Define Preprocessor Symbols
Use preprocessor symbols for conditional compilation in C# and VB.NET, and add hotkey support in VB6.
by Juval Löwy and Karl E. Peterson

January 2003 Issue

Technology Toolbox: VB.NET, C#, VB6

Q: Define Preprocessor Symbols
In C and C++, I could define macros and compiler directives using the #define keyword and related compiler instructions. Can I do the same in C#? What about VB.NET?

A:
C# doesn't have macro support, because macros are a bad idea in general: You can't step through and debug them. In C++, you should use inline functions instead of macros, and constants instead of preprocessor definitions. This is why C# doesn't support macros or preprocessor definitions of constants. C++ preprocessor directives were useful for conditional compilation in order to exclude code sections from the build if some preprocessor symbol wasn't defined. C# does support preprocessor directives and conditional compilation, although not in the same way C++ does. You use the #define directive to define a compilation symbol in C#:

```
#define Something
```

This looks similar to C++, but with a few restrictions. The defined symbol must be the first token in the file, and you can only define symbols, not code macros or constants. Like C++, C# supports conditional definitions through the #if, #elif, #else, #endif, and #undef directives. Use these directives as you would in C++:

```
#if Something
   Trace.WriteLine
      ("Something is defined");
#else
   Trace.WriteLine
      ("Something is not defined");
#endif
```

You can nest directive usage, and you can even use the Boolean operators =, =, !=, &&, and || to construct a compound condition:

```
#if Something && SomethingElse
   Trace.WriteLine("Something "
      "and SomethingElse are defined");
#endif
```

Q&A (Continued)

You can use the project's Property Pages instead of defining symbols in the source files. Go to the Build item under the Code Generation group to specify what symbols are defined for this build configuration, instead of placing the definitions in code. By far, the most common use for conditional compilation is to modify the code's behavior when switching from debug to release builds. When creating a new C# project, Visual Studio .NET defines the DEBUG and TRACE symbols automatically for your use in debug builds, and only TRACE for release builds.

C# takes conditional compilation one step further than C++ by introducing the Conditional attribute defined in the System.Diagnostics namespace. This attribute makes the compiler exclude the method decorated with it from the build, if you don't define a condition:

```
#define MySpecialCondition

public class MyClass
{
   [Conditional("MySpecialCondition")]
   public void MyMethod()
   {...}
}
```

You can use any symbol you like, but you would use the DEBUG symbol typically. If you don't define the condition, the compiler automatically excludes from the build both the method and any reference to it in the client's code:

```
MyClass obj = new MyClass();
//This line is conditional
obj.MyMethod();
```

Having the compiler do the method call exclusion automatically is a major improvement over C++. In the past, when C++ developers did code exclusion manually and wanted to put the method calls back in, they sometimes forgot to do it in one or two places and caused a defect. You can use the Conditional attribute only on methods with a void return type, so that you can build when the condition isn't met. In addition, although the compiler doesn't complain, make sure to apply the attribute only if the conditional method has no outgoing parameters, so your code behaves correctly when the symbol isn't defined.

VB.NET uses preprocessor symbols much like C# does. The only difference is that you can't define the symbols in your code; you must do so in the project's Property Pages. —J.L.

Q: Add Hotkey Support
I want to create a program that usually runs in the background, but pops up and becomes the active task if the user presses (for example) Ctrl-1 at any time. Someone pointed me to the RegisterHotKey API. Can you tell me more about it? Is a subclass required in this case?

A:
You're on the right track. RegisterHotKey is the correct API—this call instructs the system to send you a notification if the user presses the indicated key or key combination. Classic VB doesn't give you access to the main thread's message queue, so the easiest way of sinking this notification is by subclassing the desired window's message stream and reacting to a WM_HOTKEY message.

My preference when subclassing is to have the messages dispatched from the generic BAS module sink (by necessity, due to AddressOf limitations) to a companion class that in turn notifies its associated form as needed. The details of this operation could fill an entire column, but luckily, various implementations abound on the Internet, and you can download my solution here. I've wrapped this specific task up in a CHotKey class, which you set

up like this:

```
' Member variables
Private WithEvents m_Hotkey As CHotKey

Private Sub Form_Load()
   ' Setup instance of hotkey tracking
   ' class, using Ctrl-1 as the trigger.
   Set m_Hotkey = New CHotKey
   m_Hotkey.hWnd = Me.hWnd
   m_Hotkey.SetHotKey vbKey1, _
      vbCtrlMask
End Sub
```

Each time the class sinks a WM_HOTKEY message, it notifies the form by raising a Hotkey event, and the form then brings itself front and center:

```
Private Sub m_Hotkey_Hotkey()
   ' Reposition, and bring forward.
   With Me
      .WindowState = vbNormal
      .Move (Screen.Width - .Width) \ _
         2,(Screen.Height - .Height) _
         \ 2
      .Show
   End With
End Sub
```

You could've wrapped these response details into the companion class just as easily, but they're as likely as not to change from application to application, so I felt they were better left within the individual form.

Q&A (Continued)

RegisterHotKey requires four parameters. The first parameter tells the system which window to notify when the user presses a hotkey. The second parameter is an identification value unique to your application. The third and fourth parameters identify the modifier key(s) and virtual key codes for your desired hotkey. The ID value and the modifier key value(s) are slightly counterintuitive in some aspects.

You can generate a system-wide unique value to use as your hotkey's ID value by calling the GlobalAddAtom API and passing a string representation of the current time. GlobalAddAtom returns a value in the range &hC000-&hFFFF, which is the documented range shared DLLs (as opposed to standalone EXEs) should use with RegisterHotKey. Fortunately, no ill effect seems to result from using this convenient number-generation scheme. Be sure to match a call to GlobalDeleteAtom for each call to GlobalAddAtom to avoid overfilling the global atom table.

Another potential problem exists in that the system-defined key modifiers—MOD_ALT, MOD_CONTROL, and MOD_SHIFT—have values that don't match their VB counterparts (ShiftConstants) directly. The Shift and Alt modifier values are flip-flopped in these two constant schemes, so you need to recombine the modifiers for the API if you want to offer the class consumer IntelliSense support for your Modifiers parameter (see Listing 1).

Hotkeys work on a first-come, first-served basis. If another application has registered your desired hotkey already, RegisterHotKey fails and Err.LastDllError contains 1409—the error code for ERROR_HOTKEY_ALREADY_REGISTERED. Your only option should that occur is to fall back to second, or even third, choices and alert your user. On NT-class systems, don't use F12 for your hotkey, because NT reserves this particular key for use by debuggers. —K.E.P.

About the Authors
Juval Löwy is a software architect and the principal of IDesign, a consulting and training company focused on .NET design and .NET migration. Juval is a Microsoft regional director for the Silicon Valley, working with Microsoft on helping the industry adopt .NET. He's the author of COM and .NET Component Services (O'Reilly). Contact him at www.idesign.net.

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the VSM Editorial Advisory Board. Online, he's a Microsoft MVP and a section leader on several DevX forums. Find more of Karl's VB samples at www.mvps.org/vb.