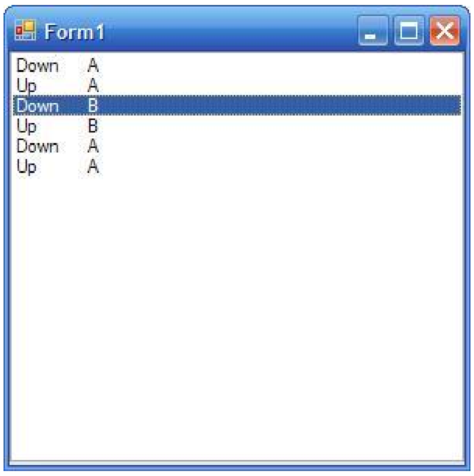
# A Simple C# Global Low Level Keyboard Hook

By **StormySpike**, 30 May 2007

★★★★★ 4.66 (61 votes)

**Download source - 29.62 KB**

**Download sample application - 4.59 KB**



```
Down    A
Up      A
Down    B
Up      B
Down    A
Up      A
```

## Introduction

This article discusses a class that I wrote that wraps a global low level keyboard hook. The sample hooks the A and B keys, just for demonstration.

## Background

I was trying to find a way for an application that I am writing to restore itself when a combination of keys was pressed. This was born from searching around for the answer.

## Using the Code

First download the source, and add *globalKeyboardHook.cs* to your project. Then add...

Collapse | Copy Code

```
using Utilities;
```

... to the top of the file you are going to use it in. Next add an instance of `globalKeyboardHook` to your class:

Collapse | Copy Code

```
globalKeyboardHook gkh = new globalKeyboardHook() ;
```

When a `globalKeyboardHook` is constructed, it automatically installs the hook, so all there is left to do is add some keys for it to watch, and define some event handlers for the `KeyDown` and `KeyUp` events. I usually do this on the main form's load event handler, like this:

About Article

A simple description and sample of creating a global low level keyboard hook in C#

| | |
|---|---|
| Type | Article |
| Licence | CPOL |
| First Posted | 30 May 2007 |
| Views | 297,257 |
| Downloads | 21,361 |
| Bookmarked | 126 times |

C#2.0 Windows .NET Visual-Studio Dev

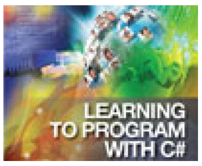
Bookmark Print Email

Top News

Salaries for Developers and Designers across the world

Get the Insider News free each morning.

Related Videos



Related Articles

Using Window Messages to Implement Global System Hooks in C#

Global Windows Hooks

Mouse and KeyBoard Hooking utility with VC++

Global System Hooks in .NET

⊟ Collapse | Copy Code

```
private void Form1_Load(object sender, EventArgs e) {
    gkh.HookedKeys.Add(Keys.A);
    gkh.HookedKeys.Add(Keys.B);
    gkh.KeyDown += new KeyEventHandler(gkh_KeyDown);
    gkh.KeyUp += new KeyEventHandler(gkh_KeyUp);
}

void gkh_KeyUp(object sender, KeyEventArgs e) {
    lstLog.Items.Add("Up\t" + e.KeyCode.ToString());
    e.Handled = true ;
}

void gkh_KeyDown(object sender, KeyEventArgs e) {
    lstLog.Items.Add("Down\t" + e.KeyCode.ToString());
    e.Handled = true ;
}
```

Here I have chosen to watch for the `A` and `B` keys, and defined handlers for the `KeyUp` and `KeyDown` events that both log to a listbox called `lstLog`. So whenever the user presses the `A` or `B` keys, no matter what has focus, the application will be notified. Setting `e.Handled` to `true` makes it so no other notifications for this event go out, in the sample, this effectively stops the user from typing an `A` or `B`. This can be useful in ensuring that key combinations are not also typed out when used.

You can add hooks for as many keys as you would like, just add them like above. Don't get frustrated if you add a hook for a key and it doesn't work, many of them, like `Keys.Shift` show up as other more specific keys, like `Keys.LShiftKey` or `Keys.RShiftKey`. `Keys.Alt` shows up as `Keys.LMenu` or `Keys.RMenu`, `Keys.Control` shows up as `Keys.LControl` or `Keys.RControl`, just to name a few.

If you would like to hook or unhook the keyboard hook at any point, just call your `globalKeyboardHook`'s `hook` and `unhook` methods, like so:

⊟ Collapse | Copy Code

```
//unhook
gkh.unhook()
//set the hook again
gkh.hook()
```

## Points of Interest

The bulk of the work in this code is done in the `globalKeyboardHook` class, although it is a fairly simple piece of code itself. The hardest part of doing this was finding the correct parameters for `SetWindowsHookEx`.

⊟ Collapse | Copy Code

```
[DllImport("user32.dll")]
static extern IntPtr SetWindowsHookEx
    (int idHook, keyboardHookProc callback, IntPtr hInstance, uint threadId);
```

The parameters were worked out to be:

⊟ Collapse | Copy Code

```
IntPtr hInstance = LoadLibrary("User32");
hhook = SetWindowsHookEx(WH_KEYBOARD_LL, hookProc, hInstance, 0);
```

The first parameter `WH_KEYBOARD_LL` is just saying that we want to hook the low level keyboard events, `hookProc` is the callback for the event, `hInstance` is a handle to *User32.dll*, where this event is first processed (I think). The last parameter is if you want to hook a specific thread, then you would just pass a thread id instead of using the `hInstance`.

## History

- 5/30/07 - First version

## License

Global Mouse and Keyboard Library

InputManager library - Track user input and simulate input (mouse and keyboard)

Lock Windows Desktop

Popup Window Finder and Mouse Tracker in C#

Processing Global Mouse and Keyboard Hooks in C#

Using keyboard hooks in WinCE

API hooking revealed

Keystroke Monitoring

HOWTO track a user's idle time

Your Desktop and Microsoft's SetWindowsHookEx()

Keyboard hooks in WinCE

System Wide Hooking for the WM_CHAR Message

Low-level Windows API hooks from C# to stop unwanted keystrokes

Hooking in Microsoft .NET

How to check for user inactivity with and without platform invokes in C#

KeyBoard Hooks