

# Forum:[Hack Forums](#)

Thread: [hotkeys](#)

**SIM-2618262** - August 3, 2015, 2:58 pm

hello HF

so im making some hotkeys for my application and it works. but when im in a game it doesn't work, it works in cs but the game i want it to work in doesn't work. the game the hotkeys don't work in is called infestaion survivor stories. it worked on my last pc (it had windows7)

but on my new pc it don't work?(windows8.1) i have used a good while to try to figure it out but i can't seem to find much similar

**Chris White** - August 3, 2015, 4:06 pm

Found this file. Have a look at it. If you know what you're doing you'll be able to use it. I used it in a keybinder once before to send text commands in a game once before

PS: It's in C#. Compile it as a DLL and implement it in your VB.Net program

```
Code using System;
using System.Windows.Forms;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.InteropServices;
```

```
namespace MovablePython
```

```
{
    public class Hotkey : IMessageFilter
    {
        #region Interop

        [DllImport("user32.dll", SetLastError = true)]
        private static extern int RegisterHotKey(IntPtr hWnd, int id, uint fsModifiers, Keys vk);

        [DllImport("user32.dll", SetLastError=true)]
        private static extern int UnregisterHotKey(IntPtr hWnd, int id);

        private const uint WM_HOTKEY = 0x312;

        private const uint MOD_ALT = 0x1;
        private const uint MOD_CONTROL = 0x2;
        private const uint MOD_SHIFT = 0x4;
        private const uint MOD_WIN = 0x8;

        private const uint ERROR_HOTKEY_ALREADY_REGISTERED = 1409;

        #endregion

        private static int currentID;
        private const int maximumID = 0xBFFF;

        private Keys keyCode;
        private bool shift;
```

```

private bool control;
private bool alt;
private bool windows;

[XmlIgnore]
private int id;
[XmlIgnore]
private bool registered;
[XmlIgnore]
private Control windowControl;

public event EventHandler Pressed;

public Hotkey() : this(Keys.None, false, false, false, false)
{
    // No work done here!
}

public Hotkey(Keys keyCode, bool shift, bool control, bool alt, bool windows)
{
    // Assign properties
    this.KeyCode = keyCode;
    this.Shift = shift;
    this.Control = control;
    this.Alt = alt;
    this.Windows = windows;

    // Register us as a message filter
    Application.AddMessageFilter(this);
}

~Hotkey()
{
    // Unregister the hotkey if necessary
    if (this.Registered)
    { this.Unregister(); }
}

public Hotkey Clone()
{
    // Clone the whole object
    return new Hotkey(this.keyCode, this.shift, this.control, this.alt, this.windows);
}

public bool GetCanRegister(Control windowControl)
{
    // Handle any exceptions: they mean "no, you can't register" :)
    try
    {
        // Attempt to register
        if (!this.Register(windowControl))
        { return false; }
    }
}

```

```

        // Unregister and say we managed it
        this.Unregister();
        return true;
    }
    catch (Win32Exception)
    { return false; }
    catch (NotSupportedException)
    { return false; }
}

public bool Register(Control windowControl)
{
    // Check that we have not registered
    if (this.registered)
    { throw new NotSupportedException("You cannot register a hotkey that is already
registered"); }

    // We can't register an empty hotkey
    if (this.Empty)
    { throw new NotSupportedException("You cannot register an empty hotkey"); }

    // Get an ID for the hotkey and increase current ID
    this.id = Hotkey.currentID;
    Hotkey.currentID = Hotkey.currentID + 1 % Hotkey.maximumID;

    // Translate modifier keys into unmanaged version
    uint modifiers = (this.Alt ? Hotkey.MOD_ALT : 0) | (this.Control ? Hotkey.MOD_CONTROL :
0) |
        (this.Shift ? Hotkey.MOD_SHIFT : 0) | (this.Windows ? Hotkey.MOD_WIN : 0);

    // Register the hotkey
    if (Hotkey.RegisterHotKey(windowControl.Handle, this.id, modifiers, keyCode) == 0)
    {
        // Is the error that the hotkey is registered?
        if (Marshal.GetLastWin32Error() == ERROR_HOTKEY_ALREADY_REGISTERED)
        { return false; }
        else
        { throw new Win32Exception(); }
    }

    // Save the control reference and register state
    this.registered = true;
    this.windowControl = windowControl;

    // We successfully registered
    return true;
}

public void Unregister()
{
    // Check that we have registered
    if (!this.registered)
    { throw new NotSupportedException("You cannot unregister a hotkey that is not

```

```
registered"); }
```

```
// It's possible that the control itself has died: in that case, no need to unregister!  
if (!this.windowControl.IsDisposed)  
{  
    // Clean up after ourselves  
    if (Hotkey.UnregisterHotKey(this.windowControl.Handle, this.id) == 0)  
        { throw new Win32Exception(); }  
}
```

```
// Clear the control reference and register state  
this.registered = false;  
this.windowControl = null;  
}
```

```
private void Reregister()  
{  
    // Only do something if the key is already registered  
    if (!this.registered)  
        { return; }  
  
    // Save control reference  
    Control windowControl = this.windowControl;  
  
    // Unregister and then reregister again  
    this.Unregister();  
    this.Register(windowControl);  
}
```

```
public bool PreFilterMessage(ref Message message)  
{  
    // Only process WM_HOTKEY messages  
    if (message.Msg != Hotkey.WM_HOTKEY)  
        { return false; }  
  
    // Check that the ID is our key and we are registered  
    if (this.registered && (message.WParam.ToInt32() == this.id))  
    {  
        // Fire the event and pass on the event if our handlers didn't handle it  
        return this.OnPressed();  
    }  
    else  
        { return false; }  
}
```

```
private bool OnPressed()  
{  
    // Fire the event if we can  
    HandledEventArgs handledEventArgs = new HandledEventArgs(false);  
    if (this.Pressed != null)  
        { this.Pressed(this, handledEventArgs); }  
  
    // Return whether we handled the event or not
```

```

        return handledEventArgs.Handled;
    }

    public override string ToString()
    {
        // We can be empty
        if (this.Empty)
        { return "(none)"; }

        // Build key name
        string keyName = Enum.GetName(typeof(Keys), this.keyCode);
        switch (this.keyCode)
        {
            case Keys.D0:
            case Keys.D1:
            case Keys.D2:
            case Keys.D3:
            case Keys.D4:
            case Keys.D5:
            case Keys.D6:
            case Keys.D7:
            case Keys.D8:
            case Keys.D9:
                // Strip the first character
                keyName = keyName.Substring(1);
                break;
            default:
                // Leave everything alone
                break;
        }

        // Build modifiers
        string modifiers = "";
        if (this.shift)
        { modifiers += "Shift+"; }
        if (this.control)
        { modifiers += "Control+"; }
        if (this.alt)
        { modifiers += "Alt+"; }
        if (this.windows)
        { modifiers += "Windows+"; }

        // Return result
        return modifiers + keyName;
    }

    public bool Empty
    {
        get { return this.keyCode == Keys.None; }
    }

    public bool Registered
    {

```

```
    get { return this.registered; }  
}
```

```
public Keys KeyCode  
{  
    get { return this.keyCode; }  
    set  
    {  
        // Save and reregister  
        this.keyCode = value;  
        this.Reregister();  
    }  
}
```

```
public bool Shift  
{  
    get { return this.shift; }  
    set  
    {  
        // Save and reregister  
        this.shift = value;  
        this.Reregister();  
    }  
}
```

```
public bool Control  
{  
    get { return this.control; }  
    set  
    {  
        // Save and reregister  
        this.control = value;  
        this.Reregister();  
    }  
}
```

```
public bool Alt  
{  
    get { return this.alt; }  
    set  
    {  
        // Save and reregister  
        this.alt = value;  
        this.Reregister();  
    }  
}
```

```
public bool Windows  
{  
    get { return this.windows; }  
    set  
    {  
        // Save and reregister
```

```
        this.windows = value;  
        this.Reregister();  
    }  
}  
}
```

**Jackie Chun** - *August 3, 2015, 6:32 pm*

Try the RegisterHotKey Function. You can probably find tons of examples of it.

Edit: the guy above me already gave u a registerhotkey function example lol