form.ActiveForm.Handle;
            }
            iTmp = GetForegroundWindow();
            if (iTmp == IntPtr.Zero) continue;
            GetWindowThreadProcessId(iTmp, ref iCurrentProcID);
            if (iCurrentProcID == 0)
            {
                iCurrentProcID = 1;
                continue;
            }
            if (iCurrentProcID != iMyProcID)
            {
                SystemParametersInfo(SPI_GETFOREGROUNDLOCKTIMEOUT, 0,
IntPtr.Zero, 0);
                SystemParametersInfo(SPI_SETFOREGROUNDLOCKTIMEOUT, 0,
IntPtr.Zero, SPIF_SENDCHANGE);
                BringWindowToTop(iActiveForm);
                SetForegroundWindow(iActiveForm);
            }
            else iActiveForm = iTmp;
        }
        catch (Exception ex)
        {
            Definitions.UnhandledExceptionHandler(ex, 103106);
        }
    }
}
```

I don`t bother repasting the definitions...

Share
Improve this answer
Follow

edited Dec 10, 2018 at 20:46
Jimi
32.1k ● 8 ● 48 ● 75

answered Apr 4, 2009 at 17:12
Ivan Arabadzhiev

0

You shouldn't need to import any win32 functions for this. If .Focus() isn't enough the form should also have a .BringToFront() method you can use. If that fails, you can set it's .TopMost property to true. You don't want to *leave* it true forever, so then call Application.DoEvents so the form can process that message and set it back to false.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:16
Joel Coehoorn
415k ● 114 ● 577 ● 813

0

Don't you just want the dialog to be a child of the calling form?

To do that you'll need the pass in the calling window and use the ShowDialog(IWin32Window owner) method.

Share Improve this answer Follow

answered Sep 30, 2008 at 22:18
jyoung
5,151 ● 4 ● 33 ● 48