Draw Behind Desktop Icons in Windows 8

Gerald Degeneve, 23 Dec 2014 CPOL

4.98 (31 votes)

Draw or render a Windows Form directly over the Wallpaper, behind the Desktop Icons in Windows 8

Download source - 13.8 KB

Introduction

Those who read this article probably know DreamScene, that Windows Vista feature, that allows to render video sequences (in .dream format) as desktop background. Then there is a tool called rainmeter, that allows you to place widgets/gadgets/whatever on your desktop in a top most, top, bottom and under desktop icons manner.

These tools have one thing in common, they do not fully support Windows 8, at least the "under the desktop icons" part they don't.

How It Used To Work (Windows XP, Vista, 7)

There is the window tree. This tree contains all windows that are currently displayed/or hidden on the current desktop and then there is a tool called Spy++ (Visual Studio -> Tools -> Spy++), that can be used to display and navigate that tree. That tool is part of Visual Studio.

The last leaf of this tree is the Program Manager. This window represents the whole shell. In Windows XP, Vista and 7 (Aero turned off) this Program Manager (Progman) contained a window (`SysListView32`), that rendered the desktop icons. So if you set that Program Manager as your parent window, you could position yourself right behind those desktop icons.

There is a great article on drawing to the desktop (in front of and behind desktop icons) here on CodeProject:

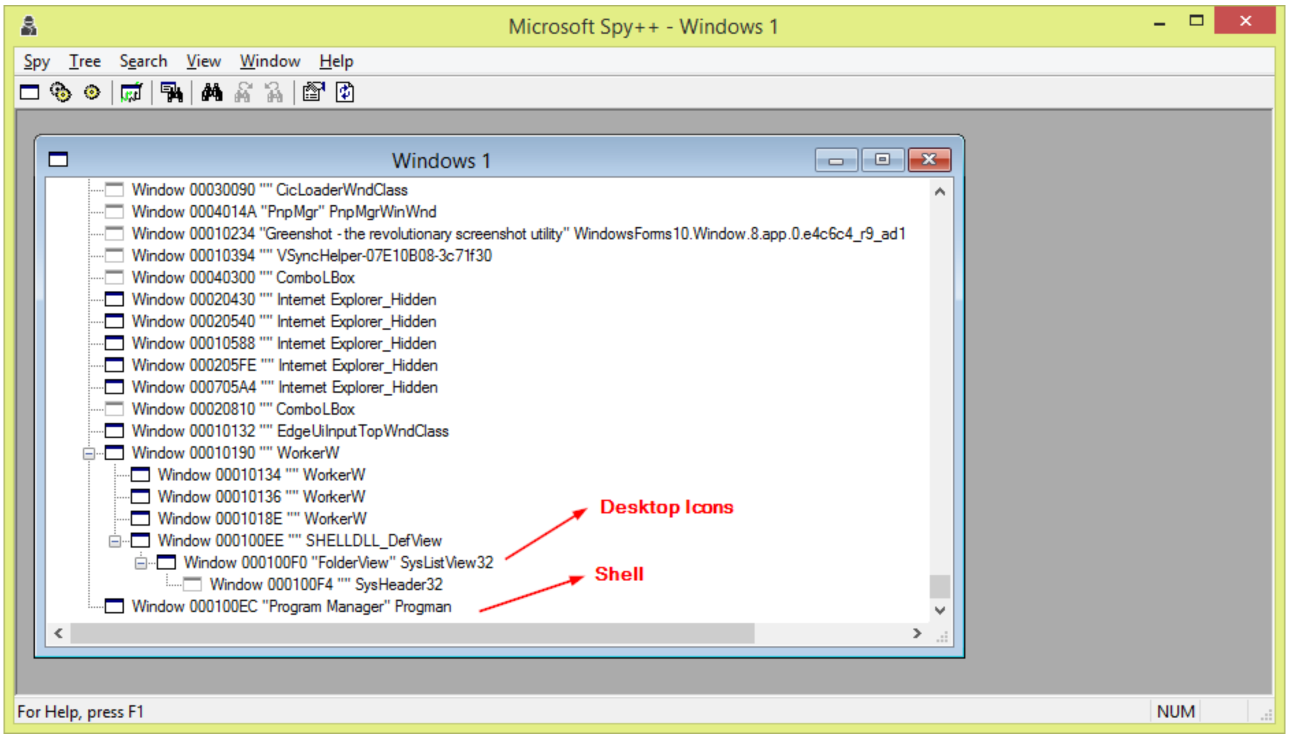
- Falling Snow on Your Desktop! Part II by Igor Tolmachev

But for the life of me, I could not find an approach that worked for Windows 8.

The Problem with Windows 8

Windows 7 and Windows 8 are very similar. To make the Windows XP approach work for Windows 7, you had to turn off aero desktop. With Windows 8, you cannot turn aero off, so there has to be another way.

The following image shows how the window tree is structured in Windows 8.



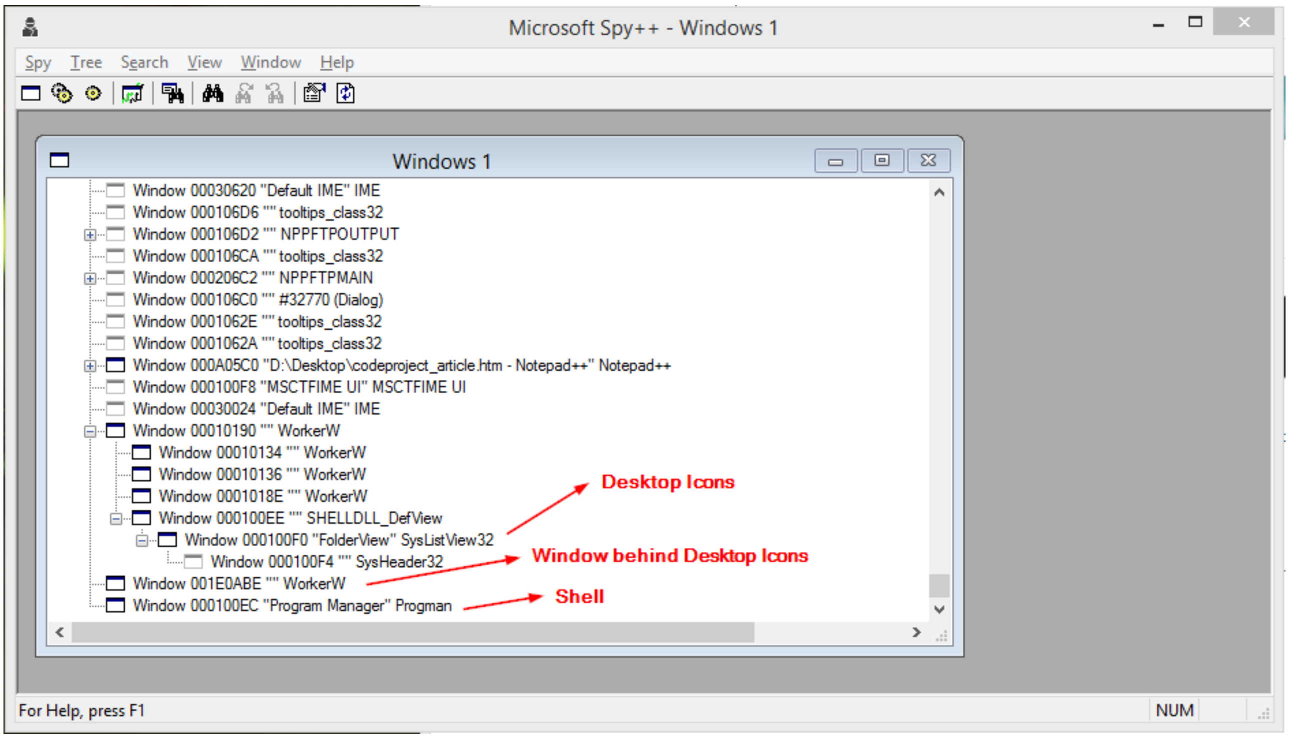
```
Window 00030090 "" CicLoaderWndClass
Window 0004014A "PnpMgr" PnpMgrWinWnd
Window 00010234 "Greenshot - the revolutionary screenshot utility" WindowsForms10.Window.8.app.0.e4c6c4_r9_ad1
Window 00010394 "" VSyncHelper-07E10B08-3c71f30
Window 00040300 "" ComboLBox
Window 00020430 "" Internet Explorer_Hidden
Window 00020540 "" Internet Explorer_Hidden
Window 00010588 "" Internet Explorer_Hidden
Window 000205FE "" Internet Explorer_Hidden
Window 000705A4 "" Internet Explorer_Hidden
Window 00020810 "" ComboLBox
Window 00010132 "" EdgeUiInputTopWndClass
Window 00010190 "" WorkerW
  Window 00010134 "" WorkerW
  Window 00010136 "" WorkerW
  Window 0001018E "" WorkerW
  Window 000100EE "" SHELLDLL_DefView
    Window 000100F0 "FolderView" SysListView32
      Window 000100F4 "" SysHeader32
Window 000100EC "Program Manager" Progman
```

The `SysListView32` is now separated from the Program Manager. This alone is not that big of a problem. The problem is the desktop background. It is now fused with the Desktop Icons. So we can only draw over everything including the icons, or draw under everything including the background. It is not possible to position a window in the Z-Order so it is between the desktop icons and the desktop wallpaper. I tried every position.

A Way to Find a Solution

What led me to the solution presented in this article was the personalization dialog. When you set a wallpaper manually, you do not see a sudden change, instead the wallpaper is set using a smooth fade animation. In my opinion, such an animation is only possible if the system can somehow draw behind the desktop icons, since setting a wallpaper in rapid succession would be very slow and ugly.

So I set up Spy++, opened the personalization dialog and changed the wallpaper. It turns out, that when you change the desktop wallpaper, a new `WorkerW` window between the `WorkerW` instance that holds the Desktop Icons (`SysListView32`) and the Desktop Manager is created.



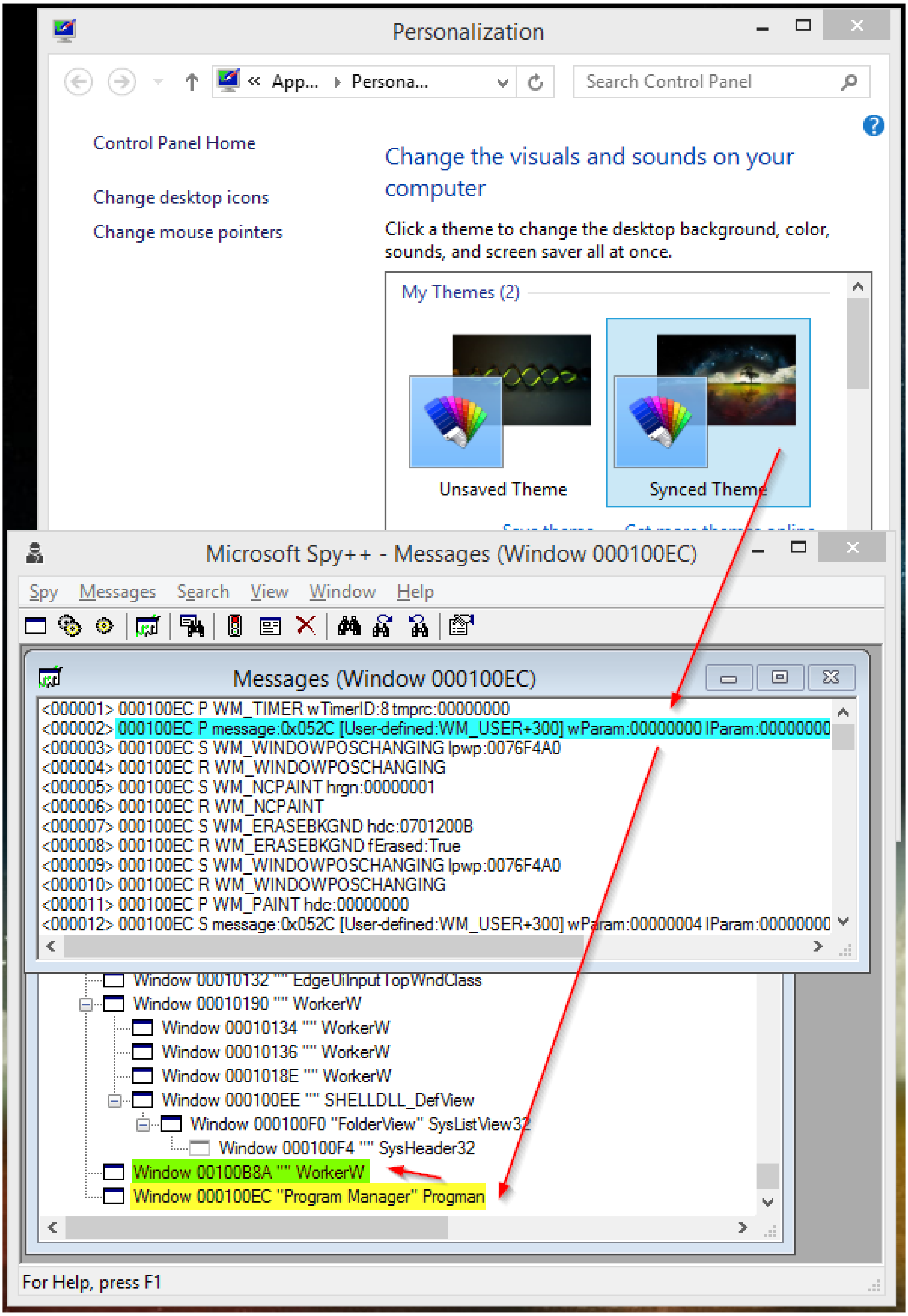
```
Window 00030620 "Default IME" IME
Window 000106D6 "" tooltips_class32
Window 000106D2 "" NPPFTPOUTPUT
Window 000106CA "" tooltips_class32
Window 000206C2 "" NPPFTPMAIN
Window 000106C0 "" #32770 (Dialog)
Window 0001062E "" tooltips_class32
Window 0001062A "" tooltips_class32
Window 000A05C0 "D:\Desktop\codeproject_article.htm - Notepad++" Notepad++
Window 000100F8 "MSCTFIME UI" MSCTFIME UI
Window 00030024 "Default IME" IME
Window 00010190 "" WorkerW
    Window 00010134 "" WorkerW
    Window 00010136 "" WorkerW
    Window 0001018E "" WorkerW
    Window 000100EE "" SHELLDLL_DefView
        Window 000100F0 "FolderView" SysListView32
            Window 000100F4 "" SysHeader32
Window 001E0ABE "" WorkerW
Window 000100EC "Program Manager" Progman
```

I took the handle of that new `WorkerW` window and put it into my test program and it was finally able to draw behind the desktop icons, directly above the wallpaper!

One problem remained. When I closed the personalization dialog, it took down that new `WorkerW` window with it.

I had to find a way to trigger the creation of this `WorkerW` window.

Spy++ reports the window as a sibling and child of the Program Manager, so it looks like the Program Manager creates it. I turned on message monitoring for the Program Manager using Spy++ and found what I was looking for.


Personalization
App... › Persona...
Search Control Panel
Control Panel Home
Change desktop icons
Change mouse pointers
Change the visuals and sounds on your computer
Click a theme to change the desktop background, color, sounds, and screen saver all at once.
My Themes (2)
Unsaved Theme
Synced Theme
Microsoft Spy++ - Messages (Window 000100EC)
Spy Messages Search View Window Help
Messages (Window 000100EC)
For Help, press F1

```
<000001> 000100EC P WM_TIMER wTimerID:8 tmprc:00000000
<000002> 000100EC P message:0x052C [User-defined:WM_USER+300] wParam:00000000 lParam:00000000
<000003> 000100EC S WM_WINDOWPOSCHANGING lpwp:0076F4A0
<000004> 000100EC R WM_WINDOWPOSCHANGING
<000005> 000100EC S WM_NCPAINT hrgn:00000001
<000006> 000100EC R WM_NCPAINT
<000007> 000100EC S WM_ERASEBKGND hdc:0701200B
<000008> 000100EC R WM_ERASEBKGND fErased:True
<000009> 000100EC S WM_WINDOWPOSCHANGING lpwp:0076F4A0
<000010> 000100EC R WM_WINDOWPOSCHANGING
<000011> 000100EC P WM_PAINT hdc:00000000
<000012> 000100EC S message:0x052C [User-defined:WM_USER+300] wParam:00000004 lParam:00000000
```

```
Window 00010132 "" EdgeUiInputTopWndClass
Window 00010190 "" WorkerW
    Window 00010134 "" WorkerW
    Window 00010136 "" WorkerW
    Window 0001018E "" WorkerW
    Window 000100EE "" SHELLDLL_DefView
        Window 000100F0 "FolderView" SysListView32
            Window 000100F4 "" SysHeader32
Window 00100B8A "" WorkerW
Window 000100EC "Program Manager" Progman
```

Right after the click to change the desktop wallpaper, the Program Manager receives a bunch of messages. The first one being a user defined and undocumented message. This reeks after being "recently" added.

I extended the test program to send exactly this user defined message (0x052C) to the Program Manager. And it caused exactly what I was hoping for. After receiving the message, the Program Manager creates the `WorkerW` window.

With all ingredients ready, I wrote a small demo application that illustrates how to draw on the desktop (behind the icons) and how to put a window behind the desktop icons.

The Code

Obtain Program Manager Handle

First, we begin by finding the handle of the `Progman` window. We can use the `FindWindow` function provided by the Windows API to accomplish this task.

```
// Fetch the Progman window
IntPtr progman = W32.FindWindow("Progman", null);
```

Send Message to Program Manager

To trigger the creation of a `WorkerW` window between the desktop icons and the wallpaper, we have to send the Program Manager a message. That message is an undocumented one, so there is no fancy Windows API name for it, except `0x052C`. In order to send the message, we use the Windows API method `SendMessageTimeout`.

```
IntPtr result = IntPtr.Zero;

// Send 0x052C to Progman. This message directs Progman to spawn a
// WorkerW behind the desktop icons. If it is already there, nothing
// happens.
W32.SendMessageTimeout(progman,
                       0x052C,
                       new IntPtr(0),
                       IntPtr.Zero,
                       W32.SendMessageTimeoutFlags.SMTO_NORMAL,
                       1000,
                       out result);
```

Obtain Handle to Newly Created Window

Now, we have to obtain a handle to that newly created `WorkerW` window. Since there is more than one window with title "" and class "`WorkerW`", we have to go through the window tree sequentially. This can be done using the `EnumWindows` function.

`EnumWindows` calls supplied `EnumWindowProc` for every top level window. From there, we can check if the current window contains a child named "`SHELLDLL_DefView`", which indicates that the current window represents the desktop icons. We then take the next sibling of that window.

```
// Spy++ output
// .....
// 0x00010190 "" WorkerW
//   ...
//   0x000100EE "" SHELLDLL_DefView
//     0x000100F0 "FolderView" SysListView32
// 0x00100B8A "" WorkerW       <-- This is the WorkerW instance we are after!
// 0x000100EC "Program Manager" Progman

IntPtr workerw = IntPtr.Zero;

// We enumerate all Windows, until we find one, that has the SHELLDLL_DefView
// as a child.
// If we found that window, we take its next sibling and assign it to workerw.
W32.EnumWindows(new W32.EnumWindowsProc((tophandle, topparamhandle) =>
{
    IntPtr p = W32.FindWindowEx(tophandle,
                                IntPtr.Zero,
                                "SHELLDLL_DefView",
                                IntPtr.Zero);

    if (p != IntPtr.Zero)
    {
        // Gets the WorkerW Window after the current one.
        workerw = W32.FindWindowEx(IntPtr.Zero,
                                   tophandle,
                                   "WorkerW",
                                   IntPtr.Zero);
    }

    return true;
}), IntPtr.Zero);
```

Demo 1: Draw Graphics Between Icons and Wallpaper

With the `workerw` handle in hand, the fun stuff begins. The first demo is about using the `System.Drawing` classes to just draw something.

This demo draws a rectangle in the upper left corner of the desktop. If you use multiple monitors, be aware, that the desktop area spans a rectangle across all monitors, so make sure your left monitor is turned on and your monitor placement actually maps the top left corner to a monitor, in case you have four of them, with one atop the other three.

Note: Everything you draw onto this layer will stay there until you paint over it, invalidate it, or reset your wallpaper.

```
// Get the Device Context of the WorkerW
IntPtr dc = W32.GetDCEx(workerw, IntPtr.Zero, (W32.DeviceContextValues)0x403);
if (dc != IntPtr.Zero)
{
    // Create a Graphics instance from the Device Context
    using (Graphics g = Graphics.FromHdc(dc))
    {

        // Use the Graphics instance to draw a white rectangle in the upper
        // left corner. In case you have more than one monitor think of the
        // drawing area as a rectangle that spans across all monitors, and
        // the 0,0 coordinate being in the upper left corner.
        g.FillRectangle(new SolidBrush(Color.White), 0, 0, 500, 500);

    }
    // make sure to release the device context after use.
    W32.ReleaseDC(workerw, dc);
}
```

Demo 2: Put a Windows Form behind desktop icons

This demo shows how to put a normal Windows Form behind desktop icons. In essence, this can be done by setting the parent of a Windows Form to our `WorkerW` window. To set the parent of a form, we can use the `SetParent` Windows API function.

Note: For this function to work, the form has to be already created. The `form.Load` event seems to be the right place for it.

For the sake of a short example, I created the Form in place, without the "Project->Add Windows Form..." dialog and the designer.

```
Form form = new Form();
form.Text = "Test Window";

form.Load += new EventHandler((s, e) =>
{
    // Move the form right next to the in demo 1 drawn rectangle
    form.Width = 500;
    form.Height = 500;
    form.Left = 500;
    form.Top = 0;

    // Add a randomly moving button to the form
    Button button = new Button() { Text = "Catch Me" };
    form.Controls.Add(button);
    Random rnd = new Random();
    System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer();
    timer.Interval = 100;
    timer.Tick += new EventHandler((sender, eventArgs) =>
    {
        button.Left = rnd.Next(0, form.Width - button.Width);
        button.Top = rnd.Next(0, form.Height - button.Height);
    });
    timer.Start();

    // This line makes the form a child of the WorkerW window,
    // thus putting it behind the desktop icons and out of reach
    // for any user input. The form will just be rendered, no
    // keyboard or mouse input will reach it. You would have to use
    // WH_KEYBOARD_LL and WH_MOUSE_LL hooks to capture mouse and
    // keyboard input and redirect it to the windows form manually,
    // but that's another story, to be told at a later time.
    W32.SetParent(form.Handle, workerw);
});

// Start the Application Loop for the Form.
Application.Run(form);
```

You will probably notice that there is no way to interact with the form, once its parent is set to the `WorkerW` window. The desktop is not designed to have interactive children, so all events regarding mouse movement, keyboard input, etc. will not reach our Form.

There is a way around that. You can subscribe to the low level `WH_KEYBOARD_LL` and `WH_MOUSE_LL` events, also known from key loggers and mouse capture software. Via those events, you can receive mouse movements, clicks and key presses regardless of where they occur. You would have to forward those messages to your form and perform your own hit testing.

Conclusion

One command to rule them all

```
W32.SendMessageTimeout(W32.FindWindow("Progman", null),
                       0x052C,
                       new IntPtr(0),
                       IntPtr.Zero,
                       W32.SendMessageTimeoutFlags.SMTO_NORMAL,
                       1000,
                       out result);
```

License

Share

About the Author

Gerald Degeneve

Software Developer (Senior) SecureGUARD GmbH

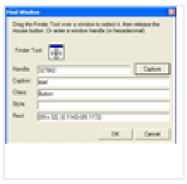
Austria

No Biography provided

You may also be interested in...

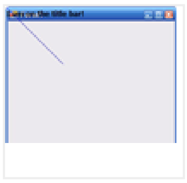
Draw with Mouse

Taking COBOL mobile



Screen Captures, Window Captures and Window Icon Captures with Spy++ style Window Finder!

COBOL programmers: Skill up and save time...



Special 'Graphics' Objects to Draw Anywhere on your Window

Barcodescanner with webcam on Intel® Edison

Comments and Discussions

27 messages have been posted for this article Visit **http://www.codeproject.com/Articles/856020/Draw-Behind-Desktop-Icons-in-Windows** to post and view comments on this article, or click **here** to get a print view with messages.