

# Coding Faster: Getting More Productive with Microsoft® **Visual Studio®**



Zain Naboulsi  
Sara Ford

# **Coding Faster: Getting More Productive with Microsoft® Visual Studio®**

**Zain Naboulsi**

**Sara Ford**

Published by Microsoft Press

*First and foremost to God and Jesus Christ for making this all possible. To my mom, Helen Naboulsi, for always encouraging me to go above and beyond to reach my goals, and to Russell Chandler for being the greatest nephew anyone could ever have.*

*Zain Naboulsi*

*Senior Developer Evangelist, Microsoft*

*To my parents, Jane and Louie Smolensky, for encouraging me to program on the computer when I was 5 years old, and to Beulah Bourgeois and Annabelle Fayard for being the best babysitters a little girl could ever hope for.*

*Sara Ford*

# Foreword

Visual Studio is quite possibly the most powerful and comprehensive software development suite available. No matter your discipline—developer, test, architect, etc.—Visual Studio provides the tools you need to help get your job done.

However, Visual Studio handles such a large number of development tasks for so many platforms that learning it can be an overwhelming process. Many learn just enough to solve the problem at hand but don't delve deep enough to unearth the gems that enable real productivity.

Visual Studio contains numerous features and options that can help you perform your tasks more effectively. Some are prominently advertised, but many of the real time-savers are buried in obscure dialog boxes and triggered with arcane keyboard shortcuts. While most of this information can be learned by wading through hundreds of pages of documentation, many of the more powerful features are yet undocumented. How can we possibly navigate the vast forest that is Visual Studio? A guide is needed.

Fortunately, we have two.

For several years, Sara Ford has championed productivity with her highly successful Visual Studio “Tip of the Day” blog. Between July 2007 and December 2008, Sara blogged nearly 400 Visual Studio tips and tricks that were essential for many of us (myself included) to get closer to attaining Visual Studio mastery. After Sara completed her journey with Visual Studio, Zain Naboulsi picked up the torch. Zain continues to blog Visual Studio tips and tricks, digging into features in the latest releases and covering some of the popular Visual Studio extensions that are available.

I vividly remember my first encounter with Zain. I had just joined Microsoft as a Program Manager on the Visual Basic and C# IDE experiences and received an email from Zain saying that he was taking the mantle from Sara and starting a Visual Studio tips and tricks blog. My first thought was, who is this guy? Sara was an alumnus of the Visual Studio team and had a great deal of “inside knowledge” to share. How could Zain go to the same level of depth that she had? I wasn’t prepared for my initial impression to be shattered so thoroughly.

It didn’t take long for me to realize that Zain really knows his stuff. After that first email, Zain kept in touch regularly with me and other members of the Visual

Studio team. As he systematically pulled away the layers of Visual Studio to find the golden nuggets of productivity beneath, he would ask questions or confirm the tips that he found. Often, Zain would find features that I didn't even know existed. In some cases, he even found bugs where something had been unintentionally left in the product (e.g., the infamous `Debug.cleartextonfoo` command).

What you hold in your hands is the crème de la crème of the sum of Zain and Sara's Visual Studio knowledge. In these pages, you will find a sure compass to help navigate the treacherous peaks and vast oceans of Visual Studio. By putting these tips, tricks, and techniques into practice, you'll grow closer to attaining Visual Studio mastery and learn to travel in style.

*Dustin Campbell*

*Program Manager, Visual Studio*

It's hard to imagine but if Sara Ford had her way, there never would have been a "Visual Studio Tip of the Day" blog. Back in 2005, we were colleagues on the Visual Studio Editor team who shared an office and a passion for making developers more productive. We both became intrigued by an email with customer feedback, which was remarkable because all of its suggested features were already in Visual Studio but the customer hadn't discovered them. We realized that many of the great productivity features that we developed in Visual Studio 2005 such as Code Snippets and Smart Tags would go unnoticed by many developers who weren't looking for them. We brainstormed several different ways that we could help customers discover all of the hidden functionality in Visual Studio until we arrived at the ultimate solution: Putting a Visual Studio Tip of the Day on the Start Page.

At the time, the Start Page was being completely rewritten to include an RSS feed which would be the perfect mechanism for users to learn how to use Visual Studio better one tip at a time. We lobbied hard to have the Start Page point to a feed of Tips & Tricks for Visual Studio. Unfortunately, there was too much skepticism that there were enough tips to generate new content every day and so the idea was rejected.

Defeated but undeterred, Sara was determined to demonstrate that not only were there enough hidden gems but that there was a huge audience for a "Tip of the Day." She had recently started blogging and challenged herself to blog every workday until she ran out of tips. Initially, we wrote a list of about 50 different tips before she set out on her challenge. Over the following months and years, Sara has worked tirelessly to find hundreds of useful tips, created an engaging blog and helped thousands of developers become better users of Visual Studio.

As we were putting the finishing touches on Visual Studio 2010 last year, I was reassured to find that Zain Naboulsi had stepped in to create the next generation of the Visual Studio Tips and Tricks blogs. He's engaged directly with the Visual Studio team to highlight each of the new features that were introduced in the latest release and the greatest from previous versions.

Based on their years of experience evangelizing Visual Studio Tips and Tricks, "Coding Faster" distills this knowledge into one easy-to-read book which will make you a better user of Visual Studio. Using the described techniques, you'll learn to write code with fewer keystrokes, manage projects and documents with ease and powerful debugging techniques. It also introduces Visual Studio Extensibility which allows you to create your own extensions or find those provided by the community.

In retrospect, it's fortunate that the tip of the day didn't make it onto the Start Page as it led Sara and Zain to write great blogs and this brilliant book to help everyone code faster.

*Sean Laberee*

*Lead Program Manager, Visual Studio*

# Introduction

Visual Studio is sexy. In the world of Integrated Development Environments (IDEs), it stands as a beautiful example of how environments should work. Yet many of the features created to improve productivity, I believe, are largely neglected. Most developers use only a small percentage of the capabilities in this wonderful product—not because they don’t want to use them, but because developers don’t know they exist.

In most books that address the various .NET languages or technologies, Visual Studio seems to be mentioned almost as an afterthought; to be fair, its focus is primarily on the language or technology that’s the subject of the book, not the IDE—which is as it should be. On the other side of the coin, books written about Visual Studio do focus on the product, but tend to be broad in scope, describing features, but without saying much about their actual use.

The goal of this book is to arm you with techniques that you can apply immediately to improve productivity. Use the content in this book anywhere, anytime, to dramatically reduce the time required to perform just about any task in Visual Studio. You won’t find an exhaustive treatment of every feature in Visual Studio in this book, but it contains sufficient coverage that we’re sure you’ll find something useful, regardless of how you use the product.

This is much more than just a tips and tricks book. Within these pages are—for the first time ever—the keyboard mapping shortcuts, commands, and menu paths for features, along with detailed descriptions of how to use them. We worked very hard to present the information in a way that makes the book easy to read cover-to-cover or as a quick reference.

## **Who Should Read This Book**

If you use Visual Studio 2005, 2008, or 2010, you should read this book. There are over 365 tips in this book (including the additional online Appendix), all selected with the single goal of helping you be more productive by showing you how to use Visual Studio features. The contents in this work are great on their own or as a perfect complement to any course, book, or other learning tool as you explore Visual Studio.

## **Assumptions**

This book assumes you have, at a minimum, Visual Studio Professional 2005, 2008, or 2010 installed. Specifically, it covers techniques that can be used in Visual Studio as well as examples in C++, C#, and/or VB where appropriate.

With a heavy focus on helping you get work done faster, we assume that you have a basic understanding of how to use Visual Studio, and have had exposure to one of the many languages supported in the product.

# Organization of This Book

This book is divided into two sections. **Part I**, written by me, Zain Naboulsi, called “Productivity Techniques,” provides information that can be used in your daily work with Visual Studio; these techniques range from very easy to quite advanced methods of using the product. The chapters are organized to take you through the most common daily tasks you perform. Within each chapter the information is arranged, essentially in order, from beginning to more advanced optimizations:

- **Chapter 1**, shows key skills to have when starting up and using Visual Studio.
- **Chapter 2**, shows ways to create and use projects and items more effectively.
- **Chapter 3**, gives guidance on how to organize the environment to best advantage.
- **Chapter 4**, illustrates how best to navigate and manipulate document windows.
- **Chapter 5**, is a collection of tips on how to find just about anything in your code or code related in Visual Studio.
- **Chapter 6**, the largest chapter, shows a host of techniques for using the editor more efficiently.
- **Chapter 7**, the second largest chapter, shows you great techniques to improve your debugging experience within the IDE.

**Part II**, written by Sara Ford and other authors, contains an examination of selected extensions from the Visual Studio Gallery that you can install to further accelerate the Visual Studio experience.

Finally, we have included an entire second book of tips in **Appendix B** (downloadable at <http://go.microsoft.com/fwlink/?LinkId=223758>) that were cut from the main book so that we could keep the print size manageable. You are literally getting two books for the price of one.

## Finding Your Best Starting Point in This Book

Each chapter—and in fact, almost every item—in **Part I** of this book stands on its own, so feel free to begin reading wherever you like. However, I suggest that you start by choosing the tasks that will have the greatest impact in your daily work. If most of your day is spent debugging, then start with **Chapter 7**, first. After you have a good handle on using the features built-in to Visual Studio from **Part I** of

this book, look in [Part II](#) to see if there's an extension that can help you advance your goals even further. Visual Studio has many extensions that can improve your overall experience.

# Conventions and Features in This Book

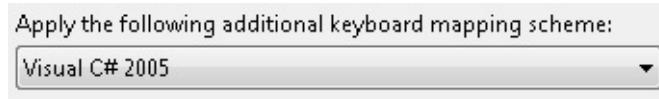
This book presents information using conventions designed to make the information readable and easy to use.

## Keyboard Settings

Throughout this book I refer to the keyboard settings often, so it is important to know the connection between development settings and keyboard mapping schemes. When first installed, Visual Studio asks you to choose a collection of settings, as shown here:



The settings chosen are directly related to Tools | Options | Environment | Keyboard within the “Apply the Following Additional Keyboard Mapping Scheme” dropdown list. For example, when you choose Visual C# Development Settings, you will see Visual C# 2005 as the keyboard mapping, as shown here:



The following table lists the setting collection and its corresponding keyboard mapping scheme:

SETTINGS	ADDITIONAL KEYBOARD SCHEME
GENERAL DEVELOPMENT SETTINGS	(Default)
PROJECT MANAGEMENT SETTINGS	(Default)
VISUAL BASIC DEVELOPMENT SETTINGS	Visual Basic 6
VISUAL C# DEVELOPMENT SETTINGS	Visual C# 2005
VISUAL C++ DEVELOPMENT SETTINGS	Visual C++ 6
VISUAL F# DEVELOPMENT SETTINGS	(Default)

<b>WEB DEVELOPMENT</b>	(Default)
<b>WEB DEVELOPMENT (CODE ONLY)</b>	(Default)

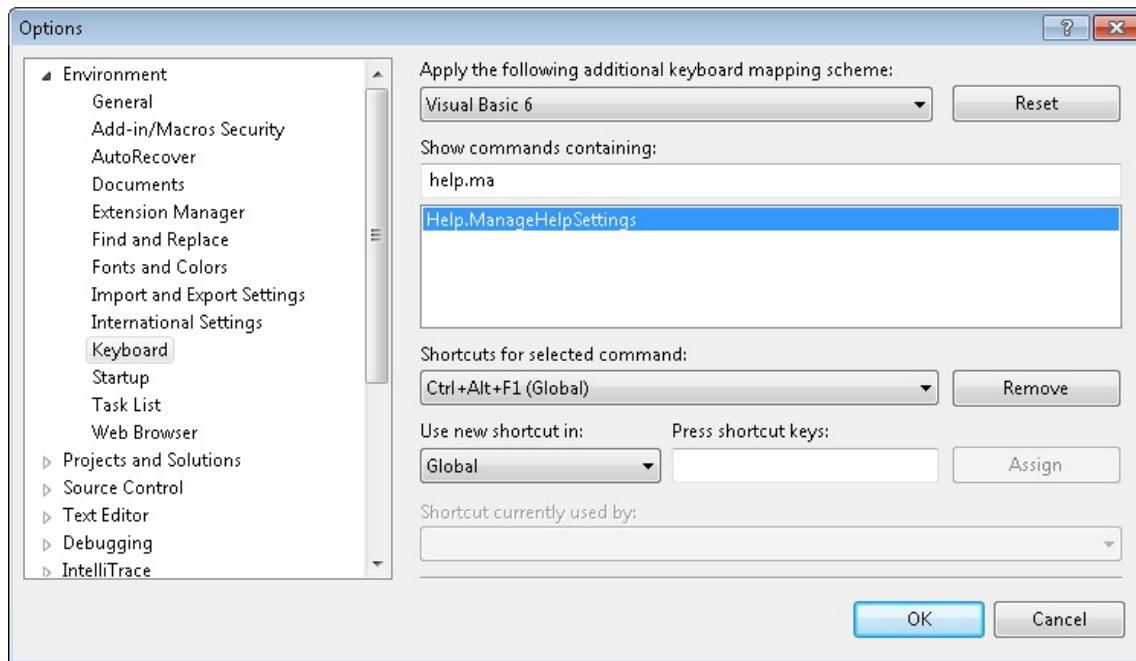
## Summary Information

One of the unique features of this book is the summary information at the top of every item. All tips will contain a table with one or more pieces of summary information that looks like this example:

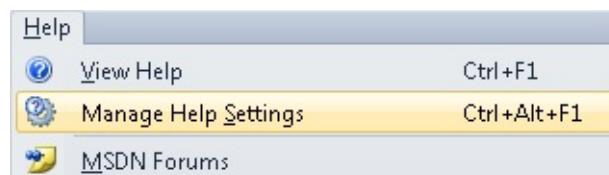
<b>DEFAULT</b>	<b>Ctrl+Alt+F1 (help settings); F1 (view help)</b>
<b>VISUAL BASIC 6</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>Visual C# 2005</b>	Ctrl+F1, M (help settings); Ctrl+F1, Ctrl+M (help settings); F1 (view help)
<b>VISUAL C++ 2</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>VISUAL C++ 6</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>WINDOWS</b>	Alt, H, S (help settings); Alt, H, V (view help)
<b>MENU</b>	Help   Manage Help Settings; Help   View Help
<b>COMMAND</b>	Help.ManageHelpSettings; Help.F1Help
<b>VERSIONS</b>	2010
<b>LANGUAGES</b>	All
<b>CODE</b>	vstipTool0120

Here is what each piece of information means:

- **Default to Visual Studio 6**—Keyboard shortcuts assigned that are mapped to the choice made for development settings. These settings can be found at Tools | Options | Environment | Keyboard. If absent, implies there are no keyboard shortcuts that apply.



- **Windows**—Keyboard shortcuts that navigate the Menu Bar for commands. If absent, implies there are no Menu Bar shortcuts that apply.



- **Menu**—Menu Bar path for using a command. Help | Manage Help Settings means click on the Help menu and choose Manage Help Settings item underneath it. If absent, implies the item can not be accessed from the Menu Bar.
- **Command**—Visual Studio command used to assign keyboard shortcuts, aliases, run macros, etc. Found at Tools | Options | Environment | Keyboard. If absent, implies there is no command available for this activity.
- **Versions**—Versions of Visual Studio that support the information given in the tip. If absent, implies all versions are supported.
- **Languages**—Languages supported (C++, C#, and/or VB). If absent, implies all languages are supported.
- **Code**—Unique identifier for each tip for looking up references to the tip in the book, online, etc.

## Additional Information

- Boxed elements with labels such as “Warning” are used to tell you about items that may impact you negatively. Be aware that these activities are done at your own risk.
- Boxed elements with labels such as “Note” provide additional information or alternative methods for completing a step successfully.
- Text that you type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, “Ctrl+Alt+L” means that you hold down the Ctrl key while you press the Alt key and the L key.
- A comma (,) between key names means you press each key separately. For example, “Alt+T, O” means you hold down Alt while you press T then let up on the keys and finally press O by itself.
- A vertical bar between two or more menu items (e.g., File | Close), means that you should select the first menu or menu item, then the next, and so on.

# **System Requirements**

You will need the following hardware and software to complete the practice exercises in this book.

## **Software Requirements**

- Windows XP (x86) with Service Pack 3—all editions except Starter Edition
- Windows Vista (x86 & x64) with Service Pack 2—all editions except Starter Edition
- Windows 7 (x86 & x64)
- Windows Server 2003 (x86 & x64) with Service Pack 2
- Windows Server 2003 R2 (x86 & x64)
- Windows Server 2008 (x86 & x64) with Service Pack 2
- Windows Server 2008 R2 (x64)

## **Supported Architectures:**

- 32-Bit (x86)
- 64-Bit (x64)

## **Hardware Requirements**

- Computer that has a 1.6GHz or faster processor
- 1 GB (32-Bit) or 2 GB (64-Bit) RAM (Add 512 MB if running in a virtual machine)
- 3 GB of available hard disk space
- 5400 RPM hard disk drive
- DirectX 9 capable video card running at 1024 x 768 or higher-resolution display
- DVD-ROM Drive

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio.

# Acknowledgments

From day one this book has been a community-driven effort. The readers of my (Zain) and Sara's blogs have been a constant source of content, comments, and ideas. Our heartfelt thanks for all our readers have done to make this book a reality.

I used to make fun of those people who win awards on TV because they always have a huge list of people to thank and never seem to get through them. It looks like it's my turn now and I *know* that I will forget someone, so let me just say that behind every effort like this you will always have a great deal of people helping you in one way or another. Below is just a partial list of people Sara and I want to thank for contributing, directly or indirectly, to the effort:

Russell Jones and Adam Zaremba—Editors at O'Reilly Media, who herded the cats to make this book happen.

Kevin Stevens—Who came up with the name of the book and was instrumental in the technical review process.

Paul Millsaps, Bill Needels—For doing some of the technical review for the book.

Sean Laberee—Senior Program Manager Lead at Microsoft who helped both Sara and me get started with Tips and Tricks.

Dustin Campbell—Program Manager at Microsoft who continues to be a constant source of information when I get stuck on a feature or concept.

Brittany Behrens—Program Manager at Microsoft who helped me during those first tenuous days after I took over Sara's work.

Matt Manela—for writing the content for the Snippet Designer extension.

Andrew Steele—for writing the content for the Productivity Power Tools extension.

Jim Christopher—for writing the content for both the GhostDoc and the StudioShell extensions.

Terry Leeper—Principal Architect, Windows C++ Team, my main contact with the C++ folks and a good friend that has helped me resolve questions about features since I started doing the tips.

Lisa Feigenbaum and Beth Massi—Program Managers at Microsoft who constantly provided guidance and support as the content of the book evolved.

Brian Moore—Director, DPE Central Region, for providing support and being a great manager.

Clint Edmonson—Senior Architect Evangelist at Microsoft who I have toured with throughout the country delivering Visual Studio talks to thousands of people.

Phil Wheat—My best friend at Microsoft and a constant source of information. Phil is easily the smartest person I know.

Jared Bienz, Mike Azocar—Very good friends who have been great to bounce ideas around as I worked on the book.

Rob Bagby, Mike Benkovich, John Weston, Keith Combs—My old Microsoft Across America buddies.

Finally, Sara Ford wishes to thank Dr. Terrance Delaney and Dr. Michael McMurray for fixing her chronic shin splints during the course of writing this book.

## **Errata & Book Support**

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site at [oreilly.com](http://oreilly.com):

<http://go.microsoft.com/fwlink/?LinkId=226221>

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at [mspin@microsoft.com](mailto:mspin@microsoft.com).

Please note that product support for Microsoft software is not offered through the addresses above.

## We Want to Hear from You

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

<http://www.microsoft.com/learning/booksurvey>

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

## **Stay in Touch**

Let's keep the conversation going! We're on Twitter:  
<http://twitter.com/MicrosoftPress>

# Part I. Productivity Techniques

## In this part:

[Chapter 1](#)

[Chapter 2](#)

[Chapter 3](#)

[Chapter 4](#)

[Chapter 5](#)

[Chapter 6](#)

[Chapter 7](#)

# Chapter 1. Getting Started

*“A beginning is the time for taking the most delicate care [...]”*

— Frank Herbert “Dune”

This chapter addresses tasks that would be immediately beneficial as you work in Visual Studio. The main themes here are exporting your development settings, learning the Start Page, adjusting your performance, and other key tasks.

This chapter is arguably the most important one you will read in this book—and yet, I suspect, the one people will think they need the least. If you have been using Visual Studio for any length of time, you might easily feel that the tasks in this chapter have little application to your situation. But whether you have been using Visual Studio for ten days or ten years, these tips will help ensure that all your other efforts go smoothly, so taking time to absorb the contents is definitely worthwhile.

## 01.01 Running Multiple Versions of Visual Studio Side-By-Side

<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0054

People often ask whether you can run multiple versions of Visual Studio side-by-side on the same machine. The answer is yes you can!

You can find documentation on MSDN, in the topic “Installing Visual Studio Versions Side-by-Side,” at <http://msdn.microsoft.com/en-us/library/ms246609.aspx>.

The recommendation is that you install multiple versions from oldest to newest. So you would install Visual Studio 2005, 2008, and then 2010—in that order.

## 01.02 Getting Table of Contents in Visual Studio 2010 Online Help

<b>DEFAULT</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>VISUAL BASIC 6</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>VISUAL C# 2005</b>	Ctrl+F1, M (help settings); Ctrl+F1, Ctrl+M (help settings); F1 (view help)
<b>VISUAL C++ 2</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>VISUAL C++ 6</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+F1 (help settings); F1 (view help)
<b>WINDOWS</b>	Alt, H, S (help settings); Alt, H, V (view help)
<b>WINDOWS KEYBOARD</b>	Alt, H, S (help settings); Alt, H, V (view help)
<b>MENU</b>	Help   Manage Help Settings; Help   View Help
<b>COMMAND</b>	Help.ManageHelpSettings; Help.F1Help
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0120

I have to admit I don't like the new online help in Visual Studio 2010. Not that I think it's bad per se, but I was just used to the old help system's look and feel—particularly the table of contents list.

If you are like me and want to get that classic help look-and-feel back for online help, you need to do two things.

### Online Help

First, you need to set your default help to online help (you need Internet connectivity to use this feature) by selecting Help | Manage Help Settings. Then click Choose Online Or Local Help.



Select I Want To Use Online Help, and click OK.

- I want to use online help  
 I want to use local help

## Using Classic View

Now that you are using online help, Go to Help | View Help to see a page similar to the following:

Visual Studio 2010

Home   Library   Learn   Downloads   Support   Community

Search MSDN with Bing

MSDN Library

↑ Development Tools and Languages

Visual Studio 2010

- Visual Studio
- Visual Studio Application Lifecycle Mana...
- Visual Studio Feature Packs
- Visual Studio LightSwitch
- Technical Articles
- Featured Books on Visual Studio 2010

Community Content

Ilike VS but...  
As I'm fairly new to website dev...

Visual Studio 2010

This document provides resources for learning how to use applications.

Welcome to Visual Studio 2010

Learn about Visual Studio 2010:

- How to Obtain Visual Studio
- Visual Studio 2010 Product Highlights

In the upper-right corner of the page, if you see the Preferences link, click it.



**NOTE**

You may not see the Preferences link but instead just three links to Lightweight, ScriptFree, and Classic. In this case, just click Classic and skip the next step.

Choose Classic and click OK.

**Choose View**



Choose between a Lightweight experience that balances both richness and speed, a fast ScriptFree experience, or a feature-rich Classic experience.

**Lightweight**

A balance between a fast and rich experience.

**ScriptFree**

Super-fast with no script.

**Classic**

Classic. Feature rich view. It includes a side-by-side view of translated and English for some content.

**OK**

**Cancel**

Now your help will use the old-style contents list.

The screenshot shows the MSDN Library interface. On the left is a navigation tree:

- MSDN Library
  - Design Tools
  - Development Tools and Languages
    - Visual Studio 2010
      - Visual Studio
      - Visual Studio Application Lifecycle
      - Visual Studio Feature Packs
      - Visual Studio LightSwitch
      - Technical Articles
      - Featured Books on Visual Studio 2010
    - Visual Studio 2008
    - Visual Studio 2005
    - Visual Studio .NET

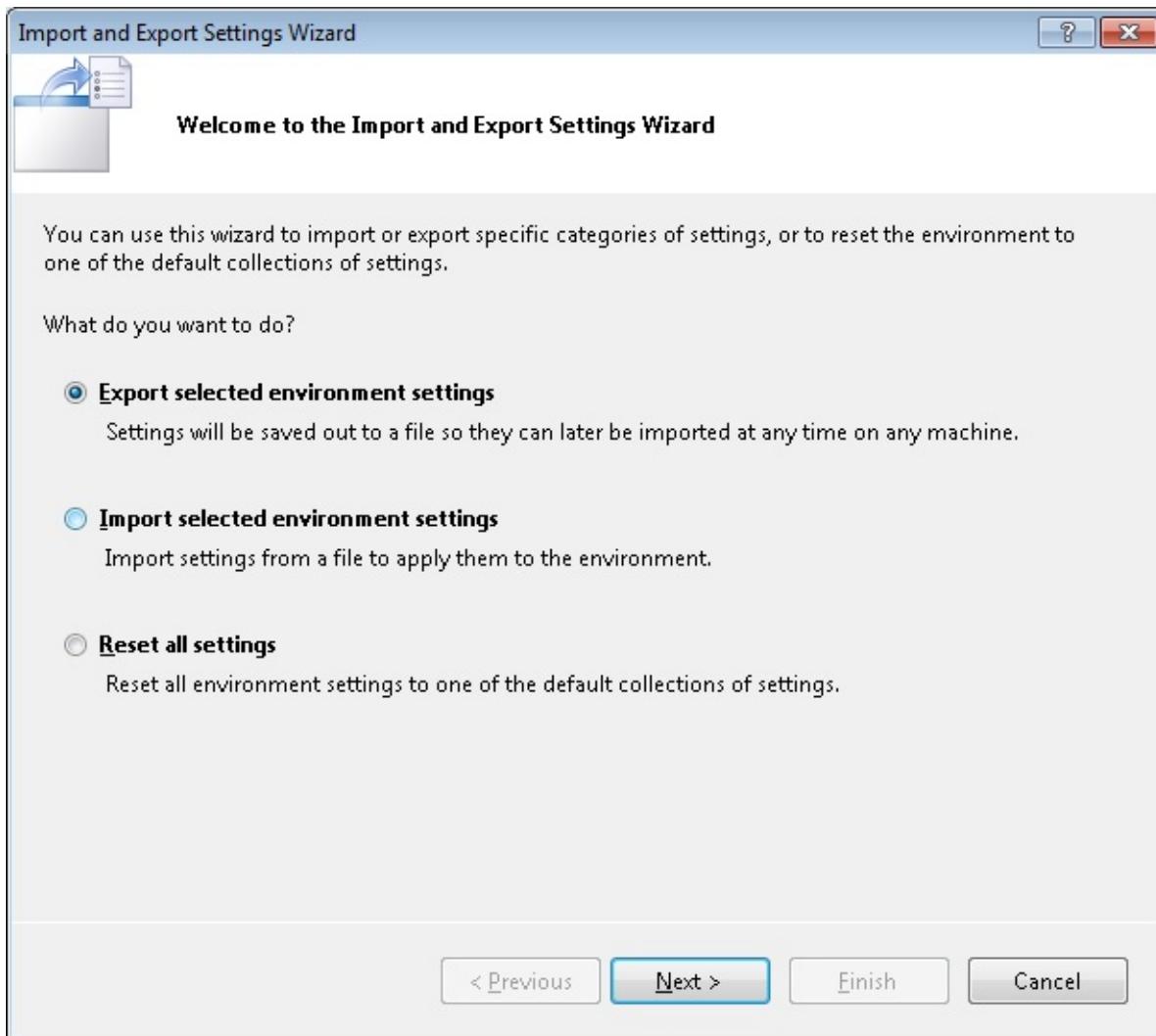
The main content area displays the Visual Studio 2010 documentation. The title is "Visual Studio 2010" and the subtitle is "Welcome to Visual Studio 2010". Below the title, it says "This document provides resources for learning how to use Visual Studio applications." There is a large "Visual Studio" logo icon. To the right of the logo, there is a list of links under "Welcome to Visual Studio 2010":

- How to Obtain Visual Studio
- Visual Studio 2010 Product Highlights

## 01.03 Exporting Your Environment Settings

<b>WINDOWS</b>	Alt,T, I
<b>MENU</b>	Tools   Import and Export Settings
<b>COMMAND</b>	Tools.ImportandExportSettings
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0021

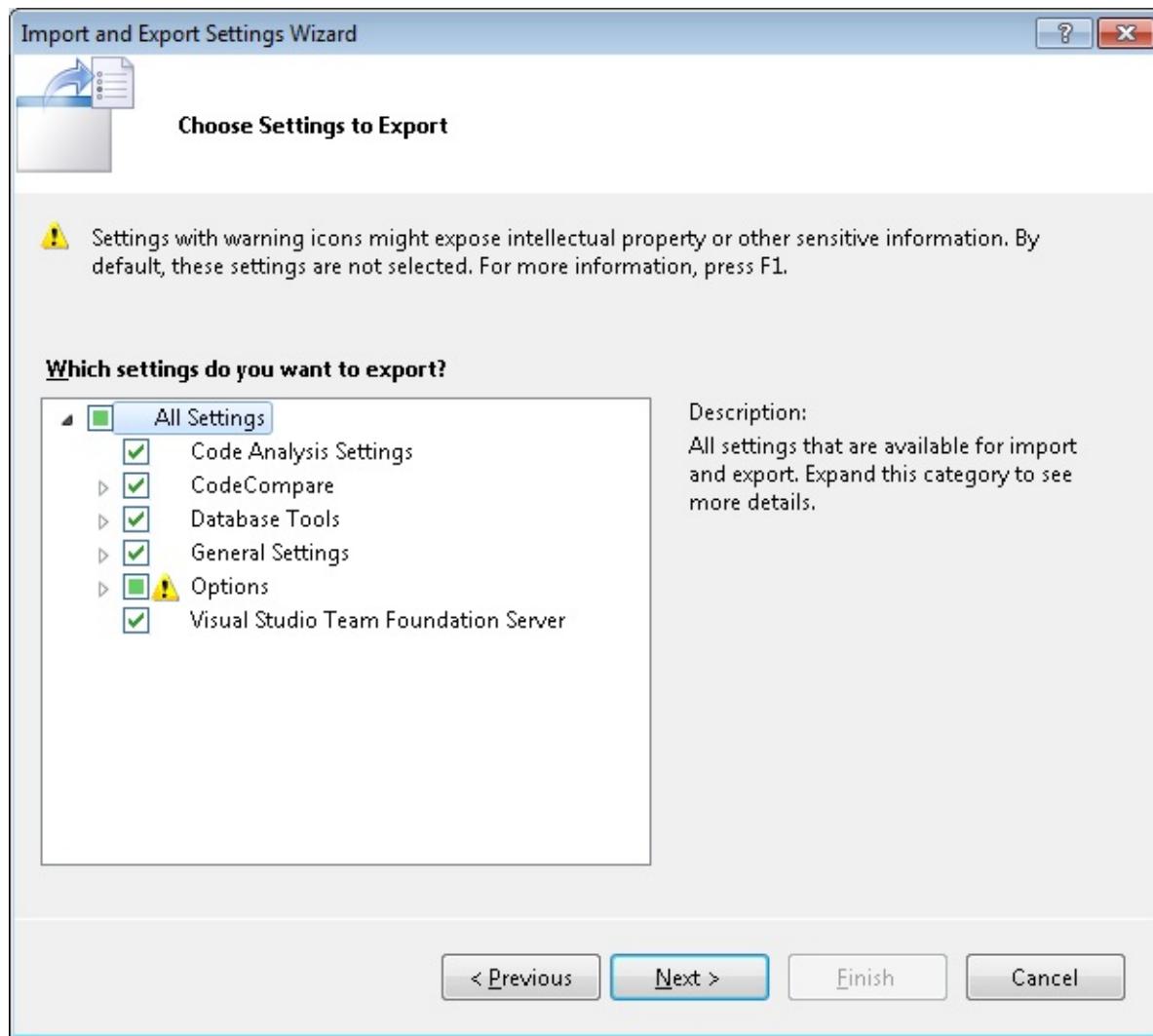
Exporting your environment settings is a great way to back them up. You can export your settings by selecting Tools | Import And Export Settings Wizard.



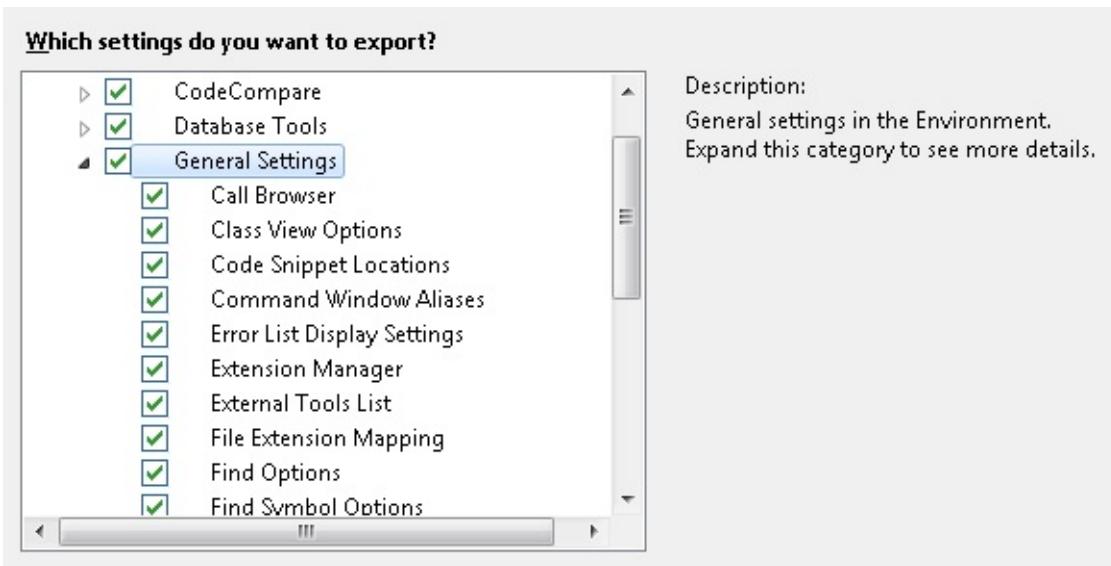
The Export Selected Environment Settings option lets you save your settings to a

.vssettings file.

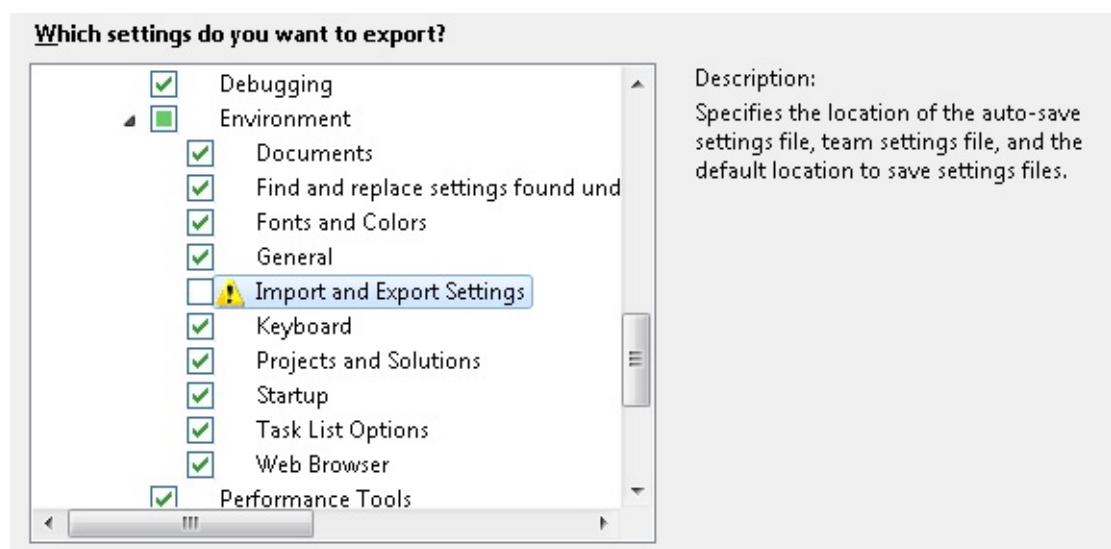
Click Next to see the Choose Settings To Export dialog box.



You can expand the areas to choose the items you want to include or exclude.



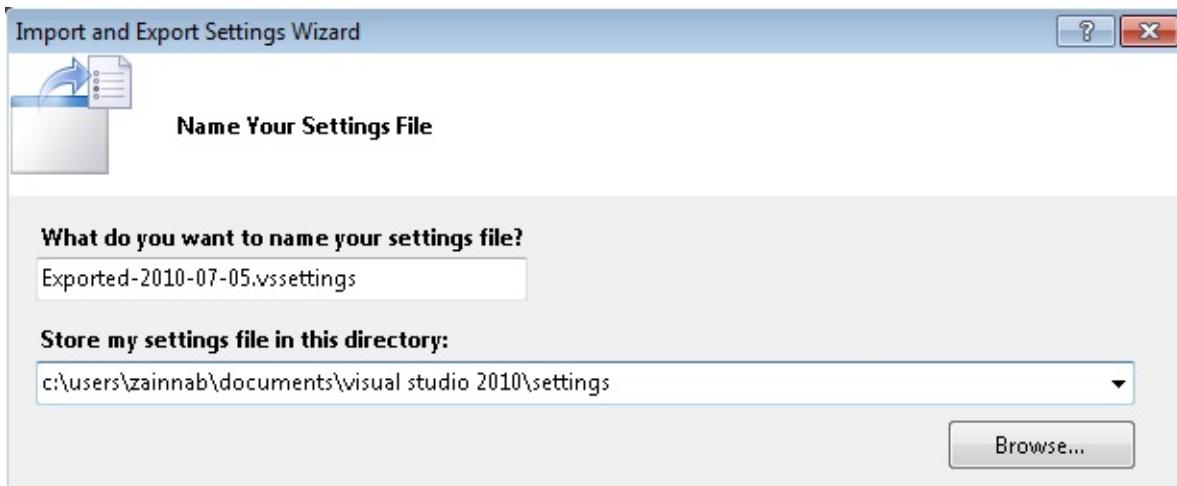
By default, almost everything is selected except for items that could expose sensitive information. You can tell which options these are by the yellow warning symbol icon next to the item.



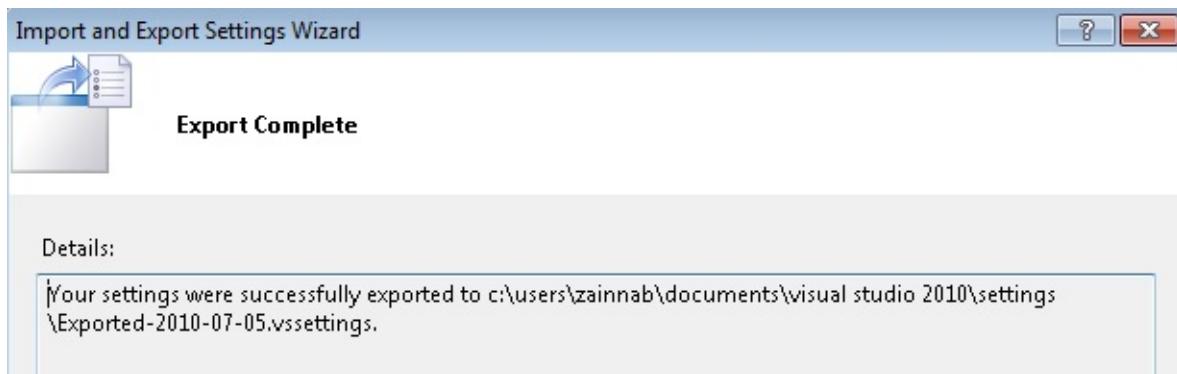
After you make your choices and click Next, you can pick the location (C:\users\<current user>\documents\visual studio <version>\settings, by default) and the filename (the current date, by default) where you want to save the exported information.

#### WARNING

If you don't give your exported settings good names it will be hard to figure out what they are for later. For example, if you are just exporting your favorite black theme fonts and colors, a name like "Fonts and Colors (Black Theme) 2010-07-05" would make sense.



When you click Finish, Visual Studio exports your settings, and the following dialog box appears.



If you are curious, the exported file is just an XML file. You can open it in Notepad and see the contents, as shown in the following illustration.

Exported-2011-06-18.vssettings - Notepad

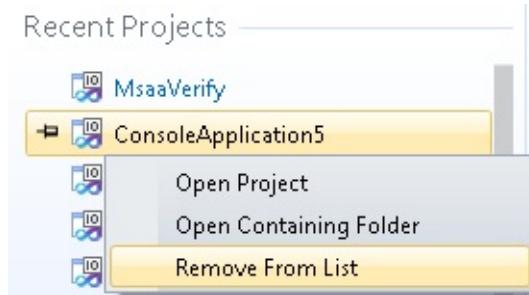
```
<UserSettings>
    <ApplicationIdentity version="10.0"/>
    <Tooloptions>
        <TooloptionsCategory name="Environment"
RegisteredName="Environment">
            <TooloptionsSubCategory name="Documents"
RegisteredName="Documents" PackageName="Visual Studio Environment Package">
                <PropertyValue
name="ShowMiscFilesProject">false</PropertyValue>
                <PropertyValue
name="AutoloadExternalChanges">false</PropertyValue>
                <PropertyValue
name="CheckForConsistentLineEndings">true</PropertyValue>
                <PropertyValue
name="SaveDocsAsUnicodeWhenDataLoss">false</PropertyValue>
                <PropertyValue
name="InitializeOpenFileFromCurrentDocument">true</PropertyValue>
                <PropertyValue
name="ReuseSavedActiveDocWindow">false</PropertyValue>
                <PropertyValue
name="DetectFileChangesOutsideIDE">true</PropertyValue>
                <PropertyValue
name="DontShowGlobalUndoChangeLossDialog">true</PropertyValue>
                <PropertyValue
name="AllowEditingReadOnlyFiles">true</PropertyValue>
                <PropertyValue
name="DocumentDockPreference">1</PropertyValue>
                <PropertyValue
name="MiscFilesProjectSavesLastNItems">0</PropertyValue>
            </TooloptionsSubCategory>
            <TooloptionsSubCategory name="FindAndReplace"
RegisteredName="FindAndReplace" PackageName="Visual Studio Environment Package">
                <PropertyValue
name="ShowWarningMessages">true</PropertyValue>
```

## 01.04 Remove Projects from the Recent Projects List

<b>WINDOWS</b>	Alt,F, J, [Number]
<b>MENU</b>	File   Recent Projects and Solutions
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0017

In Visual Studio 2010, you can now remove projects from the Recent Projects list on your Start Page.

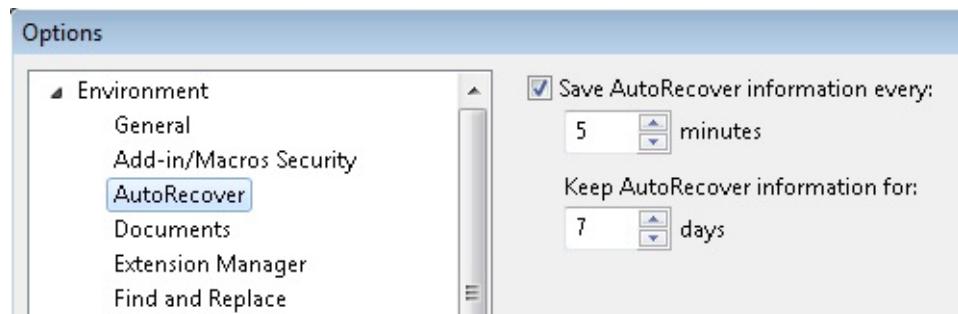
Just right-click the project, and select Remove From List, as shown in the following illustration. That's it. The project is removed from the list but not deleted. If you want to permanently delete the project, you need to do that yourself from the filesystem.



## 01.05 AutoRecover

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Environment   AutoRecover
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0019

AutoRecover can be a real life saver if the development environment crashes or if a power outage occurs. It's simple to use: Just go to Tools | Options | Environment | AutoRecover.

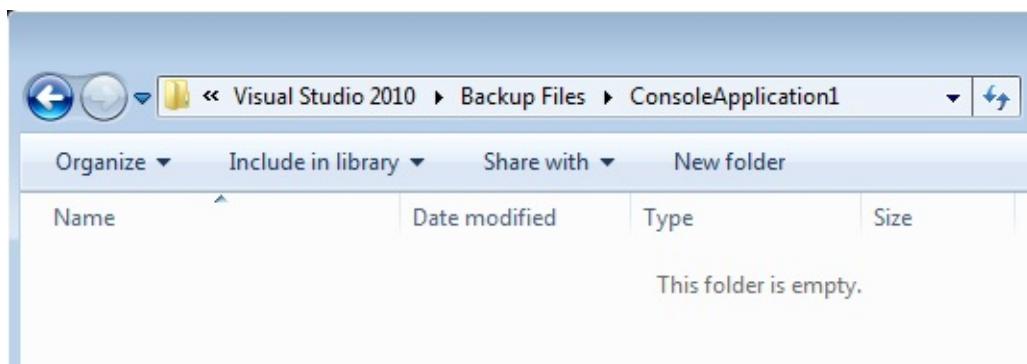


To turn this feature off (not recommended), you can clear the Save AutoRecover Information Every check box. Here's an explanation of what the other options do:

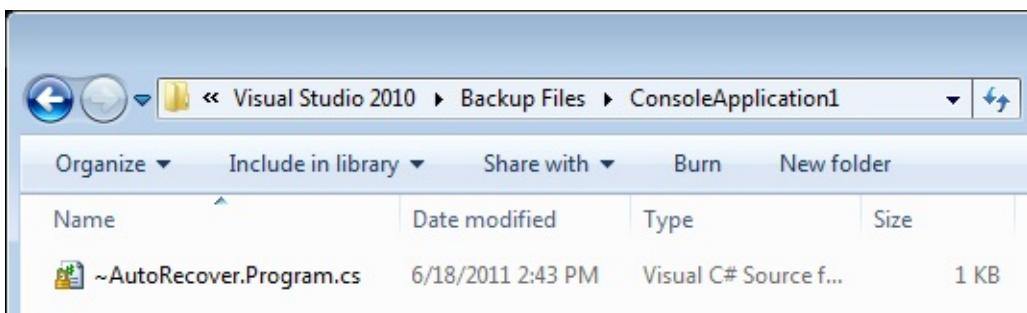
- **n minutes** Determines how often Visual Studio saves AutoRecover information for files. The default is to save every five minutes, but you can adjust that interval up or down depending on your needs. There is an inverse relationship between this value and the frequency of your updates to code. If you make frequent code updates, you should set a smaller save interval. Conversely, if you make relatively infrequent code updates, you can increase this interval. It's better to err by using an interval that's too short rather than too long; in other words, it's better to take a performance hit from file I/O than to lose a ton of work.
- **n days** Determines how long Visual Studio keeps AutoRecover files in the Backup Files directory. The default is seven days, which is adequate for most situations. If you work with a lot of projects over a short period of time, you might want to decrease this number to keep the Backup Files directory from getting too cluttered. If you're not sure about what you need for this value, it's better to guess high and later reduce the number as needed.

I want to be clear about what exactly gets saved and where it gets saved. First, recovered files are stored at My Documents\Visual Studio <version>\Backup Files\<projectname>. But not every file is saved here. The backup folder is empty when you first create a solution in Visual Studio.

When I make a change to a file and save the change, I wait five minutes to see the result.

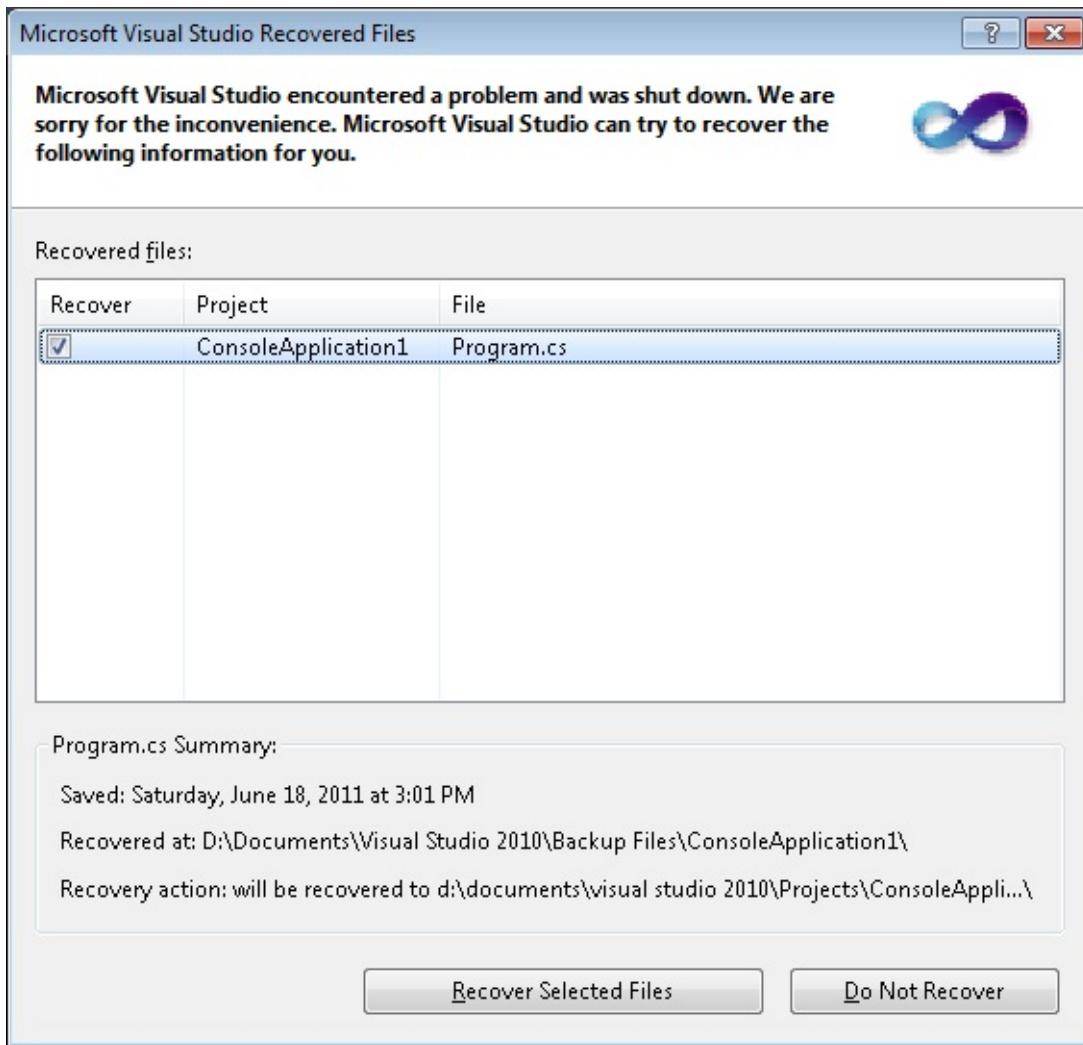


There's still nothing there, because Visual Studio knows there is no need to recover a saved file. But if I make a change to a file without saving it and wait another five minutes, here is what you see:



At this point, the AutoRecover information appears because there are unsaved changes. If Visual Studio crashes now, you would need to make a decision about whether to recover the unsaved changes or keep the last saved version. Giving you that choice is the essential function of the AutoRecover feature.

When you do finally have to recover a file, you will see the following dialog box.



To explain the terminology in the preceding dialog box:

- **Recovered files**

Lists the file(s) that can be recovered. Use a check box to select or clear the files you want to keep as well as to see some basic information.

- **<File Name> Summary**

Shows detailed information about the currently selected file, including date/time information, location of the backup file, and the destination location where the recovered file will be saved.

- **Recover Selected Files**

Performs a recovery action on the selected file(s), copying the recovered source file to the previously indicated destination.

- **Do Not Recover**

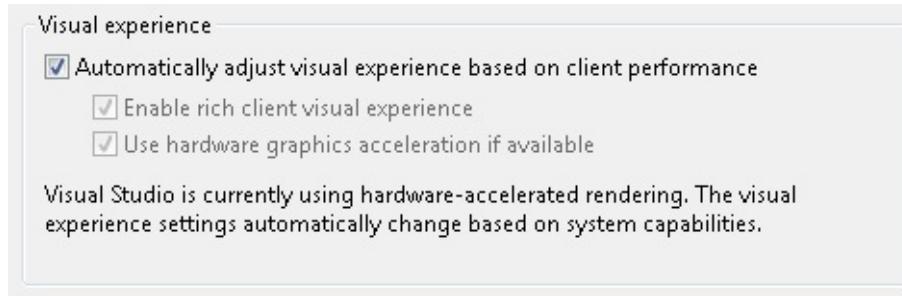
Closes the dialog box without recovering any listed files.

# 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Environment   General   Visual Experience
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEnv0017

Visual Studio 2010 automatically adjusts the visual experience depending on the situation. For example, it might eliminate or reduce the use of gradients and animations when running in Remote Desktop or virtual machine environments. It also makes use of hardware graphics acceleration when that's available.

In some situations, you can improve Visual Studio's performance by changing its Visual Experience manually. To change these settings, select Tools | Options | Environment | General | Visual Experience to see the following dialog box.



Clear the Automatically Adjust Visual Experience Based On Client Performance check box.

## NOTE

As you work with the preceding options, the message at the bottom of the dialog box does not change until you click OK to commit the changes you have made.

Following is a brief explanation of what each option does:

- **Enable Rich Client Visual Experience** This option gives you gradients and animations (also known as “eye candy”) for elements such as sliding tool windows and so on. If you leave this option selected, Visual Studio uses these

rich animations in all scenarios—including remote sessions. You should usually turn this option off in such situations to get a bump in performance.

- **Use Hardware Graphics Acceleration If Available** This option lets you decide whether Windows Presentation Foundation (WPF) hardware acceleration is something you want. If this can benefit you, you'll notice a clear change in performance when you enable or disable this option. Make sure to test both scenarios.

#### NOTE

If you have a system whose performance doesn't suffer when animations and gradients are turned on, a little eye candy can be a good thing, so this tip is really for those folks who are having performance issues in their Visual Studio experience, either locally or remotely.

Now that you have played with the preceding options a bit, you might be wondering whether you can actually see how much they can improve (or hurt) performance. The Windows SDK includes a tool called WPFFPerf that enables you to measure WPF performance. You can find a great article on how to use it at the Microsoft WindowsClient.NET site, at <http://windowsclient.net/wpf/perf/wpf-perf-tool.aspx>.

Also, when using Visual Studio 2010 over remote sessions, you should definitely read the article titled “Optimizing Visual Studio 2010 and WPF Applications for Remote Desktop,” at

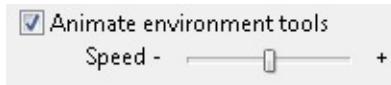
<http://blogs.msdn.com/b/jgoldb/archive/2010/02/27/optimizing-visual-studio-2010-and-wpf-applications-for-remote-desktop.aspx>. This article provides important information about how to dramatically improve performance over Remote Desktop.

## 01.07 Change Tool Window Animations

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008
<b>CODE</b>	vstipEnv0018

In the tip vstipEnv0017 ([01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#), page 14), you saw a discussion of how you can improve the visual performance of Visual Studio 2010. Now let's look at Visual Studio 2008 and 2005. You can change the animation speed of tool windows in Visual Studio 2008 and 2005, but why would you want to do this?

The answer is that you can get a performance boost by speeding up or completely turning off the animation. Select Tools | Options | Environment | General, and locate the Animate Environment Tools option.

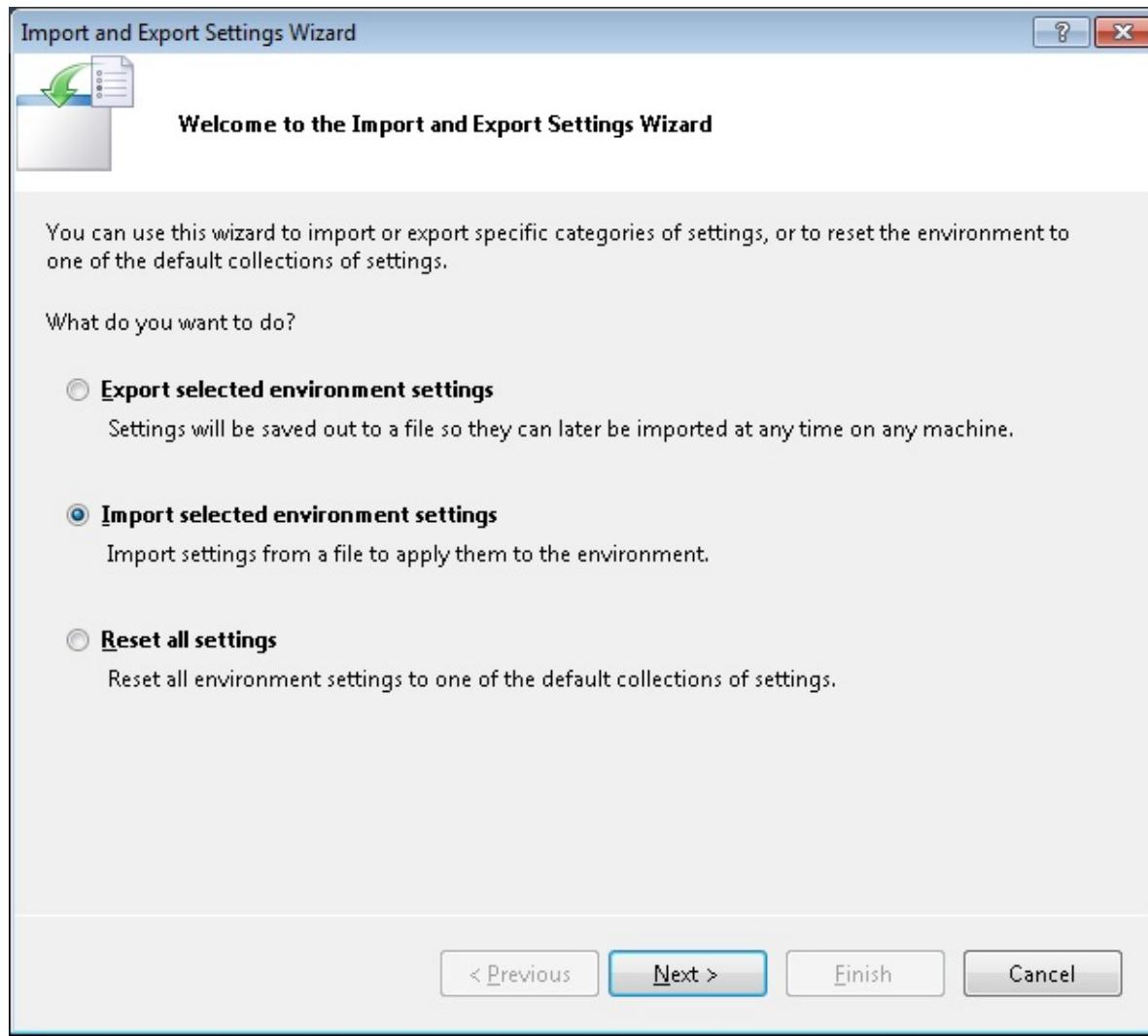


I suggest you turn off this feature to begin with, to see whether you notice any performance improvements. Later, if you want your animations back, turn on the option and set the slider to the far-right side (the fastest speed). As you test the performance, you can adjust it back to the left to determine the best setting for you.

## 01.08 Importing or Changing Your Environment Settings

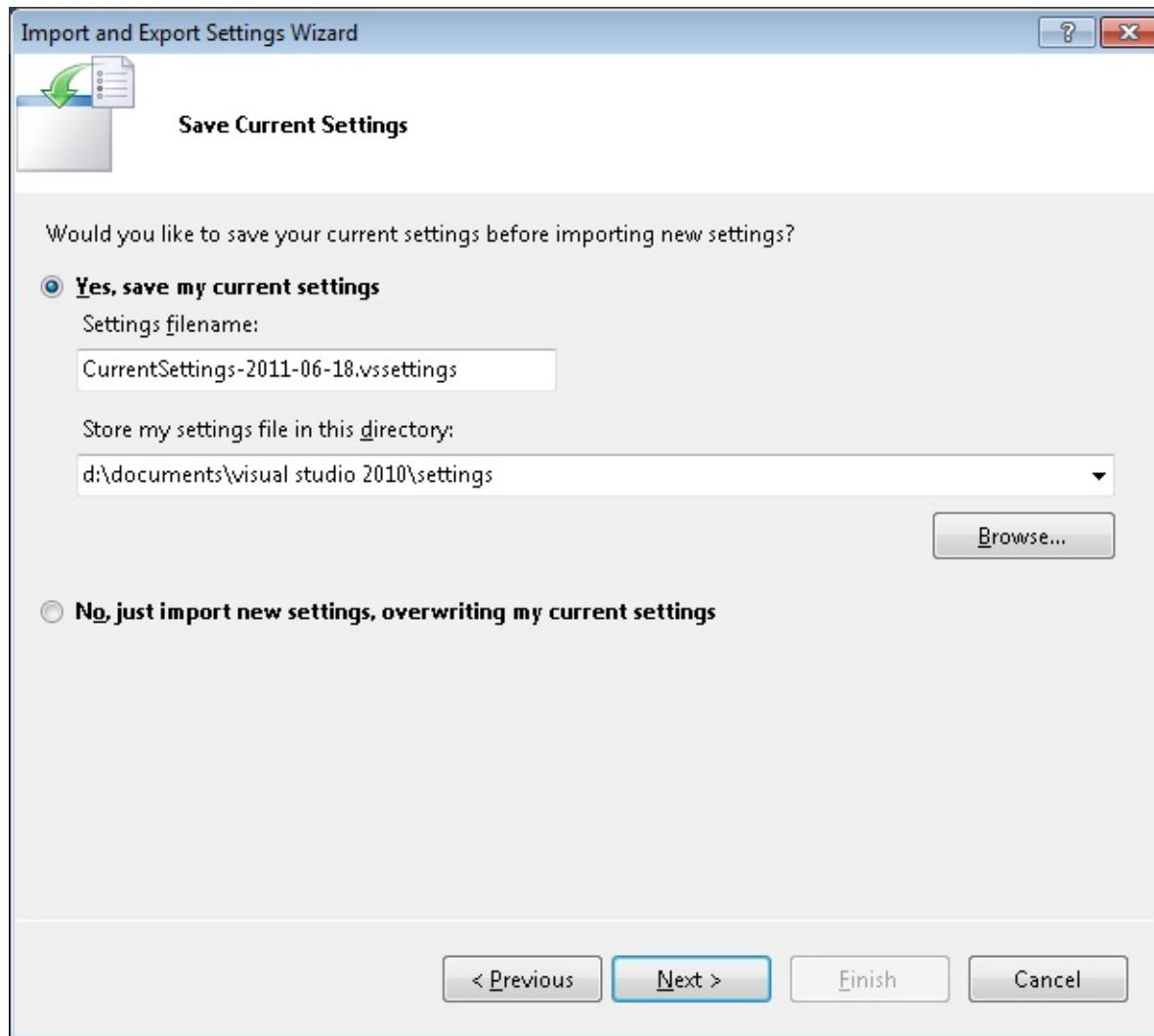
<b>WINDOWS</b>	Alt,T, I
<b>MENU</b>	Tools   Import and Export Settings
<b>COMMAND</b>	Tools.ImportandExportSettings
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0022

Assuming you have exported your settings (vstipEnv0021, [01.03 Exporting Your Environment Settings](#), page 6), you can import your settings by going to Tools | Import And Export Settings Wizard and selecting Import Selected Environment Settings:

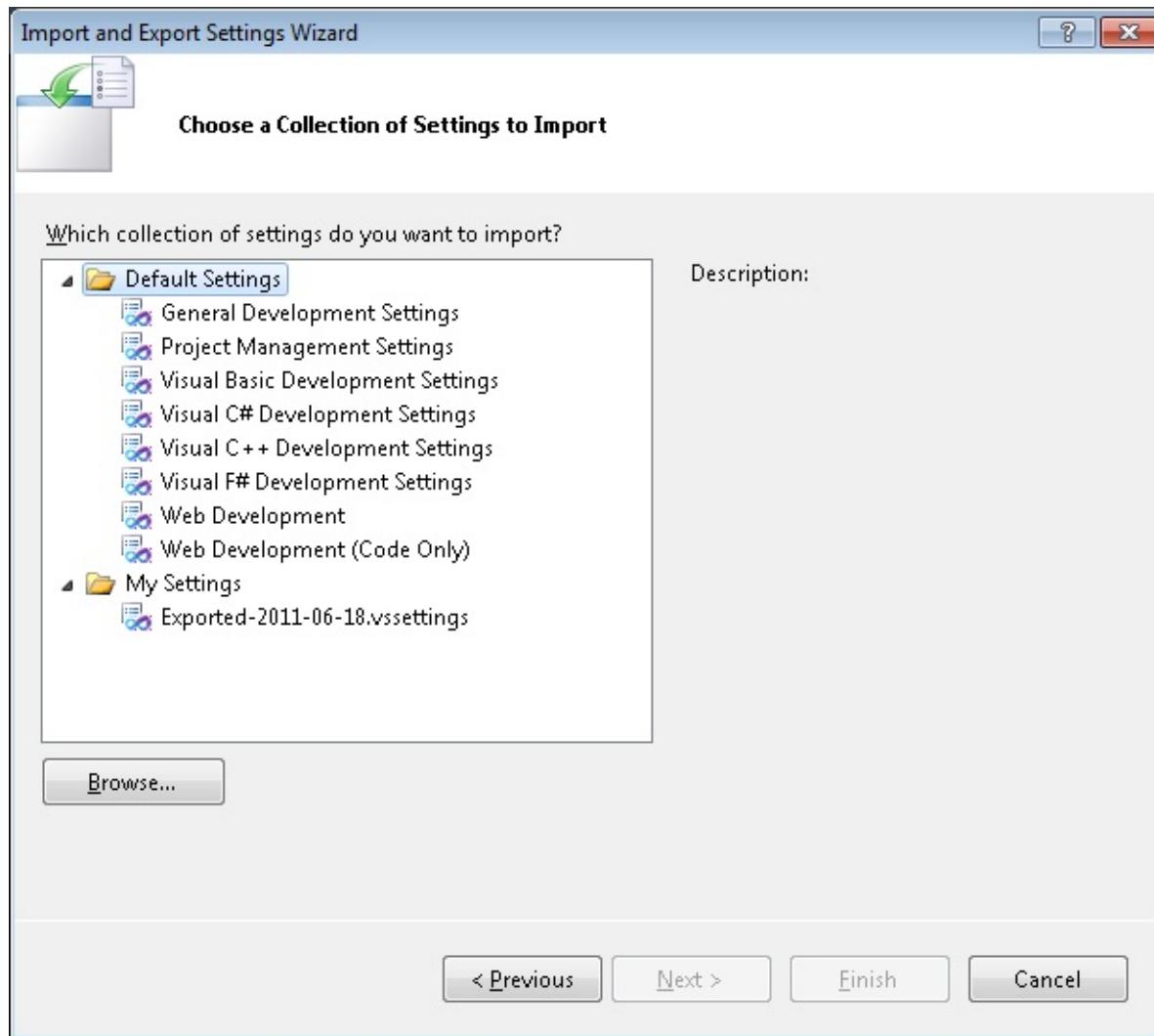


This setting enables you to import a previously exported .vssettings file.

After you click Next on the Welcome page shown in the preceding illustration, you have the option to save your existing settings (recommended) before overwriting or to just overwrite them:



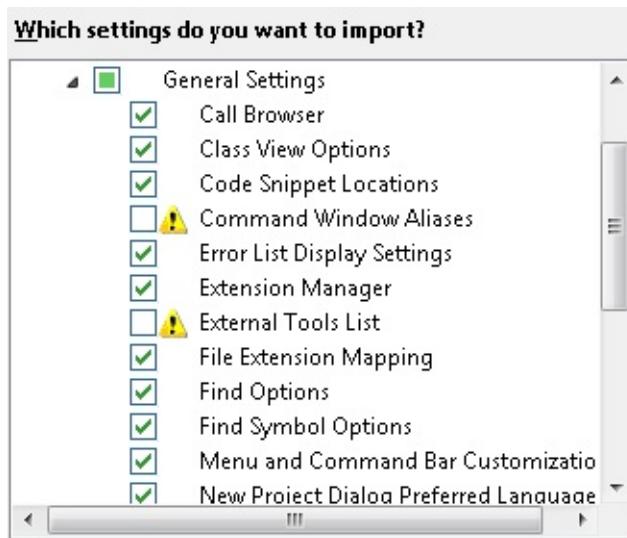
Click Next, and you can choose from the default settings, settings that have been saved previously, or you can browse for your own .vssettings file:



Now click Next again to choose what settings you want to import. All the previously exported settings are selected by default except for Command Window Aliases, External Tools List, and Import and Export Settings, which are considered potentially dangerous.

#### WARNING

You have to determine the potentially dangerous areas for yourself, but if doing a full export or import, you would most likely check all the items in this dialog. It is not recommended that you share full exports with team members as there may be information in the file you don't want to share. Instead, just export the items you want to share with team members in a separate file.



After you have checked (or unchecked) the items you want, click Finish to import the settings and to see the final page of the wizard:



Now, just click Finish and you are done.

# 01.09 Change Your Visual Studio Color Scheme

WINDOWS	Alt,T, I
MENU	Tools   Import and Export Settings
COMMAND	Tools.ImportandExportSettings
VERSIONS	2005, 2008, 2010
CODE	vstipEnv0034

Ever see a set of colors your friend or coworker has and wish you could get it too? Ever go to <http://studiosyles.info> and want some of those cool color schemes?

The screenshot shows the homepage of the studiosyles website. At the top, there's a navigation bar with 'Home' (highlighted in blue), 'Create a scheme', 'Schemes', and a search bar. A 'Feedback' button is located on the left side of the main content area. The main title 'Create and share Visual Studio color schemes' is centered above a 'Get started' button. Below this, there are two links: 'Browse 648 schemes' and 'Create a new scheme'. The 'Top-rated schemes' section displays three color schemes with their details:

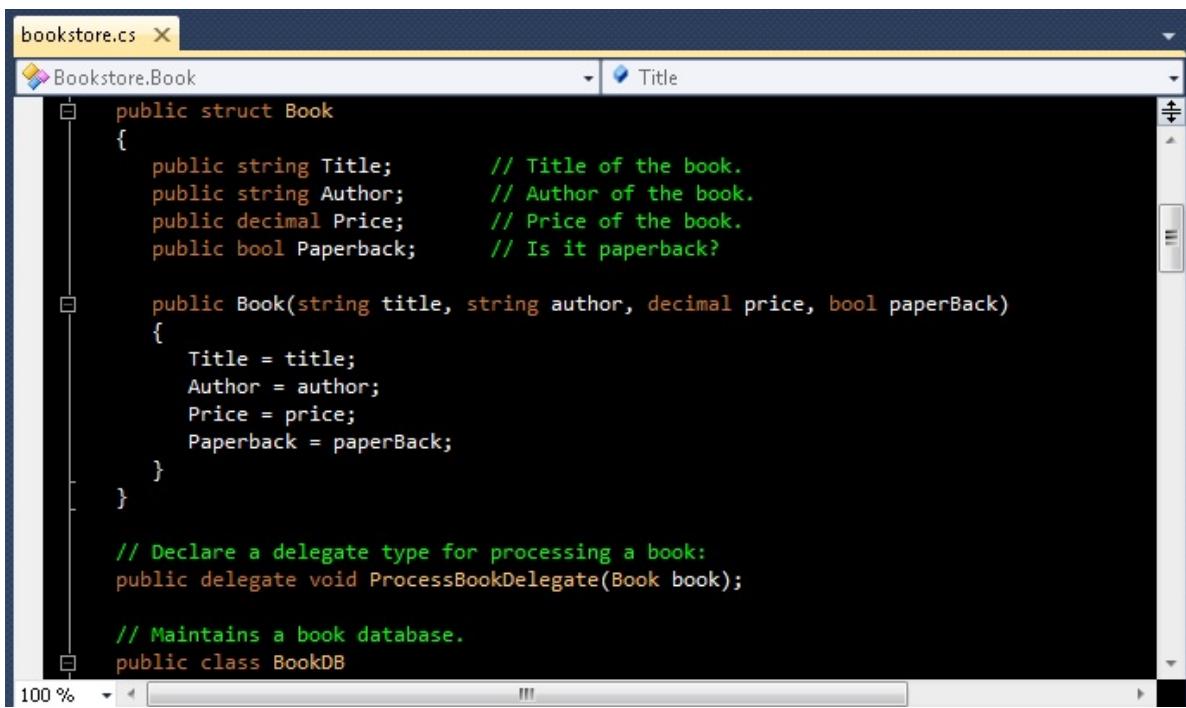
rated	58545 dls	95933 views	public class Snippet : IThemeable { static void Main() { if("Hello".Length < (43 ^ 2)) new Uri("http://there.com"); } } //Son of Obsidian	Son of Obsidian	advanced razor 1 year ago, updated 7 months ago einaros
165	62639 dls	112573 views	public class Snippet : IThemeable { static void Main() { if("Hello".Length < (43 ^ 2)) new Uri("http://there.com"); } } //WekeRoad Ink	WekeRoad Ink	advanced razor 1 year ago, updated 7 months ago Luke Sampson
134	13450 dls	30879 views	public class Snippet : IThemeable { static void Main() { if("Hello".Length < (43 ^ 2)) new Uri("http://there.com"); } } //Selenitic	Selenitic	advanced razor 1 year ago, updated 4 months ago Tim G. Thomas

Well you can get the colors you want! Let's walk through how it's done.

## Seeing What You Like

First, you see a seriously cool color scheme on someone's screen or at the Studio

Styles site:



```
bookstore.cs X
Bookstore.Book
public struct Book
{
    public string Title;          // Title of the book.
    public string Author;         // Author of the book.
    public decimal Price;         // Price of the book.
    public bool Paperback;        // Is it paperback?

    public Book(string title, string author, decimal price, bool paperBack)
    {
        Title = title;
        Author = author;
        Price = price;
        Paperback = paperBack;
    }
}

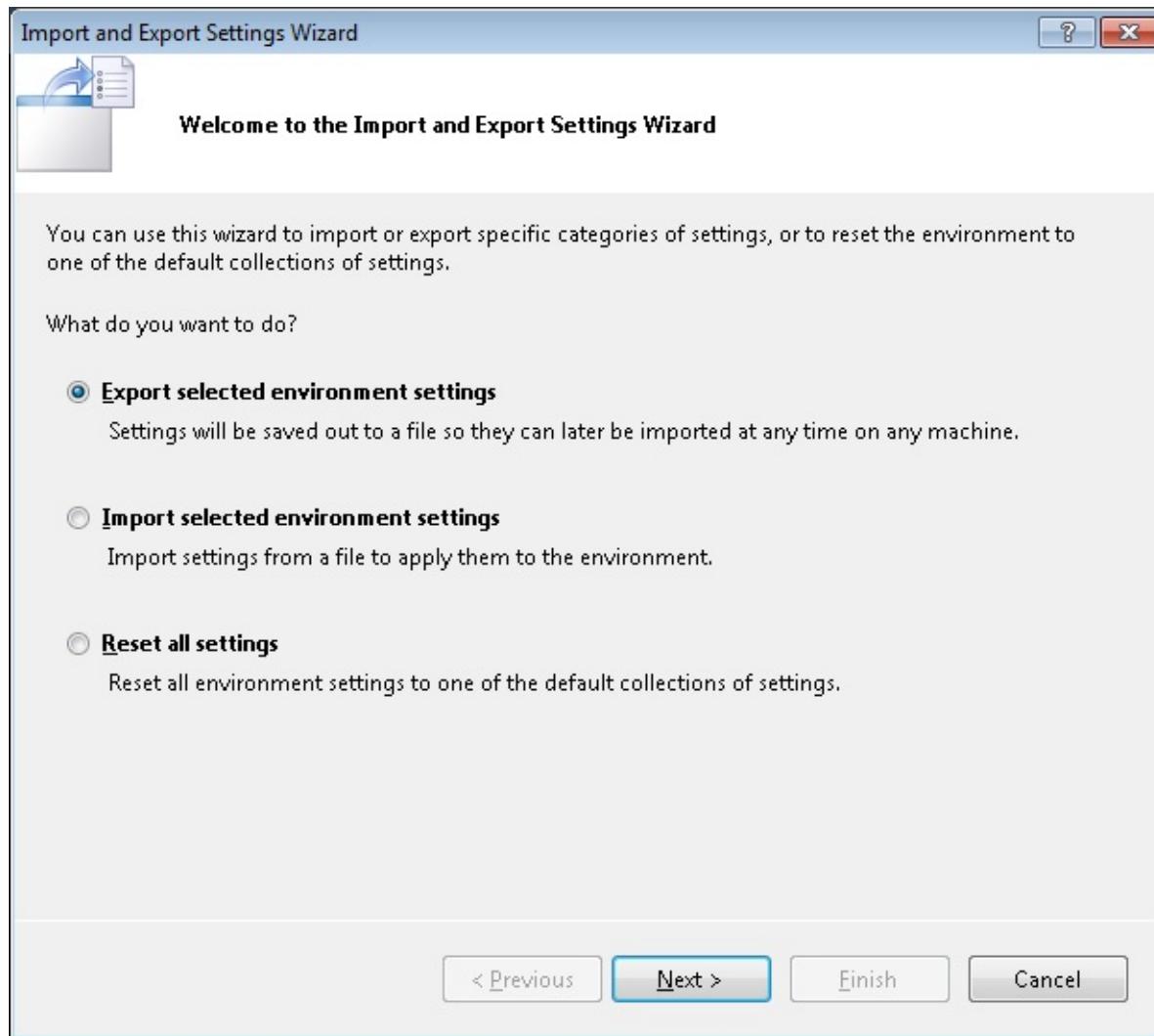
// Declare a delegate type for processing a book:
public delegate void ProcessBookDelegate(Book book);

// Maintains a book database.
public class BookDB
```

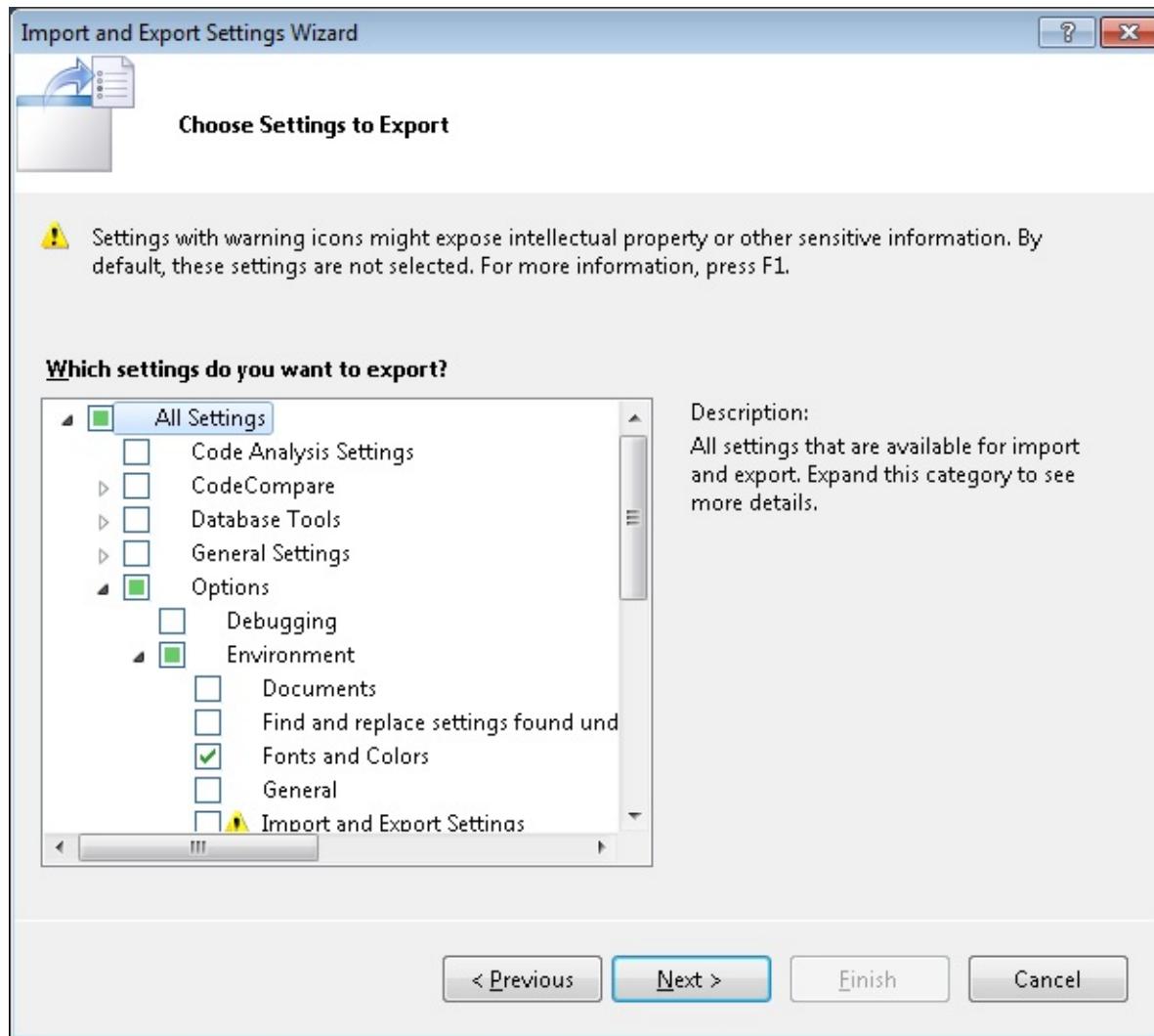
## Getting the Goods

### On someone's computer

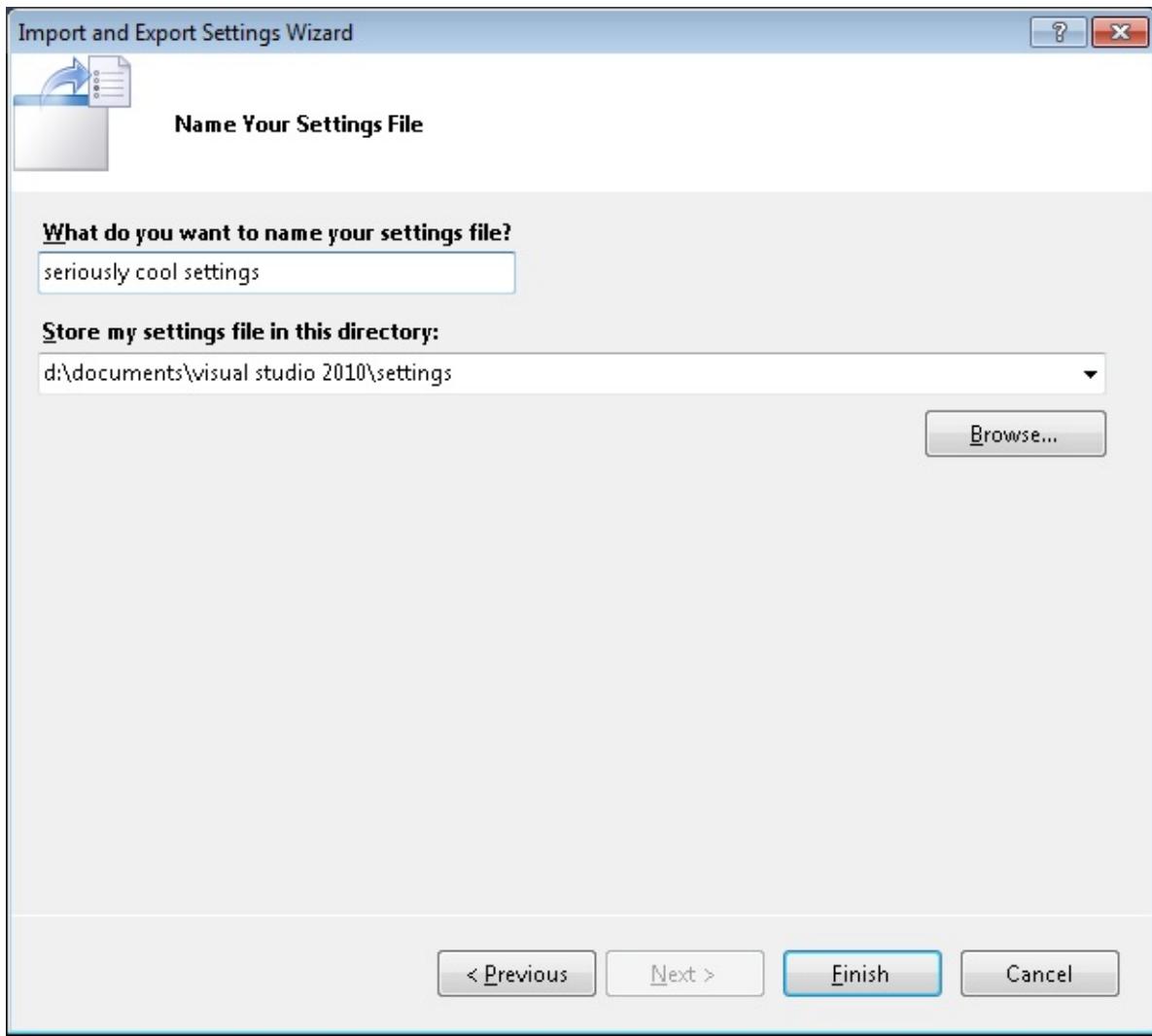
Now that you see what you like, get them to export their fonts and colors. Go to Tools | Import And Export Settings:



Click Next, and export *only* the fonts and colors—*nothing else*:



Click Next, give the settings a cool name, and click Finish:



## On the Studio Styles site

Click the style you want:

### Top-rated schemes

rated 165	58548 dls	95939 views	<pre>public class Snippet : IThemeable {     static void Main() {         if("Hello".Length &lt; (43 ^ 2))             new Uri("http://there.com");     } //Son of obsidian"</pre>	<a href="#">Son of Obsidian</a> advanced   razor   1 year ago, updated 7 months ago   <a href="#">einaros</a>
rated 134	62645 dls	112581 views	<pre>public class Snippet : IThemeable {     static void Main() {         if("Hello".Length &lt; (43 ^ 2))             new Uri("http://there.com");     } //WekeRoad Ink"</pre>	<a href="#">WekeRoad Ink</a> advanced   razor   1 year ago, updated 7 months ago   <a href="#">Luke Sampson</a>

Choose your Visual Studio version, click Download This Scheme, and follow the instructions in the next section:

## WekeRoad Ink

rated

134

Submitted by [Luke Sampson](#)

This is a refresh of Rob Conery's take on the Vibrant Ink theme WekeRoad (<http://blog.wekeroad.com/2007/10/17/textmate-th>) to advantage the expanded syntax settings in VS2010.

**Do you like it?** [Hot](#) or [not](#)

[Download this scheme](#)

for

Visual Studio 2010

Visual Studio 2010

Visual Studio 2008

Visual Studio 2005

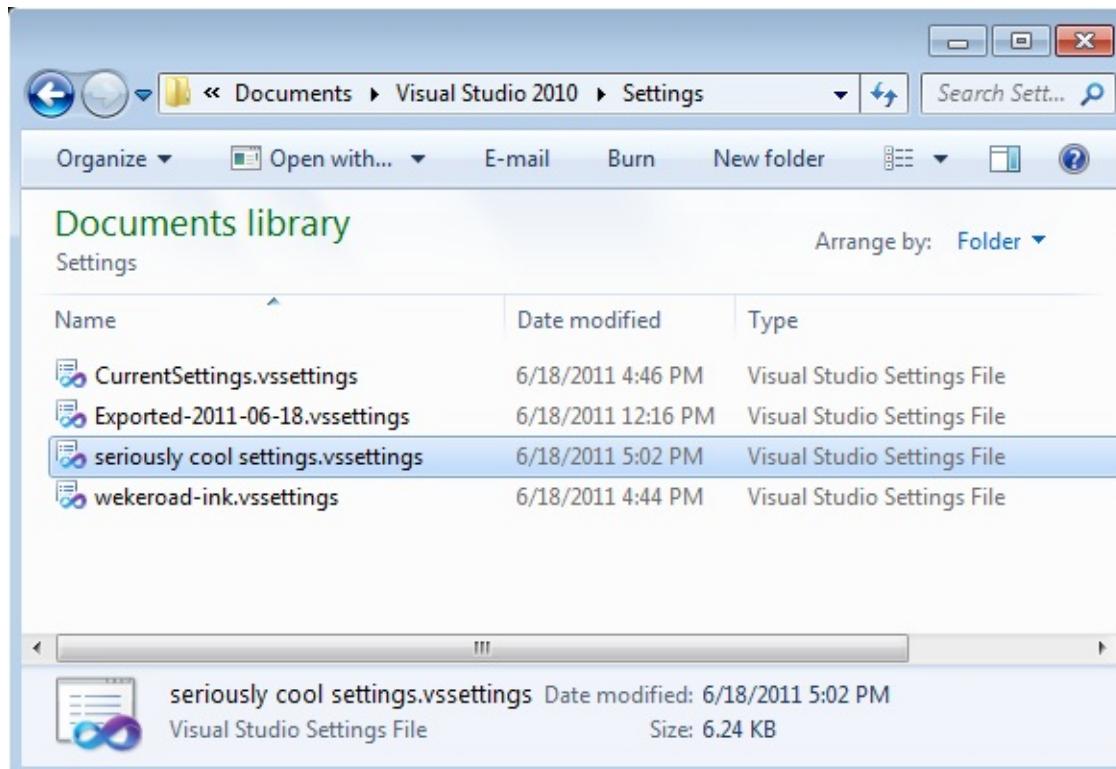
Feedback

C# code

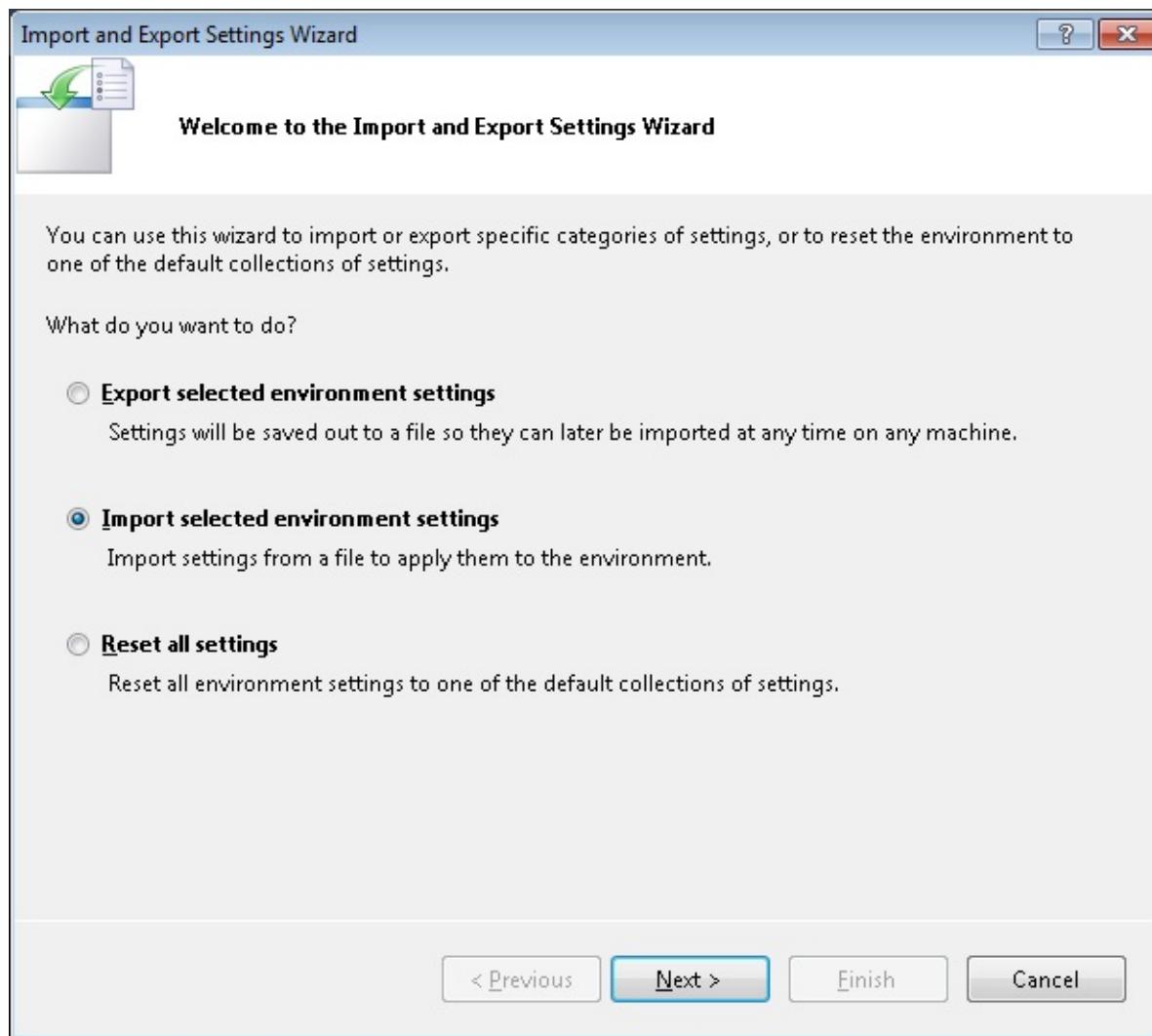
```
1 #region Studio Style
2 class Program : IThemeable
3 {
4     static int _I = 1;
5     delegate void DoSomething();
6 }
```

## Changing Your Colors

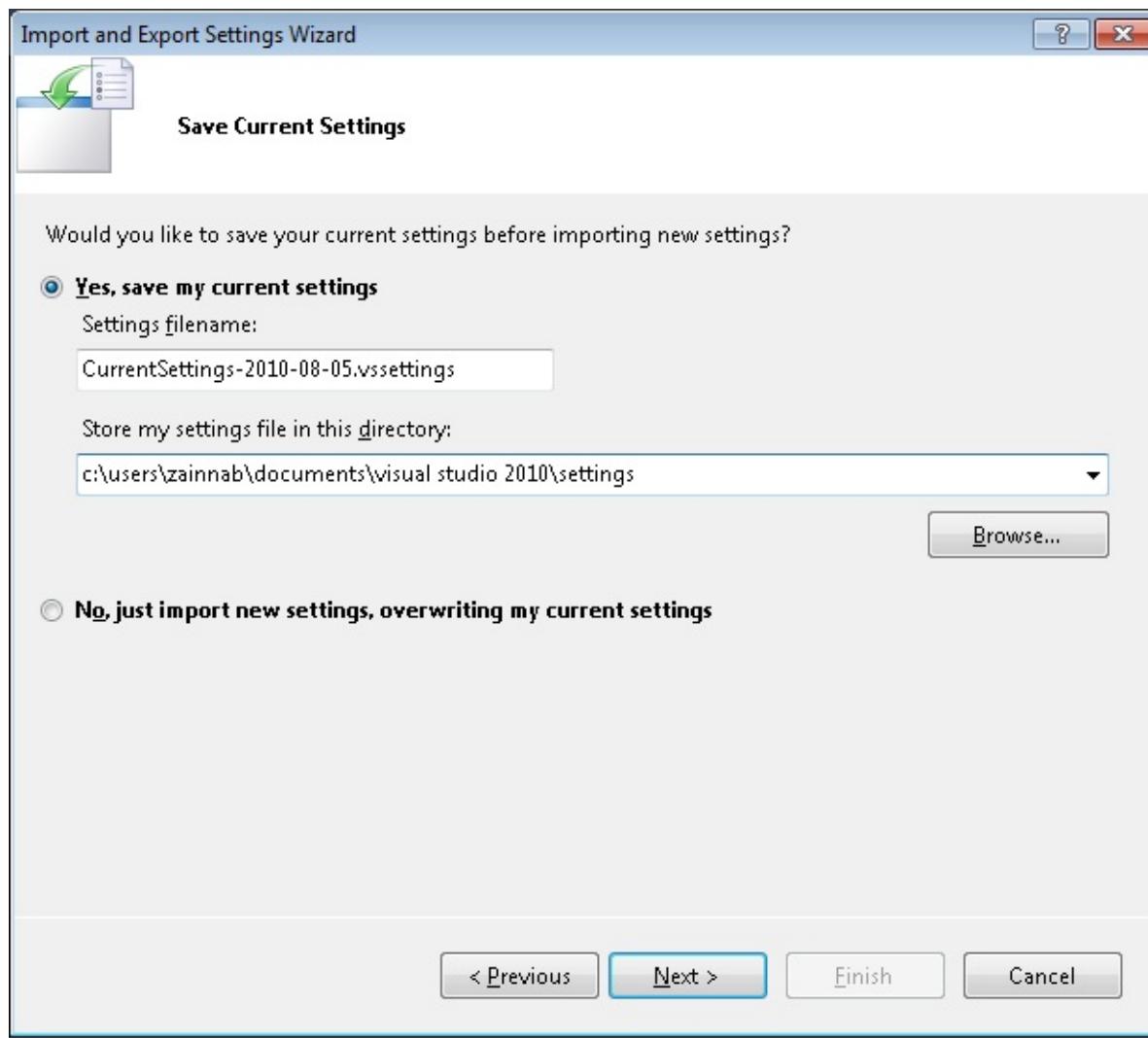
When you have a .vssettings file you want to import, copy or move the file to your computer. While you can put the file anywhere you want on your system, I prefer to put it with the other settings files located at C:\Users\<user>\Documents\Visual Studio <version>\Settings:



Now just go to Tools | Import And Export Settings on your computer:



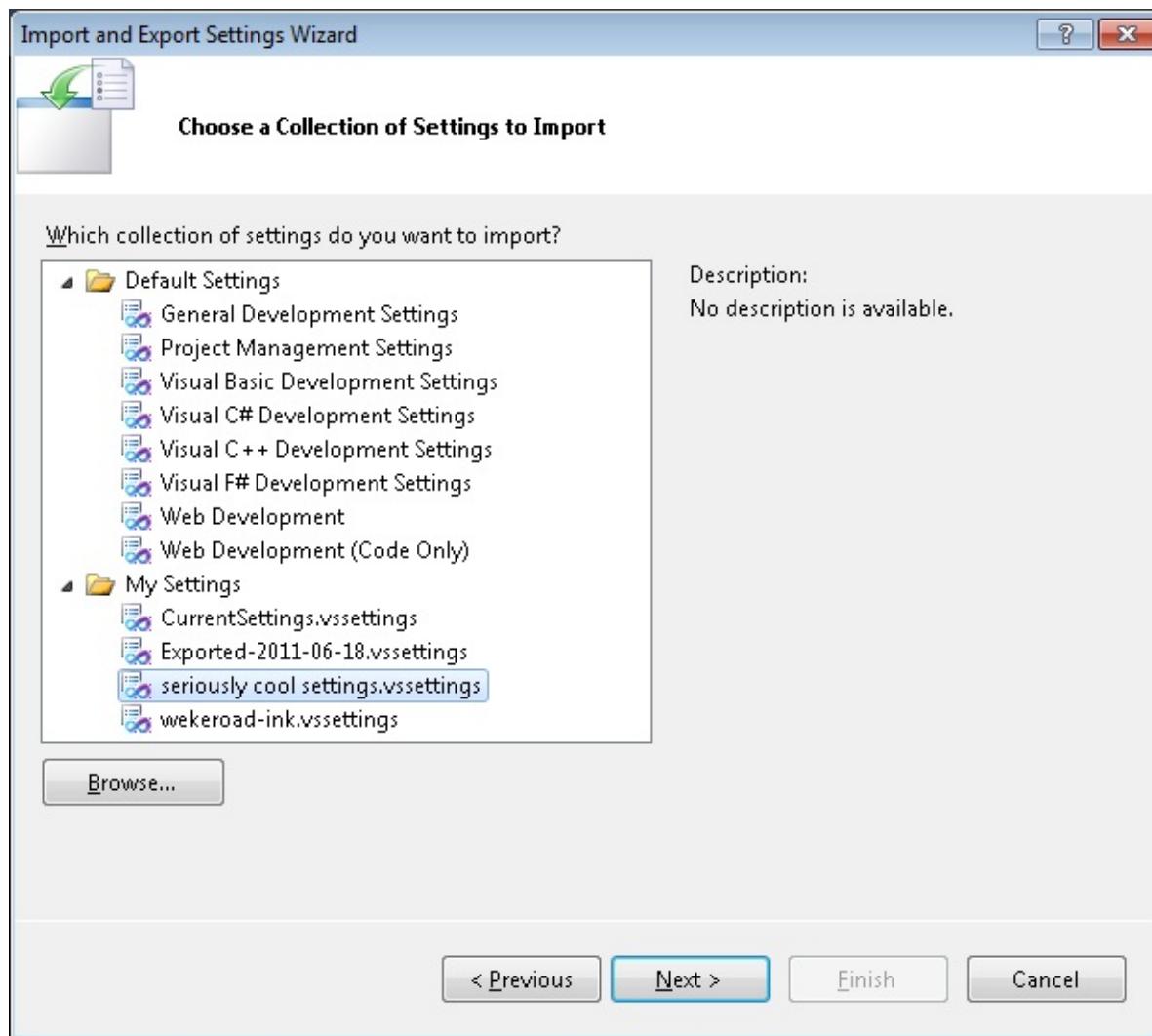
Make sure that Import Selected Environment Settings is selected, as shown in the preceding illustration, and click Next. If you haven't backed up your settings in a while, feel free to do so. Check out vtipEnv0034 ([01.09 Change Your Visual Studio Color Scheme](#), page 17) if you want more information on exporting your settings:



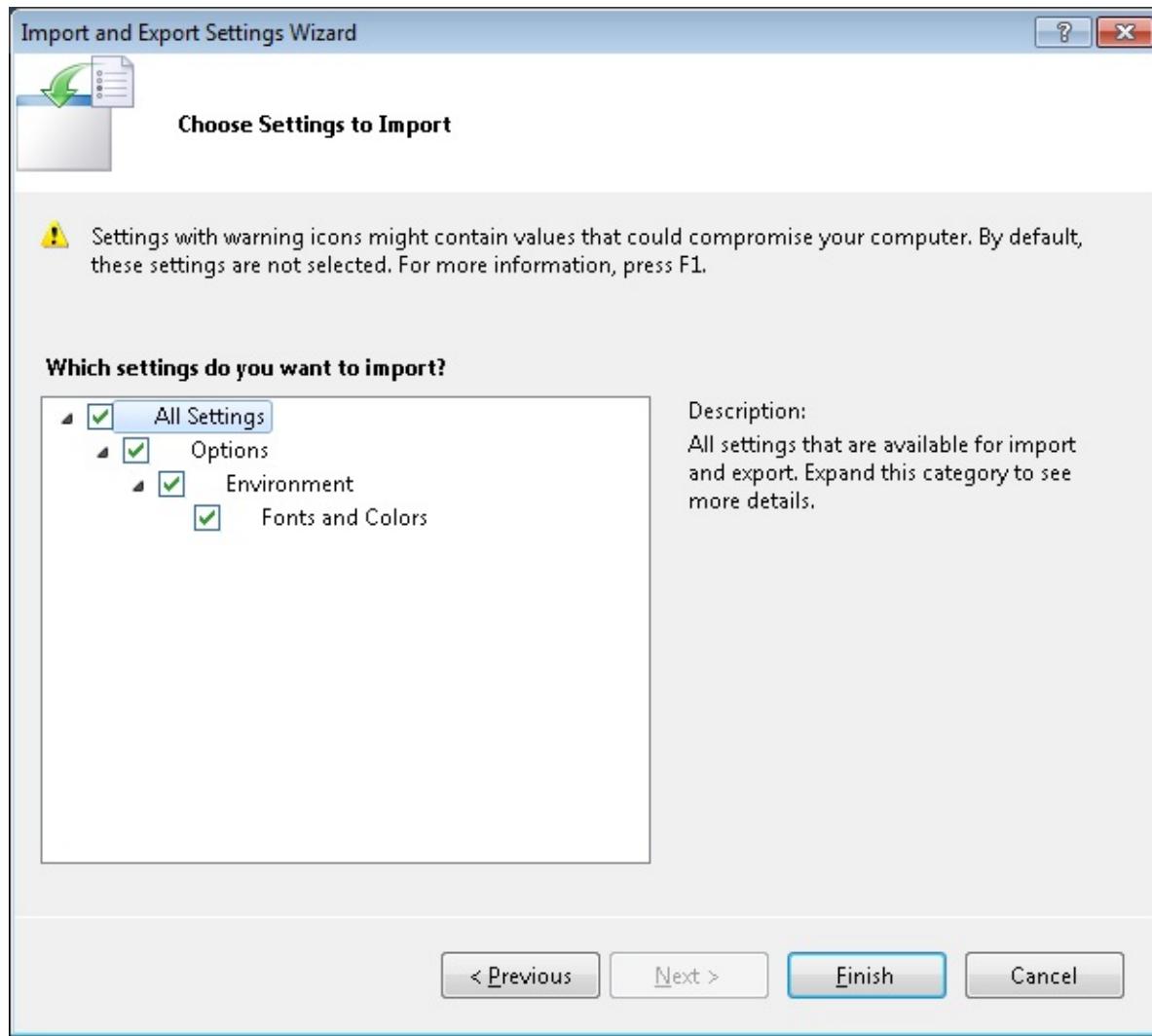
Choose the settings file that has the color scheme you want:

**NOTE**

Click Browse to find your file if you didn't put it in your Settings folder.



Click Next. Verify that the file is importing *only* fonts and colors, and then click Finish:

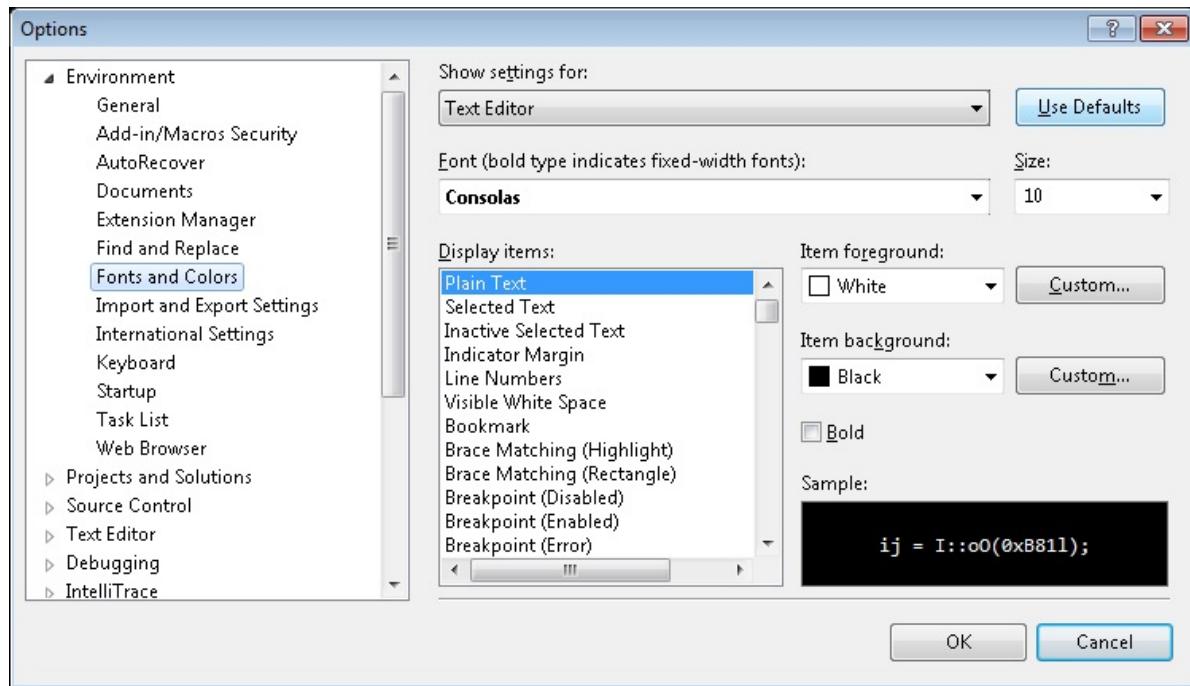


## Resetting the Colors

You should have your new colors. If things get bad (for example, you get colors you don't like and didn't make a backup of your old colors) and you need to get the default colors back, all you have to do is go to Tools | Options | Fonts And Colors and click Use Defaults.

### WARNING

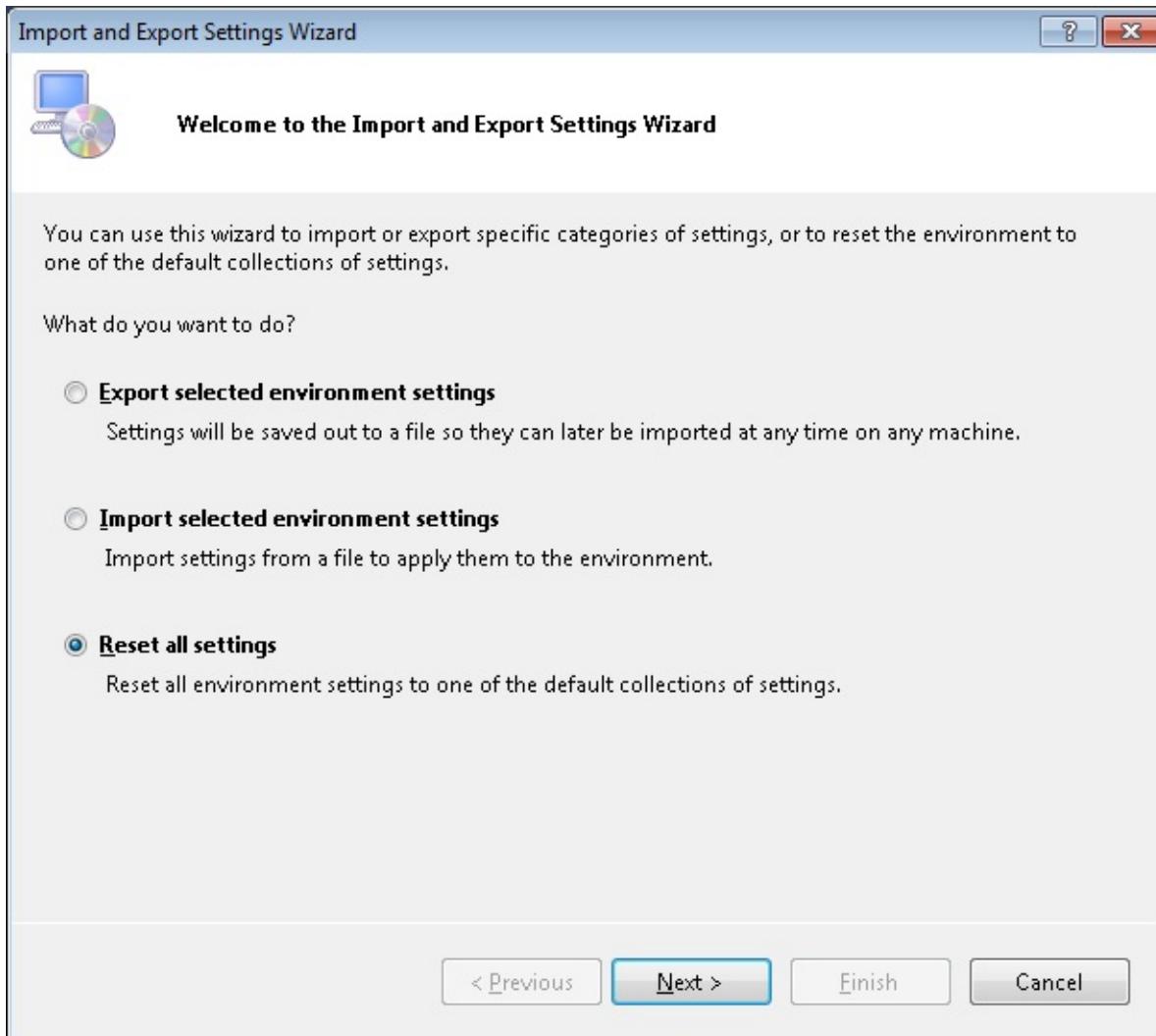
Clicking Use Defaults is an option that wipes out any custom colors used previously.



## 01.10 Reset All Your Development Settings

<b>WINDOWS</b>	Alt,T, I
<b>MENU</b>	Tools   Import and ExportSettings
<b>COMMAND</b>	Tools.ImportandExport Settings
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0023

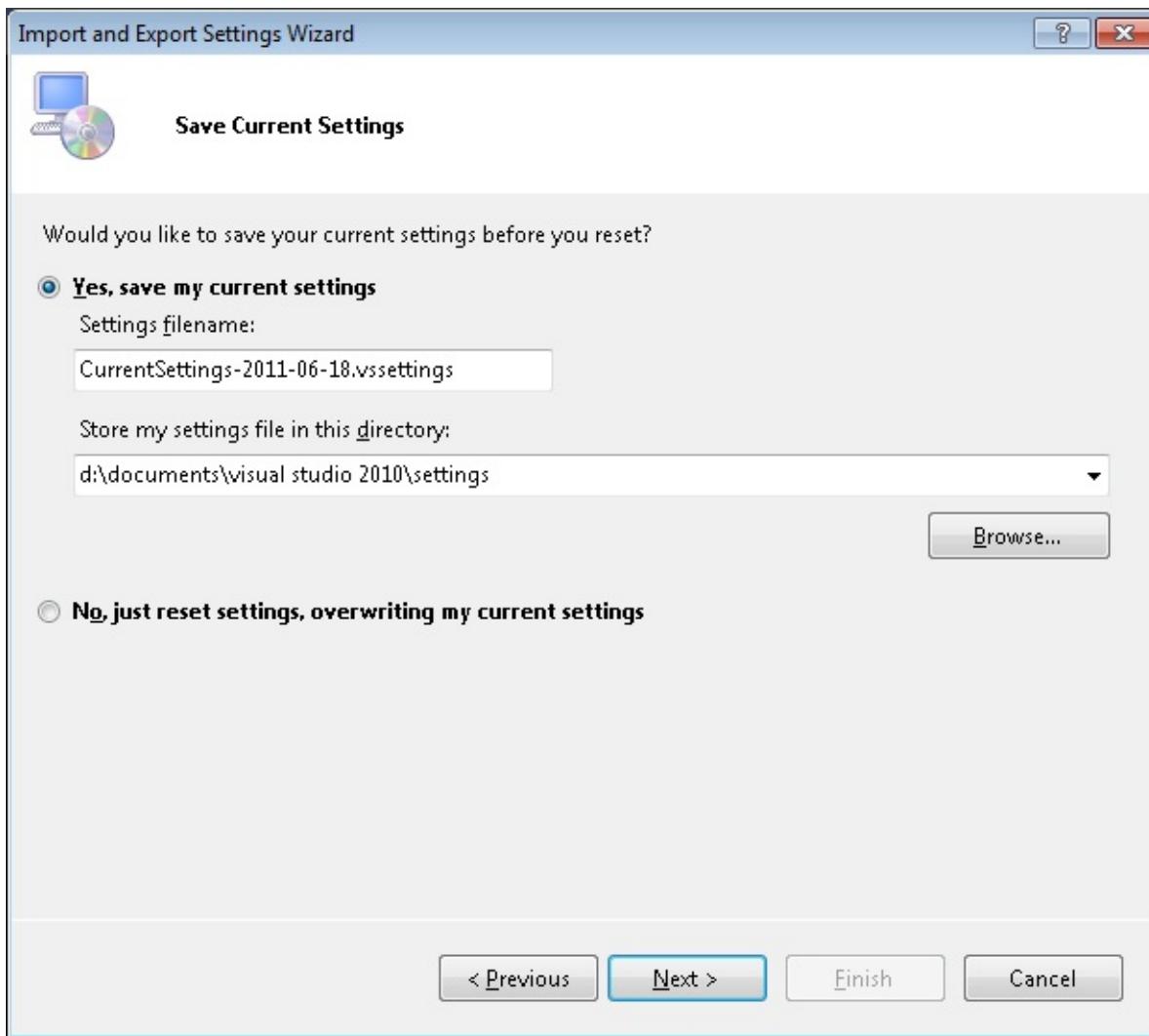
Sometimes you need to get all your settings back to their original state. You can do this with the Reset All Settings option found under Tools | Import And Export Settings:



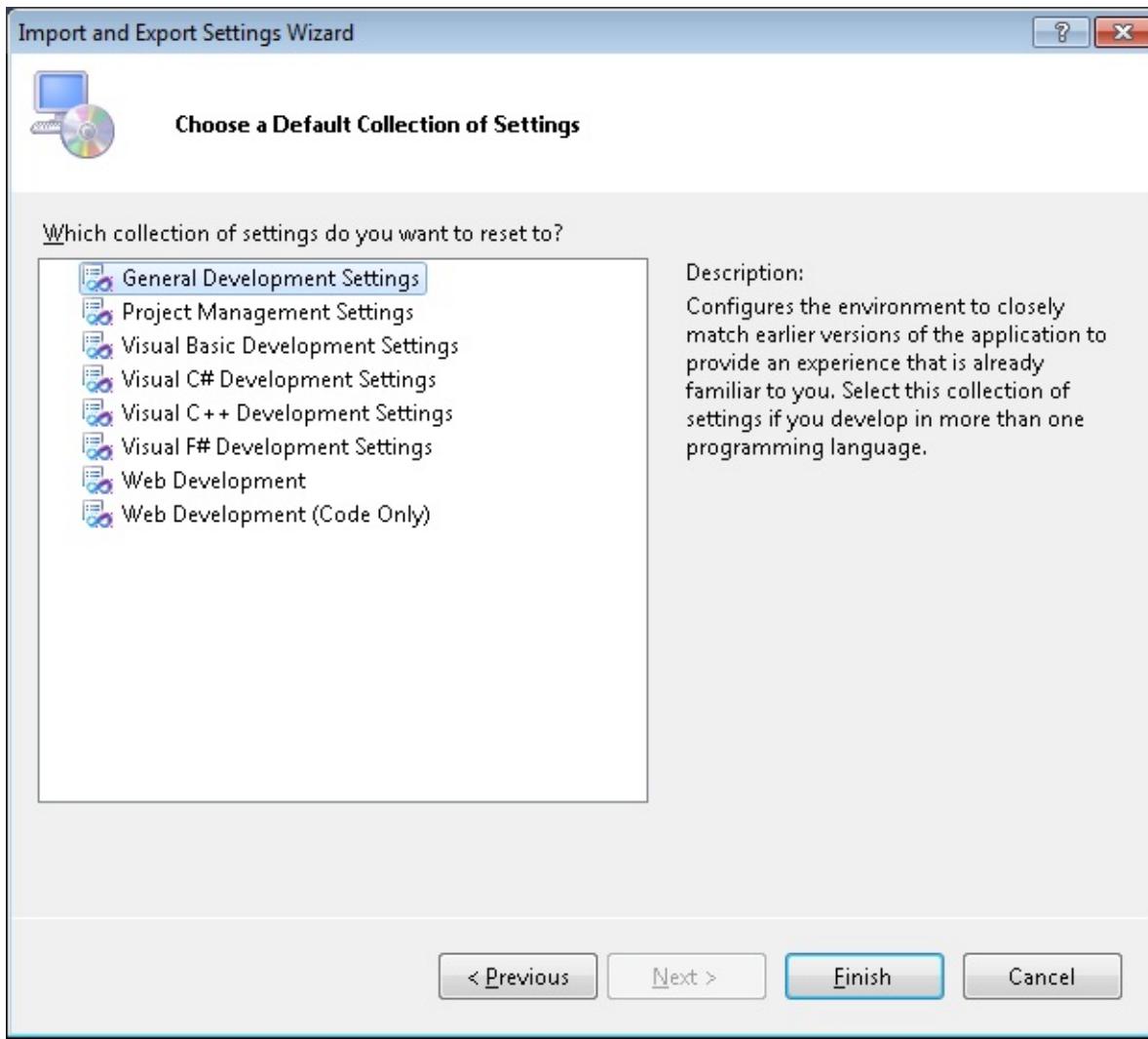
### WARNING

Use the Reset All Settings option at your own risk. It *will* reset your settings, including a reset of your Toolbox, getting rid of any custom items you have put in there.

After you click Next on the Welcome page shown in the preceding illustration, you see the option to save your current settings. *You should absolutely do this.*



The next screen lets you choose from the list of default settings:

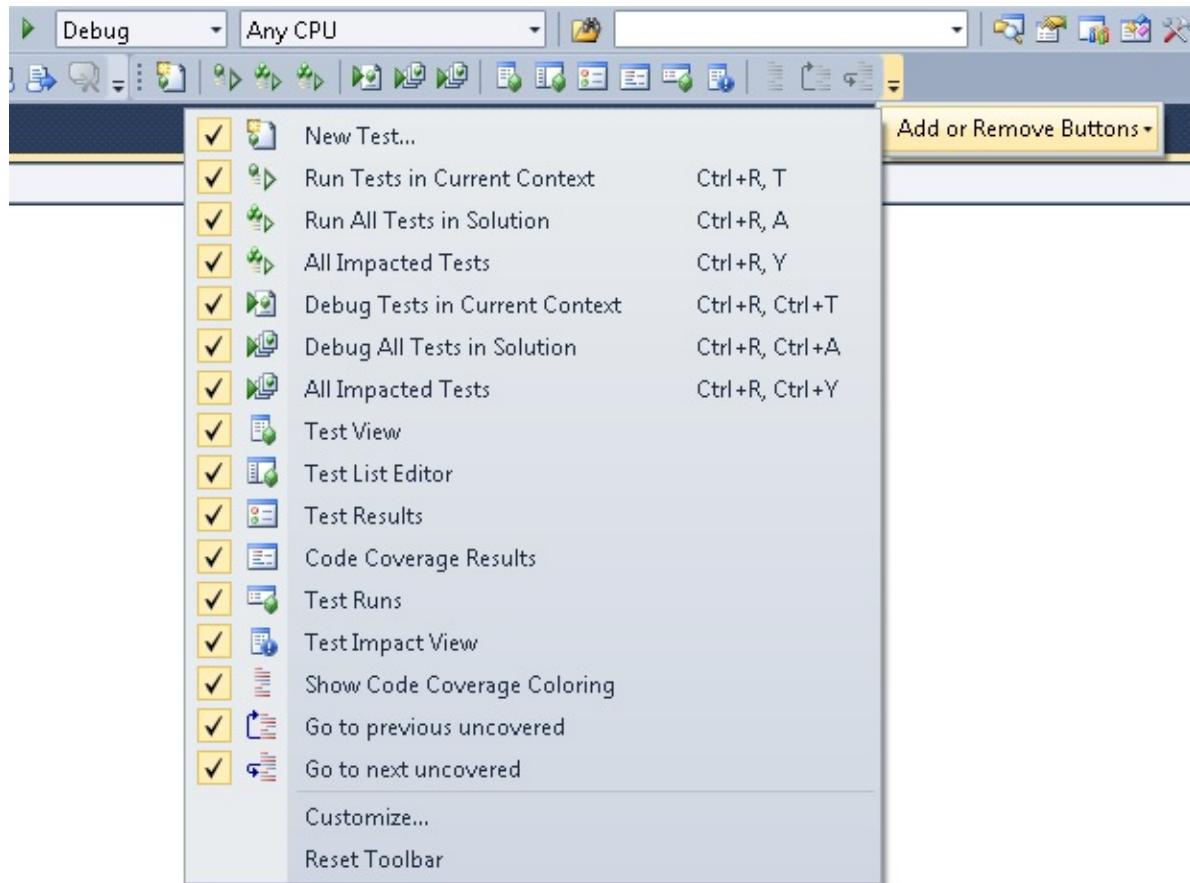


Choose your default settings, and click Finish. After the reset operation runs, it resets all your settings. This is definitely something you would do as a last resort, and remember, you can always bring back your old settings by importing settings you saved earlier (see vtipEnv0021, **01.03 Exporting Your Environment Settings**, page 6).

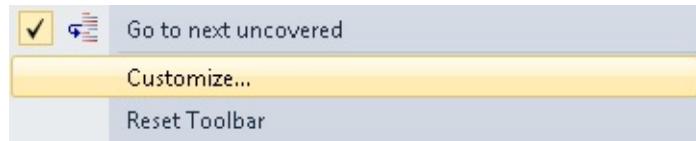
## 01.11 Customize Your Toolbars in Visual Studio 2010: Toolbars Tab

<b>WINDOWS</b>	Alt,T, C
<b>MENU</b>	Tools   Customize
<b>COMMAND</b>	Tools.Customize
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEnv0030

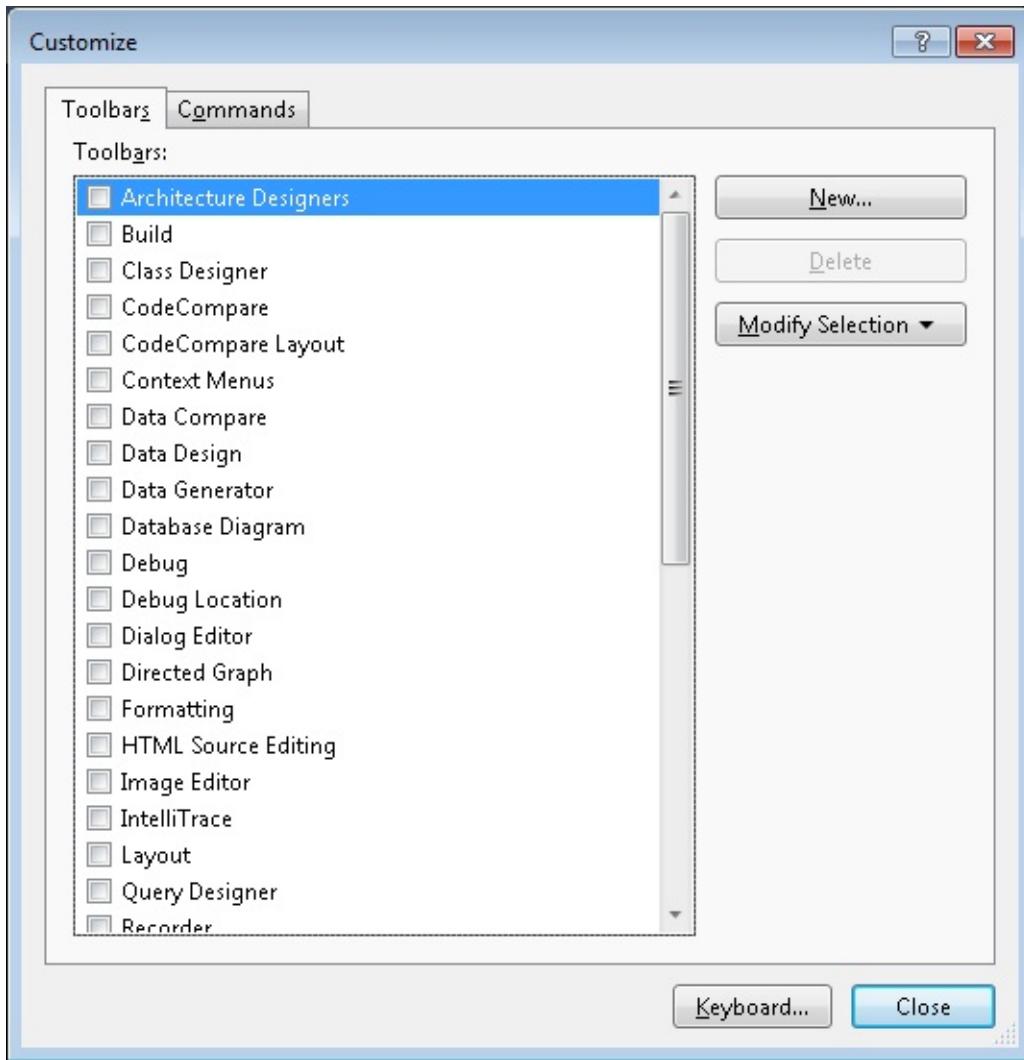
You can customize any toolbar in Visual Studio 2010. Just click the drop-down arrow to the right of any toolbar, and then click Add Or Remove Buttons:



Then click Customize:

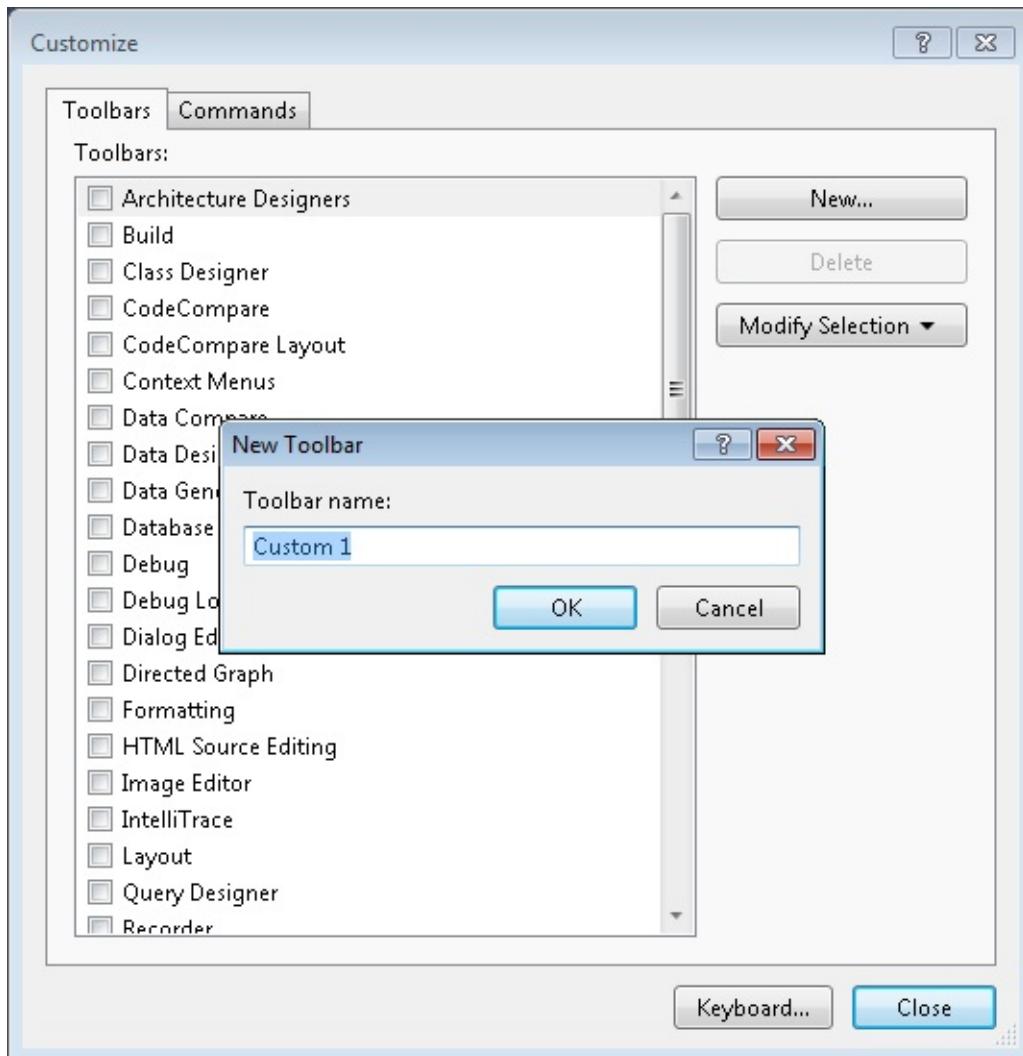


Alternatively, you can go to Tools | Customize on the menu bar. Whichever option you choose opens the Customize dialog box:

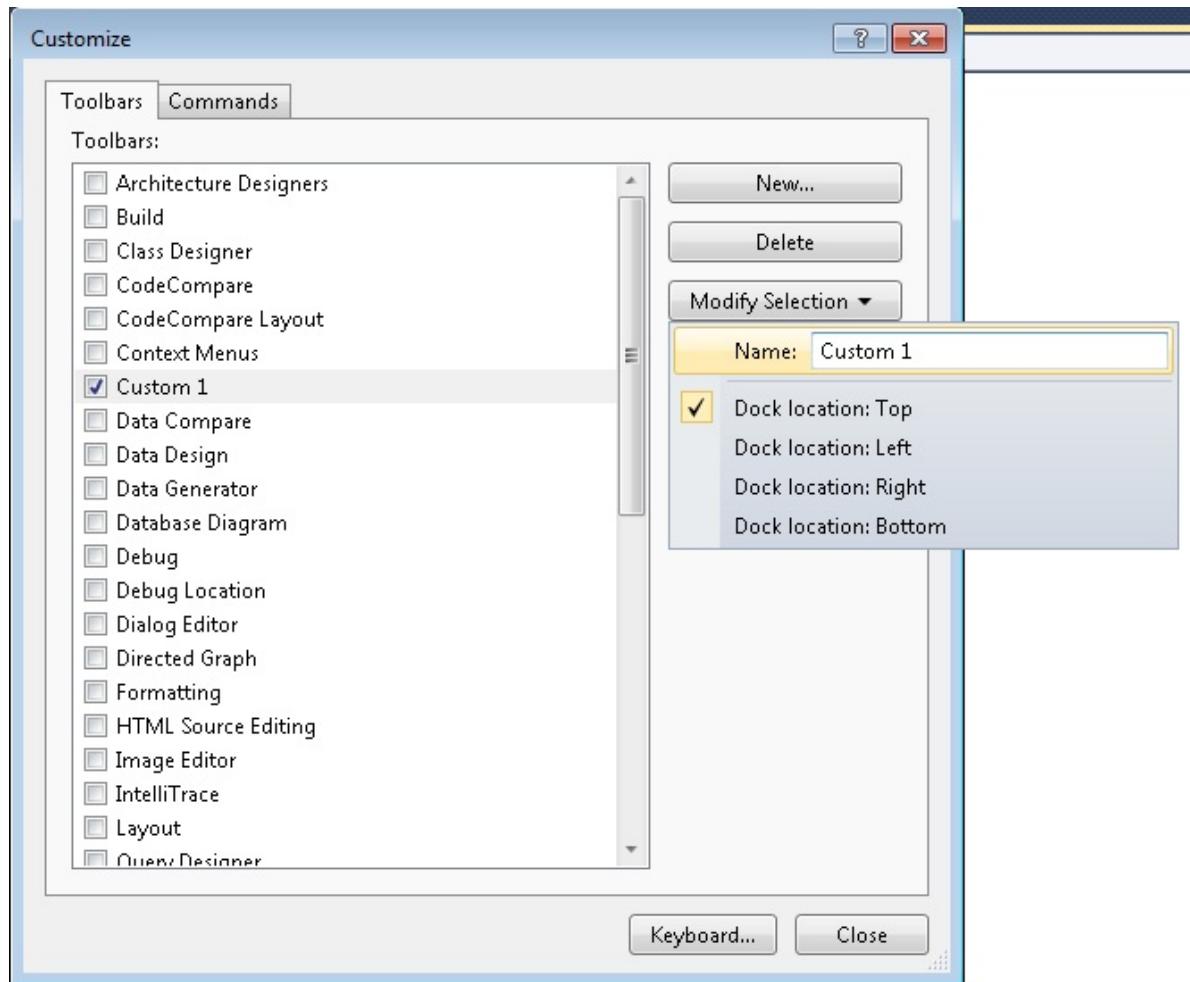


## Custom Toolbars

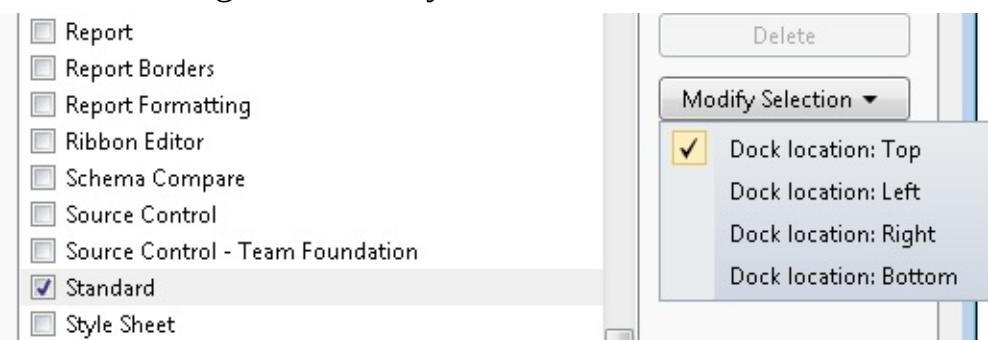
As shown in the preceding illustration, the Toolbars tab lists all the available toolbars. After you click New to create a customized toolbar, you are prompted to give the new toolbar a name:



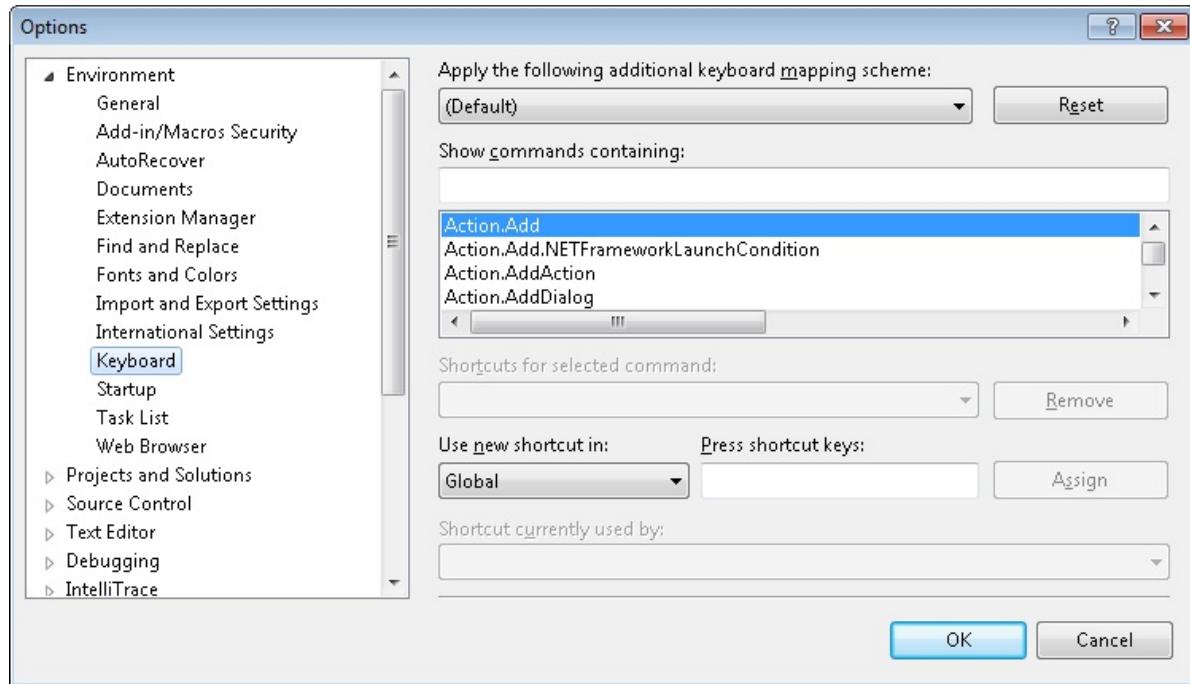
After you name it, you can delete the custom toolbar by clicking Delete, or you can change it by clicking Modify Selection to rename or relocate the toolbar:



Although you can rename custom toolbars by clicking Modify Selection, default toolbars can't be changed in this way:



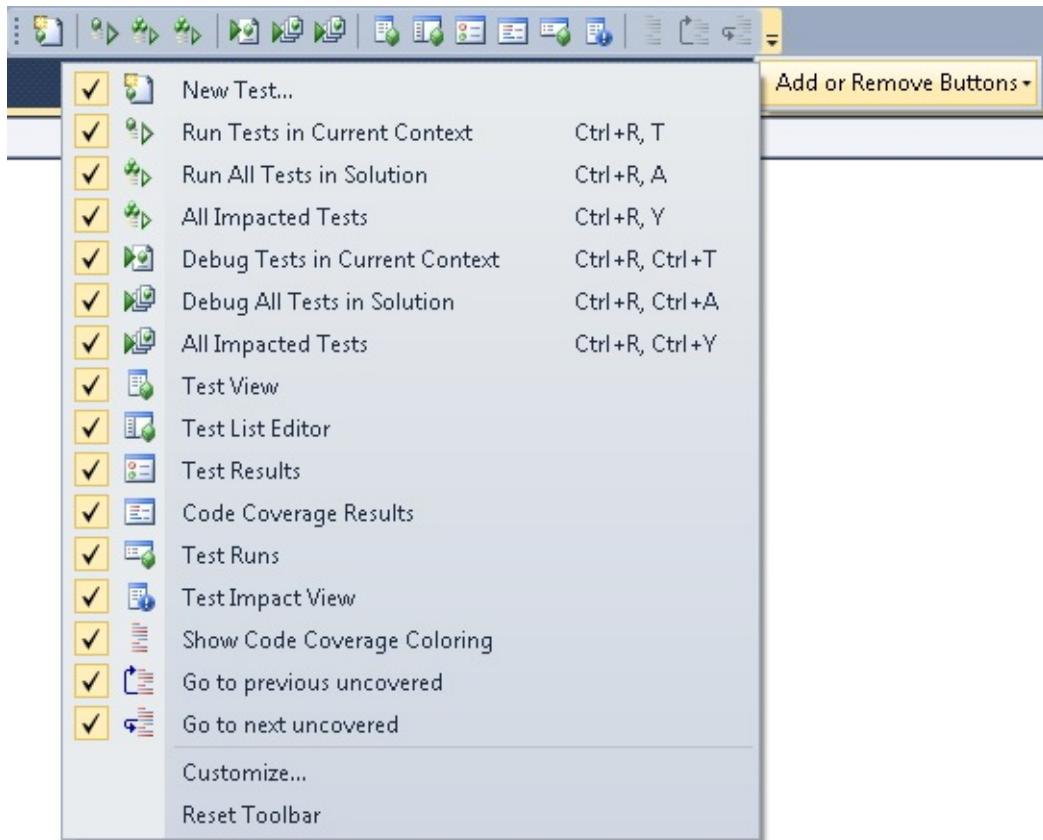
Clicking Keyboard at the bottom of the Customize dialog box takes you to the Tools | Options | Keyboard area, where you can add keyboard shortcuts for selected commands. (See vstipTool0063, [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), page 127, for details.)



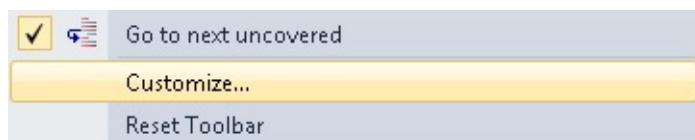
## 01.12 Customize Your Toolbars in Visual Studio 2010: Commands Tab

<b>WINDOWS</b>	Alt,T, C
<b>MENU</b>	Tools   Customize
<b>COMMAND</b>	Tools.Customize
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEnv0031

You can customize any toolbar in Visual Studio 2010. Just click the drop-down arrow to the right of any toolbar, and then click Add Or Remove Buttons:

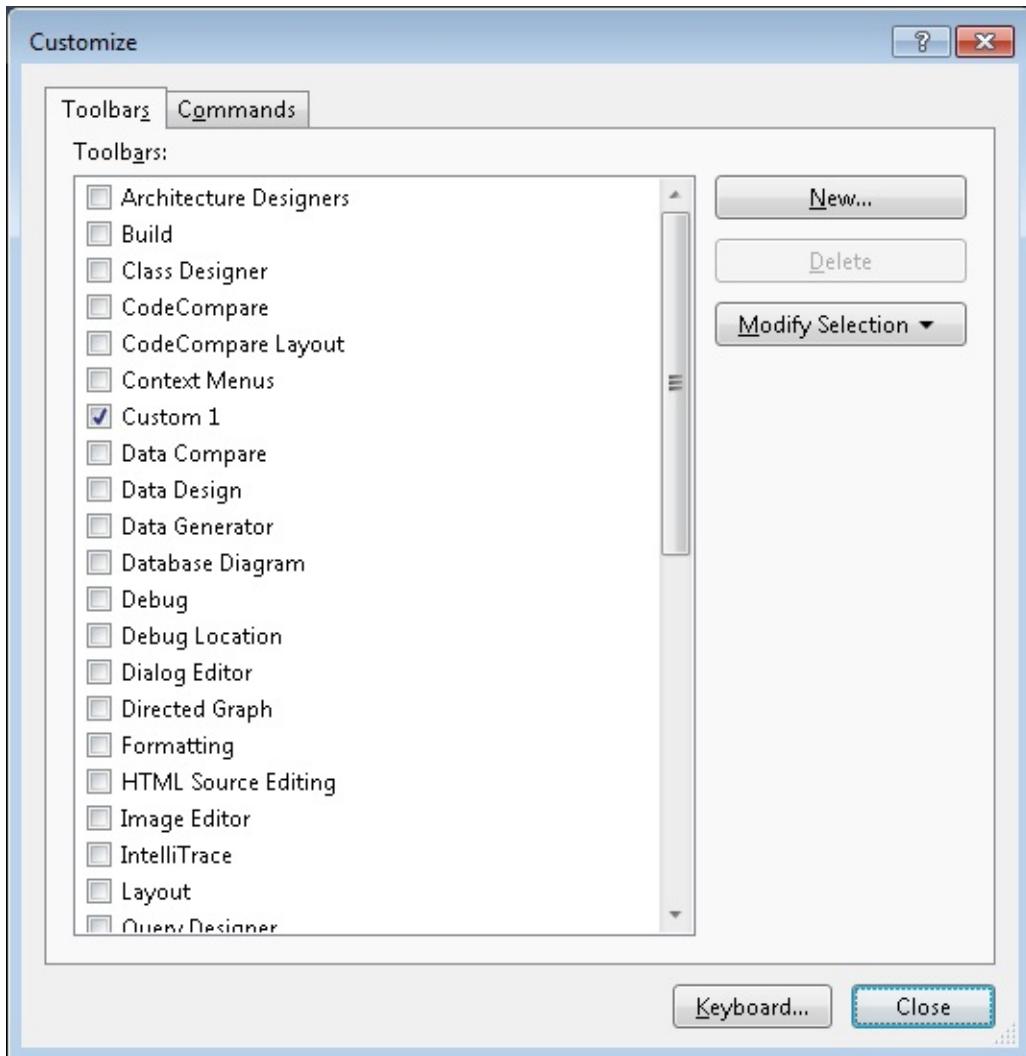


Then click Customize:



Alternatively, you can go to Tools | Customize on the menu bar.

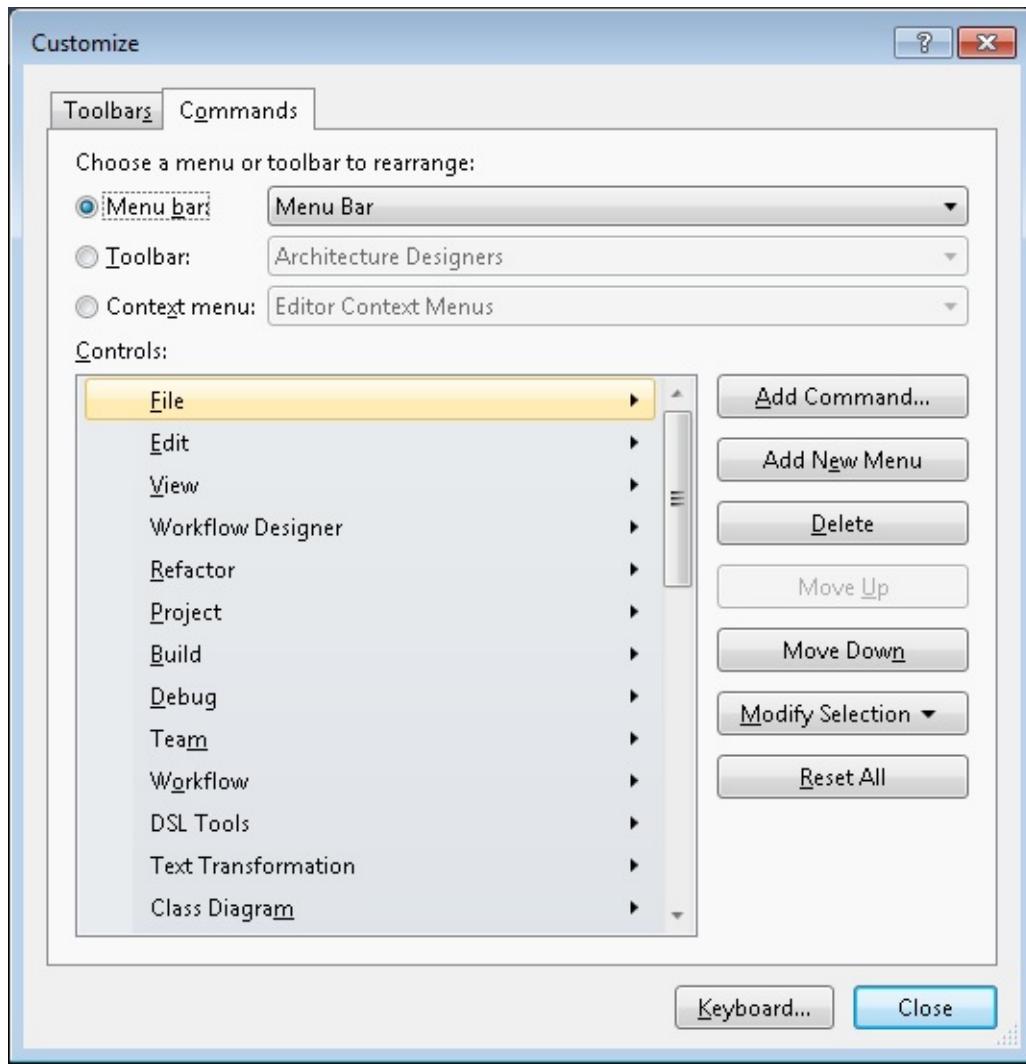
Either option you choose opens the Customize dialog box:



Click the Commands tab:

**NOTE**

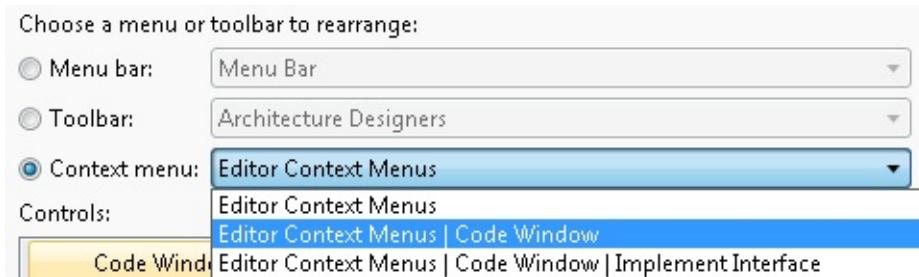
For information about the Toolbars tab, see vstipEnv0030, [01.11 Customize Your Toolbars in Visual Studio 2010: Toolbars Tab](#), page 27.



As you can see, the Customize dialog box is fairly complex, so let's break it down into its parts as we look at an example.

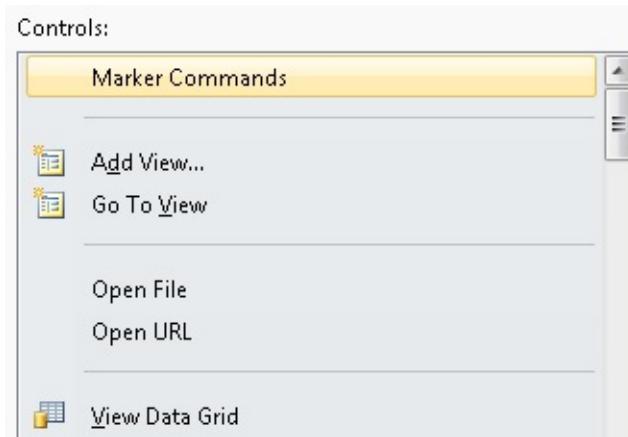
## Rearrange

First is the choice of menu or toolbar to modify. In this case, choose the Editor Context Menus | Code Window option, which is what you see when you right-click while writing code:



## Controls

Next is the Controls area that shows the items on the menu or toolbar you have chosen to modify. For this example, it shows the items available when you right-click in a code window:



Remember that not all the items you see are available all the time because these items show up only in the proper context. So while it seems you have a large number of buttons currently available, when you right-click in your code window, this is an example of what you will currently see:



## Buttons

Finally, let's look at the area of the dialog box that has all the buttons that actually

*perform actions:*

- **Add Command**

Lets you add a new item to the existing menu or toolbar.

- **Add New Menu**

Creates a new menu in the existing menu or toolbar.

- **Delete**

Removes the current item from the Controls area.

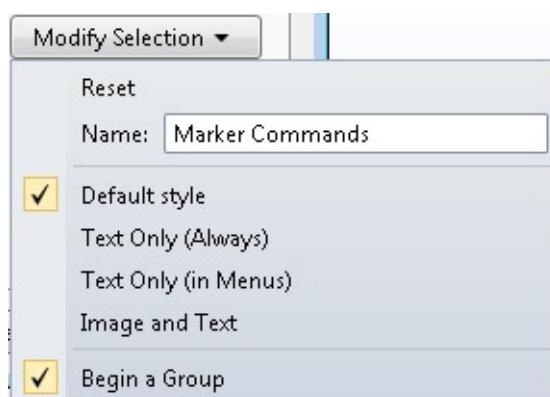
- **Move [Up or Down]**

Changes the location of the item in the Controls area.



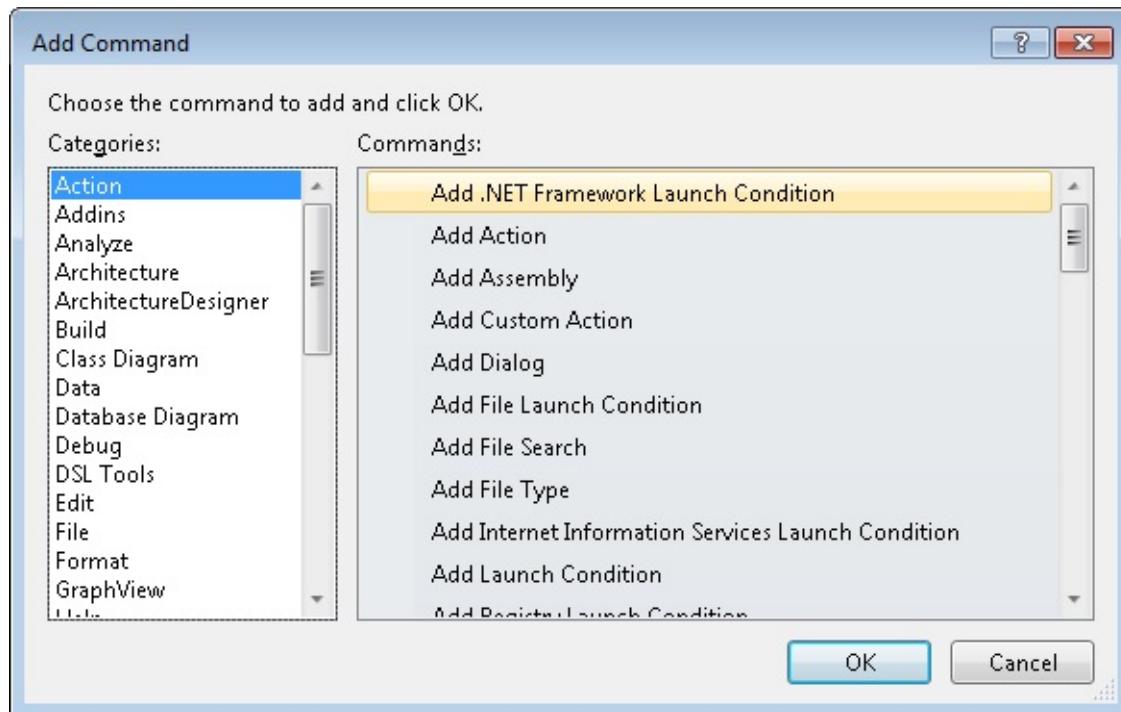
## Modify Selection

Choosing Modify Selection enables you to make changes to the existing item in the Controls area, such as resetting it to the default settings, changing the name, and modifying text visibility options. Modify Selection also enables you to make a new group on the menu or toolbar:

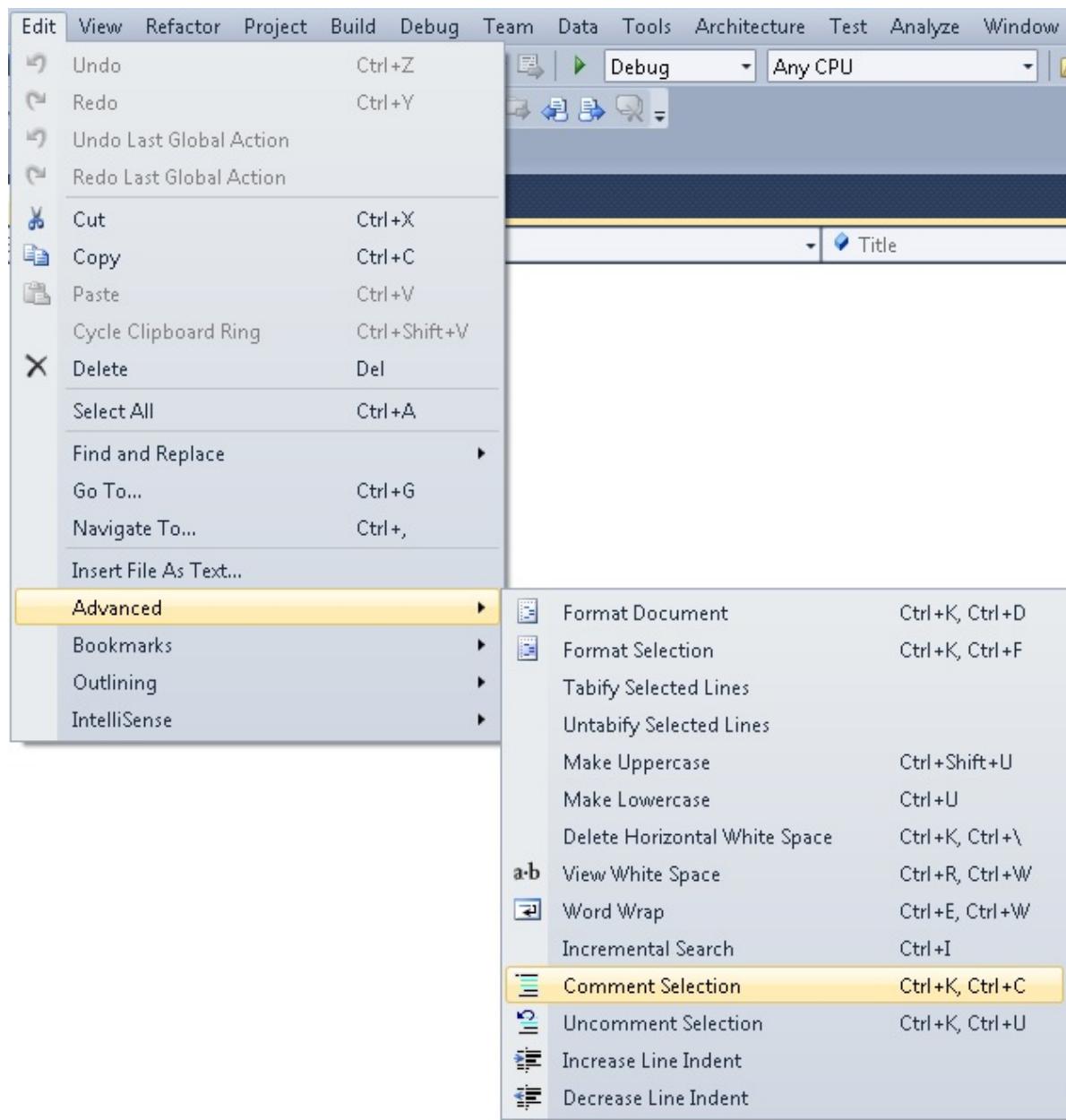


Finally, the Reset All option resets every item in the Controls area to its default settings. This capability is particularly useful if you have made a lot of changes.

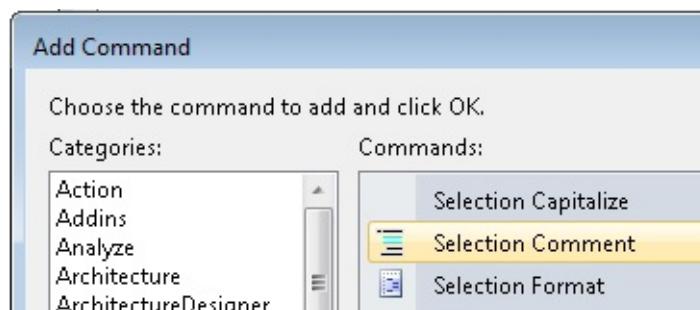
Getting back to our example: Let's assume you want to add the comment and uncomment items to the context menu so that you can use them when you select some code. First, click Add Command to bring up the Add Command dialog box:



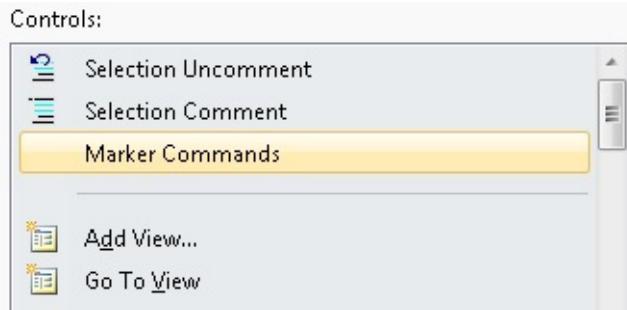
Now you need to figure out where the comment and uncomment items are located. How would you do this? Well, the best path is usually to see whether the item can be found on a menu somewhere and then use that as a clue:



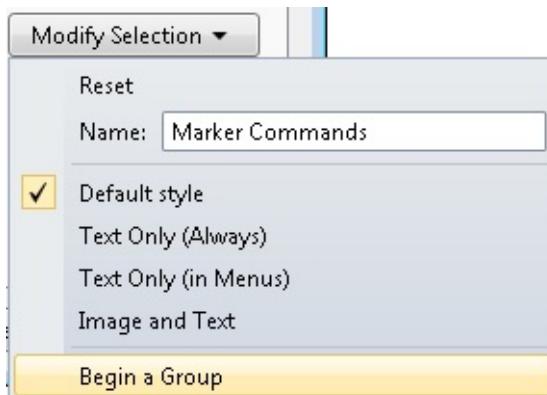
Because the items you want are off the Edit menu, you can search there first. It turns out the items you want are called Selection Comment and Selection Uncomment:



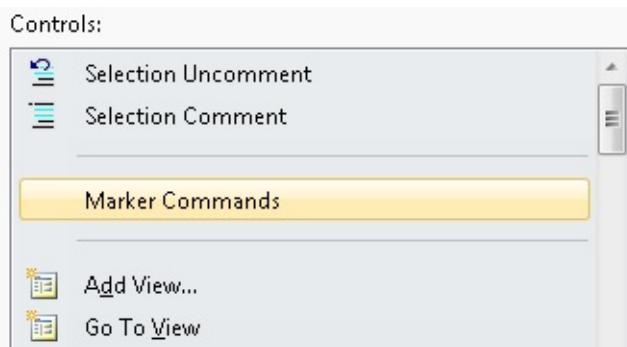
Find each one of these items, and click OK to add them to the Controls list:



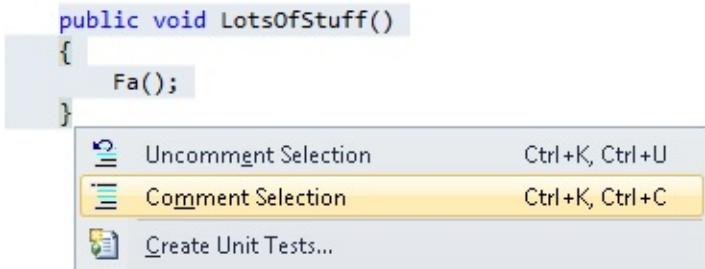
It would be nice to have these buttons in their own group, so select the item in the Controls dialog box, shown in the preceding illustration, where you would like the group line to be (Marker Commands, in this case), and then click Modify Selection and choose Begin A Group:



This creates a new group line, and your commands are in their own group:



Click Close and go to any code area. Select some code, right-click, and select Comment Selection:



It works perfectly, and you are all set to begin making your own modifications to your environment:

```
//public void LotsOfStuff()
//{
//    Fa();
//}
```

## 01.13 Visual Studio Logging

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0048

There's no doubt Visual Studio is an awesome piece of software, but occasionally you might run into a problem loading it. Did you know that it comes with a logging switch? While the documentation (<http://msdn.microsoft.com/en-us/library/ms241272.aspx>) is lacking, the community comment contributed by Paul Harrington on the Visual Studio team helps a great deal.

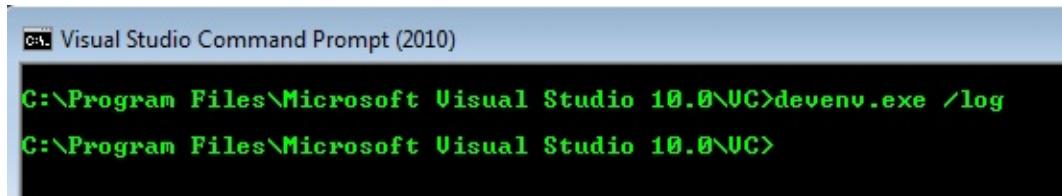
Essentially, the syntax is as follows:

```
devenv.exe /log [filename]
```

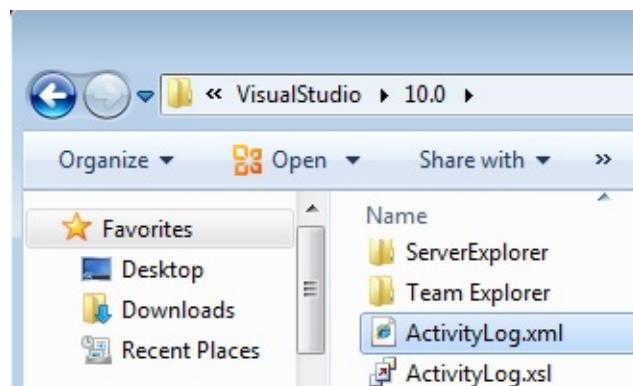
The [filename] is optional and, if not specified, the ActivityLog.xml file is called by default. The path is to the log file is:

```
%APPDATA%\Microsoft\VisualStudio\<version>\ActivityLog.xml
```

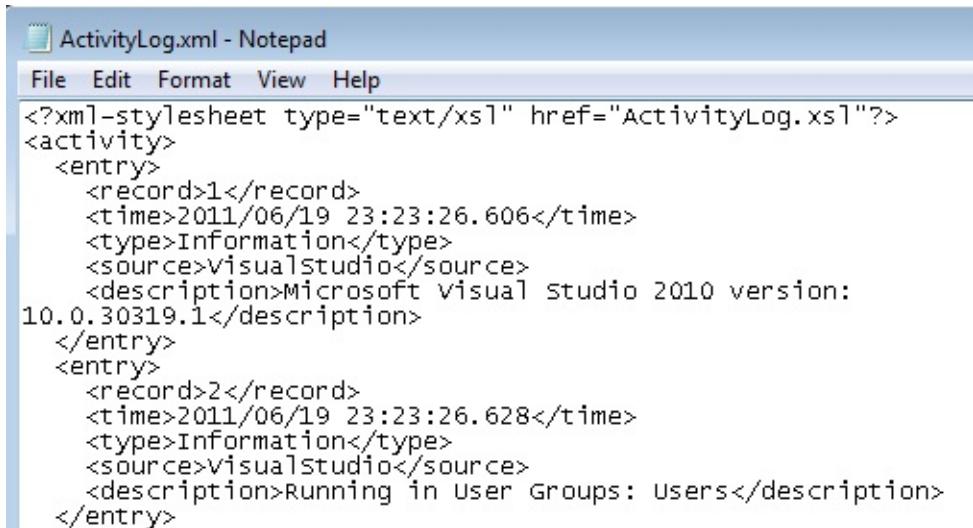
Give it a try. Go to the Visual Studio command prompt, and enter **devenv.exe /log**:



You can then navigate to the file location:



When you open the log file, the following illustration provides an example of what you might see:



```
<?xmlstylesheet type="text/xsl" href="ActivityLog.xsl"?>
<activity>
  <entry>
    <record>1</record>
    <time>2011/06/19 23:23:26.606</time>
    <type>Information</type>
    <source>VisualStudio</source>
    <description>Microsoft visual studio 2010 version: 10.0.30319.1</description>
  </entry>
  <entry>
    <record>2</record>
    <time>2011/06/19 23:23:26.628</time>
    <type>Information</type>
    <source>visualstudio</source>
    <description>Running in User Groups: Users</description>
  </entry>

```

Fortunately, an XML style sheet (XSL) comes with the data, so if you view the XML file in your browser, you can see a much cleaner view:

## Activity Monitor Log

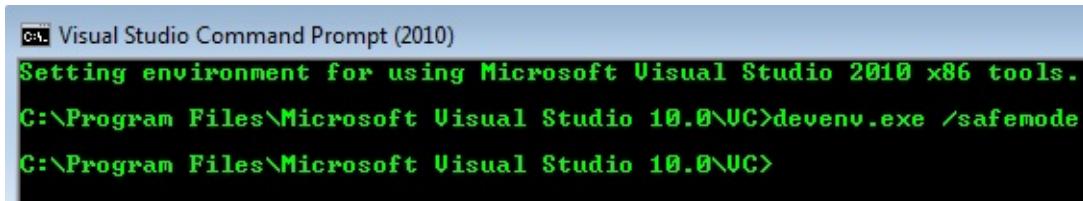
infos	276
warnings	1
errors	0
<b># Type Description</b>	
1	Microsoft Visual Studio 2010 version: 10.0.30319.1
2	Running in User Groups: Users
3	Available Drive Space: C:\ drive has 83539099648 bytes; D:\ drive has 51970506752 bytes
4	Internet Explorer Version: 8.0.7600.16821
5	.NET Framework Version: 4.0.31106.0

Now you can easily see the logging information and look for any issues.

## 01.14 Visual Studio Safe Mode

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0050

Occasionally you have a situation where Visual Studio might not start up correctly or at all. Using Visual Studio in safe mode, you can load only the default environment, services, and shipped versions of third-party packages to see whether the problem is caused by one or more third-party add-ins. Just go to the Visual Studio command prompt, and type **devenv.exe /safemode**:



```
Visual Studio Command Prompt (2010)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Program Files\Microsoft Visual Studio 10.0\VC>devenv.exe /safemode
C:\Program Files\Microsoft Visual Studio 10.0\VC>
```

### NOTE

Although I don't show it here, I suggest using the Visual Studio logging feature before running safe mode to see whether it can help you determine the source of the problem. For more information, see vstipEnv0048, [01.13 Visual Studio Logging](#), page 37.

When Visual Studio starts up, it indicates it is running in safe mode in the title bar:



From here, you can start determining what might have caused Visual Studio to fail and remedy the issue.

## 01.15 The ResetSettings Switch

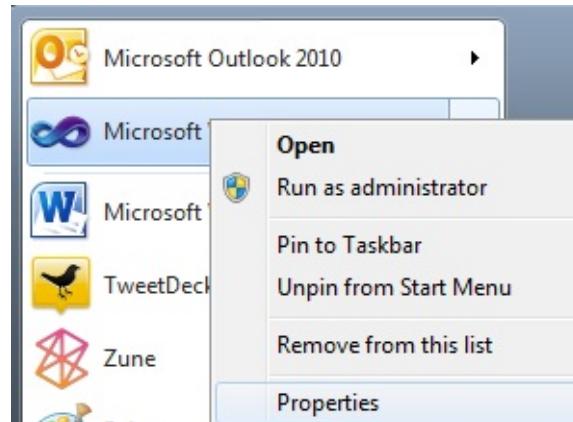
VERSIONS	2005, 2008, 2010
CODE	vstipEnv0047

Visual Studio supports several switches. One of these is the /ResetSettings switch. When used by itself, it resets Visual Studio to the default settings you initially chose during install. That's nice, but an even better option is available that can be particularly useful for people in other scenarios.

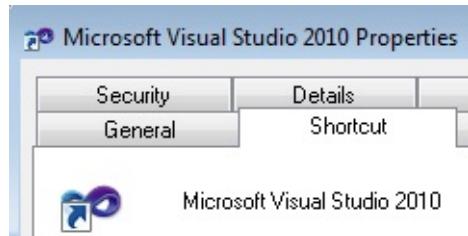
Let's take a classic example: You might have two (or more) monitors at work, but when you get home, you work with just one monitor. Your window layouts (among other things) could be very different in each place. In vstipEnv0040, [03.29 Export Your Window Layouts](#) (page 134), I showed you how to export just your window layouts. Using the exported information, you could create and use two different window layouts: one for work and one for home. This tip shows you how to do this when using two different machines or the same machine at work and home.

### Two Different Machines

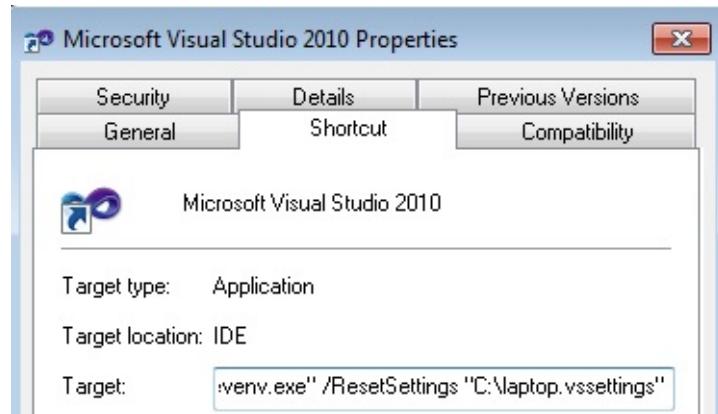
The question is: What do you do once you have exported the window layouts? Well, now you put the .vssettings files where you can easily get to them on your machines and then you go to the Properties dialog box of the Visual Studio program icon:



Click the Shortcut tab:



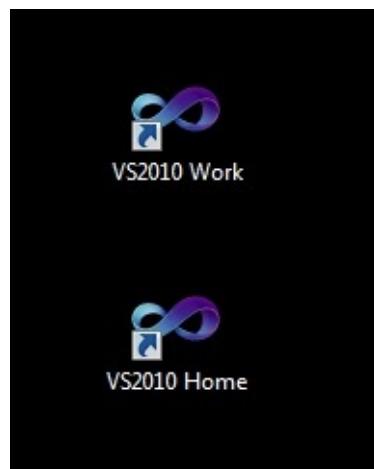
In the Target area, type **/ResetSettings [settings file]**, where [settings file] is the path to the settings file for one of your layouts:



Now Visual Studio loads up with the settings appropriate for your machine.

## Same Machine

What if you use the same machine for home and work, like a laptop? Just make two copies of the Visual Studio program icon, put them somewhere (on your Desktop, most likely), and give them names:



Now just follow the steps for the different machines for each icon, and you can use one icon when you are at work to get the work window layouts and the other for home with the home window layout.

# Chapter 2. Projects and Items

*“He recalled his exertions and solicitations, and the history of his project [...], which had been accepted for consideration [...]”*

— Leo Tolstoy “War and Peace”

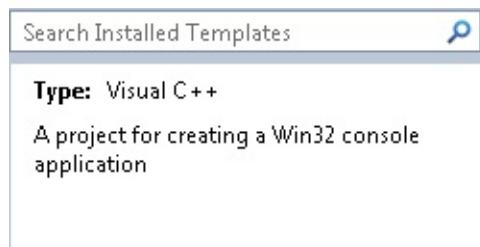
After you get past your initial customization of Visual Studio, you will start creating projects and items to get your work done. This section contains a group of resources that you will find useful early on. Some will definitely be more useful to beginners (for example, searching templates), and others will apply to more advanced users (such as creating custom templates).

On the subject of custom templates, make sure you read though (and practice) how to create them. Of all the topics in this chapter, I feel that creating custom project and item templates will save you the greatest amount of time. That’s a pretty bold statement—but I have seen properly set up templates save untold hours for developers.

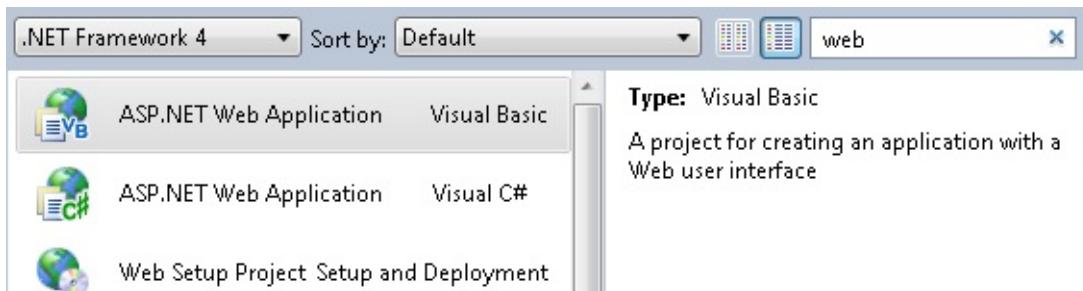
## 02.01 Search for Project Templates in the New Project Dialog Box

<b>DEFAULT</b>	Ctrl+Shift+N (new project dialog box); Ctrl+E (puts cursor in search box)
<b>VISUAL BASIC 6</b>	Ctrl+Shift+N (new project dialog box); Ctrl+N (new project); Ctrl+E (puts cursor in search box)
<b>VISUAL C# 2005</b>	Ctrl+Shift+N (new project dialog box); Ctrl+E (puts cursor in search box)
<b>VISUAL C++ 2</b>	Ctrl+Shift+N (new project dialog box); Ctrl+E (puts cursor in search box)
<b>VISUAL C++ 6</b>	Ctrl+Shift+N (new project dialog box); Ctrl+E (puts cursor in search box)
<b>VISUAL STUDIO 6</b>	Ctrl+N (new project dialog box); Ctrl+E (puts cursor in search box)
<b>WINDOWS</b>	Alt,F, N, P (new project dialog box); Alt,F, D, N (add new project)
<b>MENU</b>	File   New Project; File   Add New Project
<b>COMMAND</b>	File.NewProject; File.AddNewProject
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipProj0001

Did you know that you can search for templates in the New Project dialog box? Look in the upper-right corner, and notice the new search area.



Click there or press Ctrl+E, and type the word web into the search box. The following illustration shows what you should expect to see.



## Good News

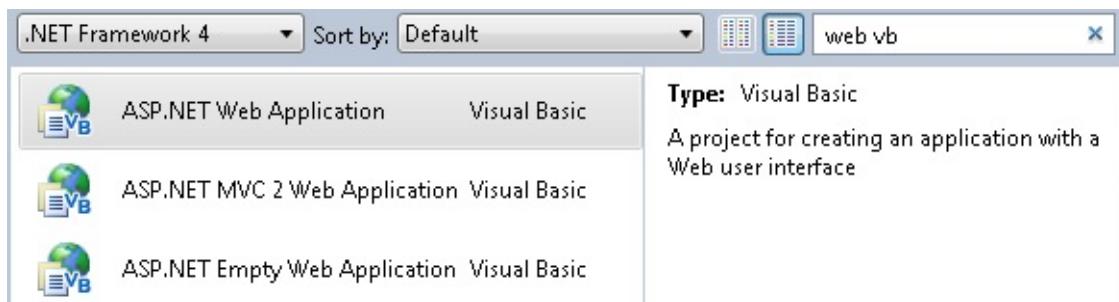
The search populates the middle pane with results from the recent, installed, or online lists, depending on which category you select.

## Bad News

The search doesn't automatically filter the results according to your preferred language, and it doesn't support any advanced search options, such as Boolean searches, regular expressions, and so on.

## More Good News

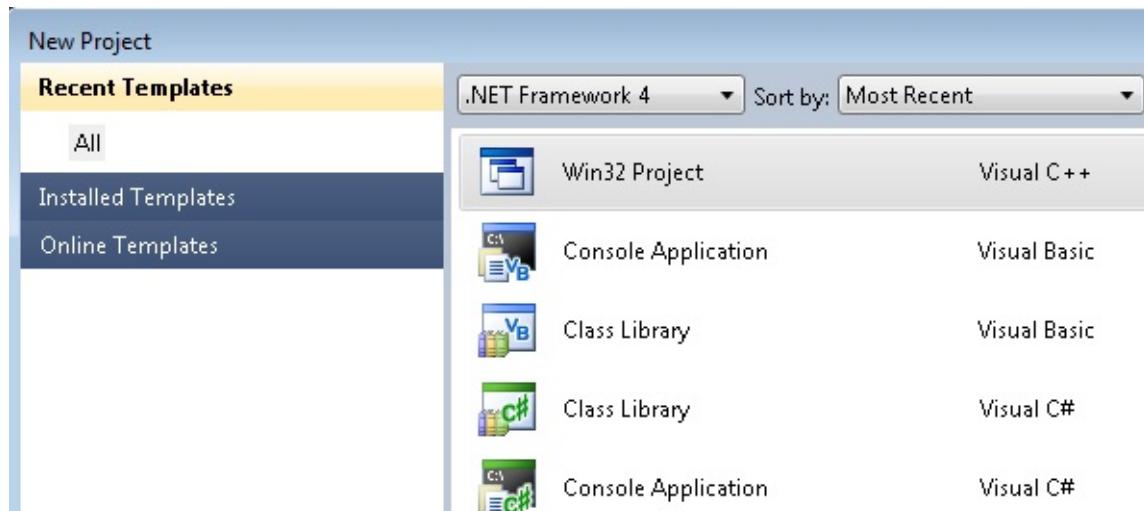
In most cases, you can easily filter on your language by simply typing in an abbreviation of your language (C#, VB, F#, or C++).



## 02.02 Recent Project Templates in the New Project Dialog Box

<b>DEFAULT</b>	Ctrl+Shift+N (new project dialog box)
<b>VISUAL BASIC 6</b>	Ctrl+Shift+N (new project dialog box); Ctrl+N (new project)
<b>VISUAL C# 2005</b>	Ctrl+Shift+N (new project dialog box)
<b>VISUAL C++ 2</b>	Ctrl+Shift+N (new project dialog box)
<b>VISUAL C++ 6</b>	Ctrl+Shift+N (new project dialog box)
<b>VISUAL STUDIO 6</b>	Ctrl+N (new project dialog box)
<b>WINDOWS</b>	Alt,F, N, P (new project); Alt,F, D, N (add new project)
<b>MENU</b>	File   New Project; File   Add New Project
<b>COMMAND</b>	File.NewProject; File.AddNewProject
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipProj0002

In the Visual Studio 2010 New Project dialog box, you can get a list of your five most recently used templates. Just click Recent Templates to see a list of the templates you have used.



## 02.03 Using Older Frameworks with Multi-Targeting

<b>DEFAULT</b>	Ctrl+Shift+N (new project dialog box)
<b>VISUAL BASIC 6</b>	Ctrl+Shift+N (new project dialog box); Ctrl+N (new project)
<b>VISUAL C# 2005</b>	Ctrl+Shift+N (new project dialog box)
<b>VISUAL C++ 2</b>	Ctrl+Shift+N (new project dialog box)
<b>VISUAL C++ 6</b>	Ctrl+Shift+N (new project dialog box)
<b>VISUAL STUDIO 6</b>	Ctrl+N (new project dialog box)
<b>WINDOWS</b>	Alt,F, N, P (new project); Alt,F, D, N (add new project)
<b>MENU</b>	File   New Project; File   Add New Project
<b>COMMAND</b>	File.NewProject; File.AddNewProject
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipProj0005

Even if you use an older version of the Microsoft .NET Framework, you can still use all the great features in Visual Studio 2008 and Visual Studio 2010 through multi-targeting.

When you create a new project, locate the drop-down list of supported .NET Framework versions and simply choose the one you prefer. You get to use most of the great features in the new IDE but still keep your older version of the .NET Framework. The following graphic shows the New Project dialog box in Visual Studio 2010.

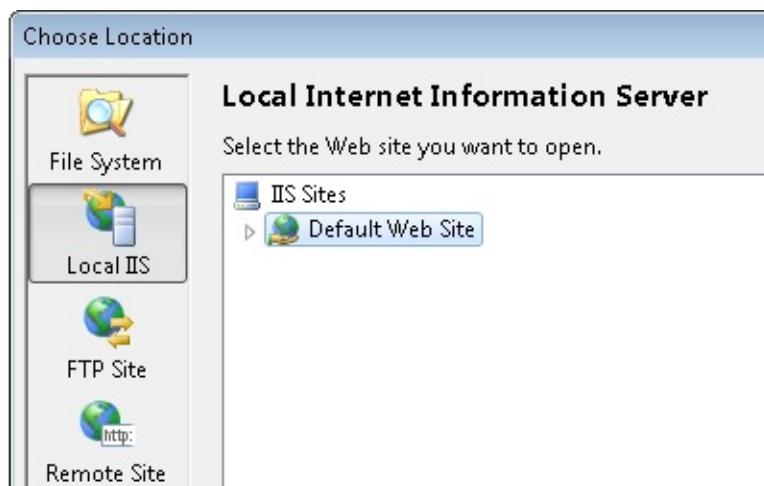


## 02.04 Create Web Application or Virtual Directory in IIS

<b>DEFAULT</b>	Shift+Alt+N
<b>VISUAL BASIC 6</b>	Shift+Alt+N
<b>VISUAL C# 2005</b>	Shift+Alt+N
<b>VISUAL C++ 2</b>	Shift+Alt+N
<b>VISUAL C++ 6</b>	Shift+Alt+N
<b>VISUAL STUDIO 6</b>	Shift+Alt+N
<b>WINDOWS</b>	Alt,F, N, W
<b>MENU</b>	File   New Web Site
<b>COMMAND</b>	File.NewWebSite
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEnv0058

How can you create Web Applications and Virtual Directories in Internet Information Server from inside Visual Studio? Just select File | New Web Site, and click Browse in the lower-right corner.

In the Choose Location dialog box, select Local IIS, and pick the website in which you want to create the new item.



In the upper-right corner of the dialog box, notice the three buttons, as shown in

the following illustration.



The button to the far left creates a new Web Application.



The middle button creates a new Virtual Directory.

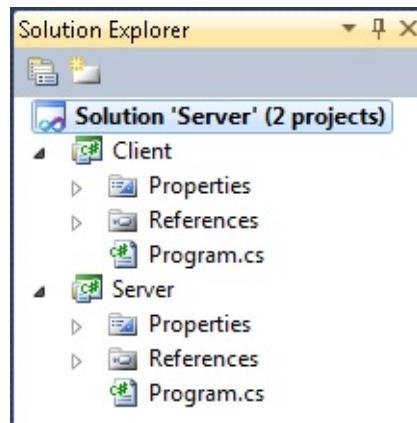


You can pick whichever one you want to create, without ever leaving Visual Studio.

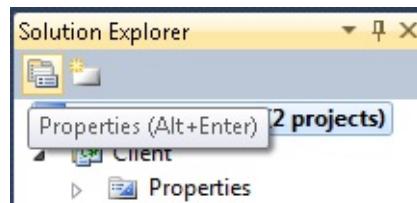
## 02.05 Multiple Startup Projects

<b>DEFAULT</b>	Alt+Enter (in Solution Explorer)
<b>VISUAL BASIC 6</b>	Alt+Enter (in Solution Explorer)
<b>VISUAL C# 2005</b>	Alt+Enter (in Solution Explorer)
<b>VISUAL C++ 2</b>	Alt+Enter (in Solution Explorer)
<b>VISUAL C++ 6</b>	Alt+Enter (in Solution Explorer)
<b>VISUAL STUDIO 6</b>	Alt+Enter (in Solution Explorer)
<b>WINDOWS</b>	Alt, P, P
<b>MENU</b>	Project   Properties [with Solution Selected in Solution Explorer]; [Right-Click the solution in Solution Explorer]   Properties
<b>COMMAND</b>	Project.Properties
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipEnv0015

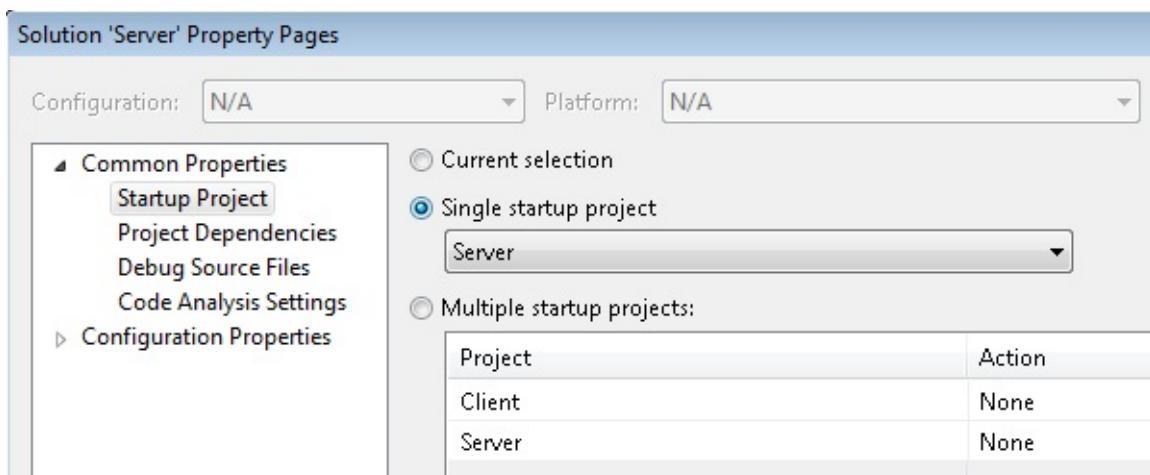
It's common for developers to work with multiple projects. For example, consider a classic Client/Server scenario: One project includes all the elements shown here:



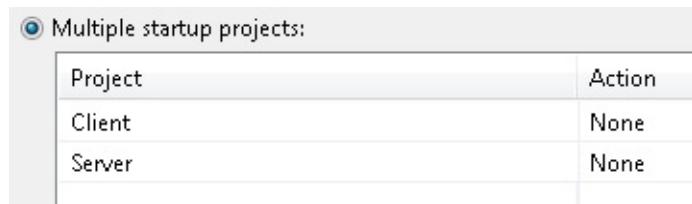
In this case, setting a single Startup Project isn't sufficient; you want both these projects to start up when you press F5. Just click the solution in Solution Explorer, and then click the Properties button (Alt+Enter) at the top.



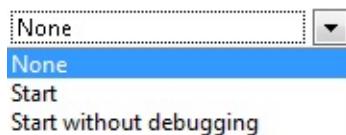
Make sure you are in the Common Properties | Startup Project area, and you should see a dialog box similar to the following.



Notice that, currently, Single Startup Project is selected, but that isn't what you need. Instead, select Multiple Startup Projects.



Now you need to indicate which action each project should take when you press F5. Click the drop-down in the Action field.



As shown in the preceding illustration, you see the following choices:

- **None**

Don't start this project.

- **Start**

Start with debugging.

- **Start without debugging**

Start without attaching the debugger.

For this example, you would choose Start for both projects.

Multiple startup projects:	
Project	Action
Client	Start
Server	Start

Now both projects start when you press F5. But there's just one little problem: The Client project launches first, and you need the Server project to launch first. To set the launch order, use the buttons to the far right of the project list, as shown in the following illustration.

Multiple startup projects:	
Project	Action
Client	Start
Server	Start

Buttons on the right side of the list allow you to move the selected project up or down in the order.

These buttons move the selected project up or down in the list so that you can arrange them to start in the order you would like. In this case, as shown in the following illustration, I've selected the Server project and then clicked the Move button to move it up in the startup order.

Multiple startup projects:	
Project	Action
Server	Start
Client	Start

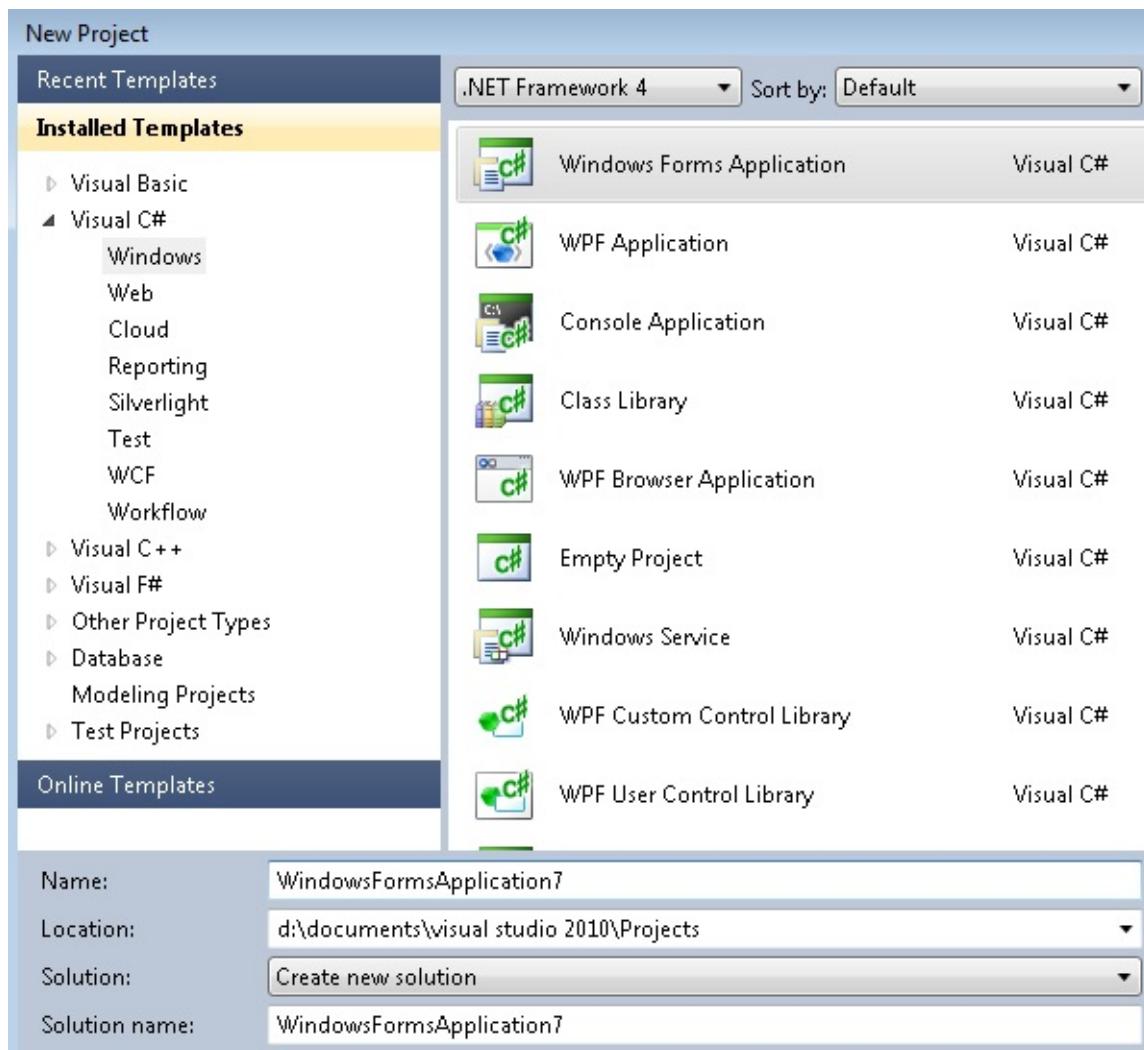
With the Server first in the list, you're all set. When you press F5, you can see the server start and then the client.

One interesting side effect of setting multiple startup projects is that the bold project name you normally see in Solution Explorer isn't there—because there is more than one startup project.

## 02.06 Change the Default New Project Location

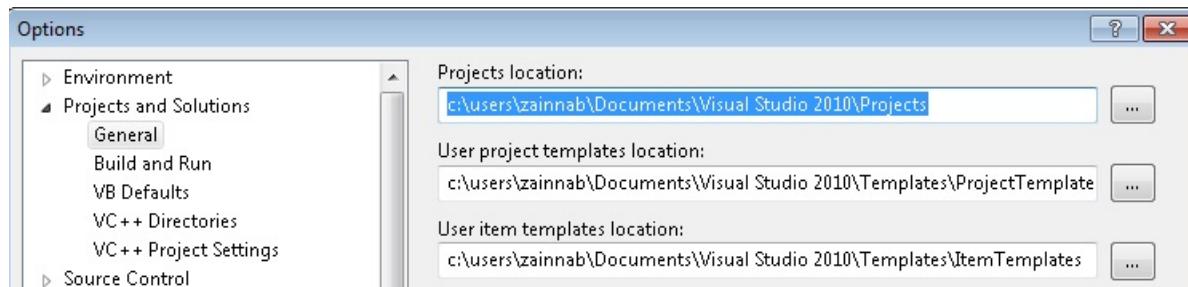
<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Projects And Solutions   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0006

You probably know that you can change the location for a new project in the New Project dialog box by entering a different location in the Location field.



But if you do this often, did you know that you can change the default location so that you don't have to keep typing in custom paths? To change the default, select

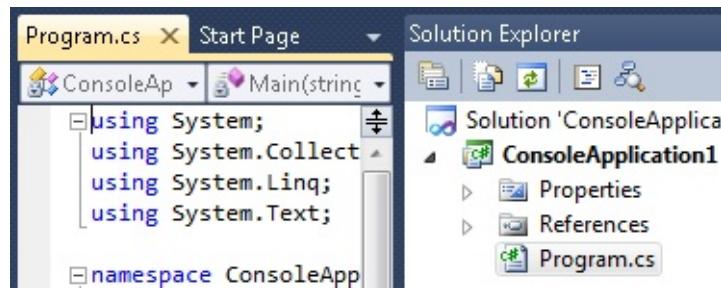
Tools | Options | Projects And Solutions | General from the menu bar. You'll see an Options dialog box, shown below, where you can change several default paths to suit your needs.



## 02.07 Track Active Item in Solution Explorer

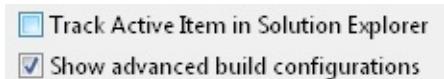
WINDOWS	Alt,T, O
MENU	Tools   Options   Projects and Solutions   General
COMMAND	View.TrackActivityinSolutionExplorer
VERSIONS	2005, 2008, 2010
CODE	vstipProj0011

By default, Visual Studio tracks the file you are currently editing in Solution Explorer. The Solution Explorer tool window highlights the current file.



As you switch between files in the editor, notice that Solution Explorer automatically highlights the file you're currently editing. This is a great way to keep track of where you are in the solution when you are working with a lot of files.

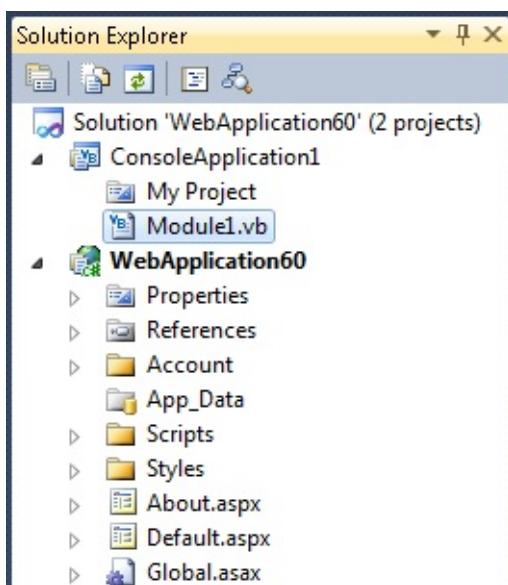
If you don't like the feature, you can turn it off. Just select Tools | Options | Projects And Solutions | General, and clear the Track Active Item In Solution Explorer check box shown in the following illustration.



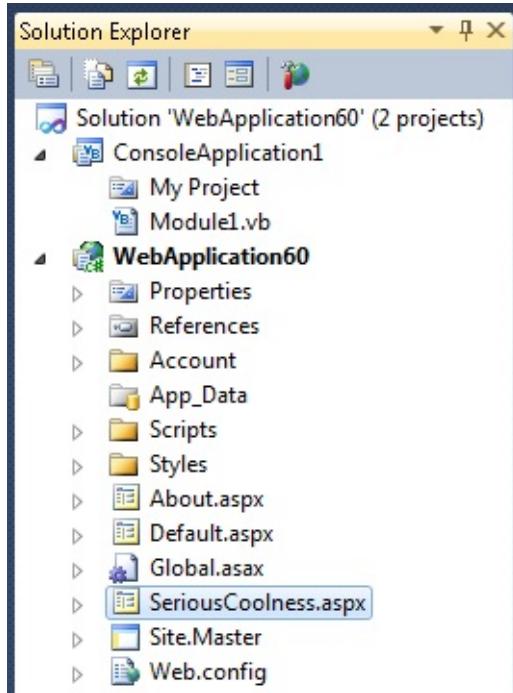
## 02.08 Type-Ahead Selection Support in Solution Explorer

<b>DEFAULT</b>	Ctrl+Alt+L
<b>VISUAL BASIC 6</b>	Ctrl+R; Ctrl+Alt+L
<b>VISUAL C# 2005</b>	Ctrl+W, S; Ctrl+W, Ctrl+S; Ctrl+Alt+L
<b>VISUAL C++ 2</b>	Alt+0; Ctrl+Alt+L
<b>VISUAL C++ 6</b>	Ctrl+Alt+L
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+J
<b>WINDOWS</b>	Alt,V, P
<b>MENU</b>	View   Solution Explorer
<b>COMMAND</b>	View.SolutionExplorer
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0010

Have you ever had a big list of files in Solution Explorer and wanted to jump to a specific file very quickly? Just click anywhere in Solution Explorer, and start typing the name of the file you want. For example, suppose you have a solution with multiple projects:



Assume you need to find a file called SeriousCoolness. To find it, click in Solution Explorer and then just start typing the name. Solution Explorer finds the file for you as you type.



What if you don't know the whole name—just that it starts with an "S"? No worries! Just type **S** several times, and the selection cycles through all the files that begin with that letter.

#### NOTE

The type-ahead feature works only with items that have been expanded, so if you have collapsed folders or projects in Solution Explorer, the tool cannot search within those areas.

## 02.09 Using Solution Folders

<b>WINDOWS</b>	(with Solution selected) Alt,P, D
<b>MENU</b>	(with Solution selected) Project   Add New Solution Folder; [Right-Click Solution]   Add   New Solution Folder
<b>COMMAND</b>	(with Solution selected) Project.AddNewSolutionFolder
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0009

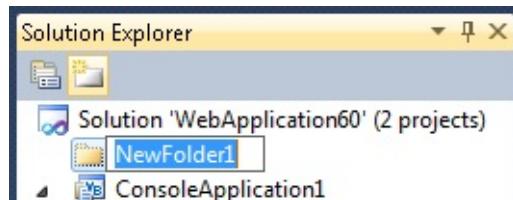
Did you know that Visual Studio provides special folders that can help you organize large solutions? They are called, appropriately enough, “Solution Folders.”

### NOTE

Solution Folders are an organizational tool in Solution Explorer; creating one doesn’t create a corresponding Windows file system folder. Microsoft recommends that you organize your projects on disk in the same way that you organize them in the Solution Folder. But of course, you’re free to organize them as you like.

## Adding Solution Folders

To create a Solution Folder, right-click your solution (or, with the solution selected, go to Project | Add New Solution Folder). Solution Explorer adds a new folder, which you can type a name for.



After you enter a name for the new folder, press Enter, and you’re done. So what can you actually do with these things? It turns out, quite a lot:

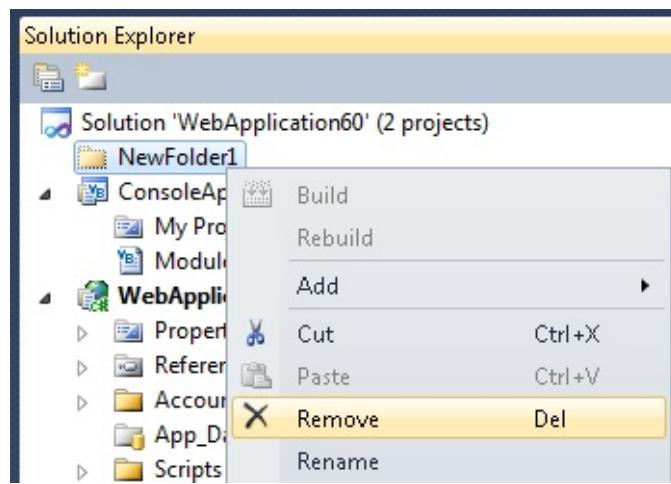
- Move or add projects to them. Solution Folders can be nested to create greater organizational structure.
- Add, delete, or rename Solution Folders at any time, if the organizational requirements of your solution change.
- Unload all projects in a Solution Folder to make them temporarily unavailable

for building.

- Collapse or hide entire Solution Folders so that you can work more easily in Solution Explorer. Hidden projects are built when you build the solution.
- Build or rebuild all the projects. The projects are built in the order specified by the project dependencies.

## Removing Solution Folders

If you want to get rid of a folder, just right-click it and choose Remove to delete it, or alternatively, select it and press the Delete key.

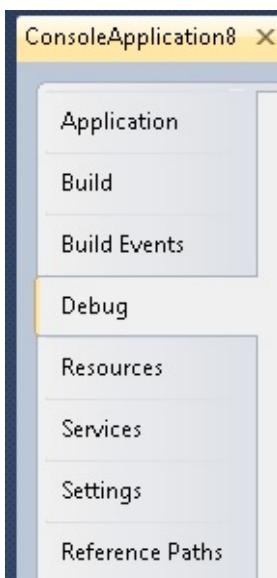


## 02.10 Navigating Property Tabs in the Project Properties

<b>DEFAULT</b>	Ctrl+PgUp; Ctrl+PgDn
<b>VISUAL BASIC 6</b>	Ctrl+PgUp; Ctrl+PgDn
<b>VISUAL C# 2005</b>	Ctrl+PgUp; Ctrl+PgDn
<b>VISUAL C++ 2</b>	Ctrl+PgUp; Ctrl+PgDn
<b>VISUAL C++ 6</b>	Ctrl+PgUp; Ctrl+PgDn
<b>VISUAL STUDIO 6</b>	Ctrl+PgUp; Ctrl+PgDn
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Window.PreviousTab; Window.NextTab
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0023

When you are looking at your project's properties, you might have wondered whether you can navigate among the property tabs by using the keyboard.

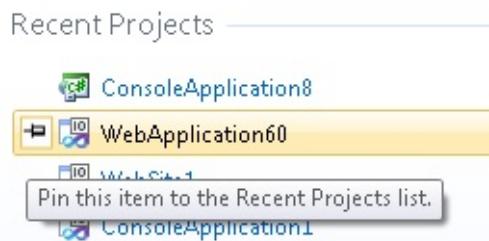
It turns out that you can. Just use Ctrl+PgUp or Ctrl+PgDn to move between the property tabs. This also works for properties in C++ projects if you want to quickly navigate among the categories.



## 02.11 Pin a Project to the Recent Projects List

VERSIONS	2010
CODE	vstipTool0003

Tired of your projects getting pushed out of the Recent Projects list on the Start Page? You can pin projects to the Recent Projects list in Visual Studio 2010 so that they stay around until you unpin them.

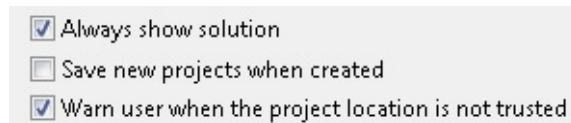


Pinned projects do not stay at the top of the list; instead, they're sorted according to when you use them. In other words, the most recent project is on top—pinned or not. Pinning guarantees only that the project will not be pushed out of the list.

## 02.12 Creating Temporary Projects

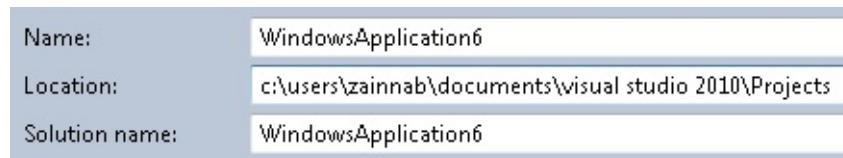
<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Projects and Solutions   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0010

Temporary projects are particularly useful for showing a colleague some trick or technique quickly, or for performing ad hoc demos. To create temporary projects, select Tools | Options | Projects And Solutions | General and clear the Save New Projects When Created check box, as shown in the following illustration.

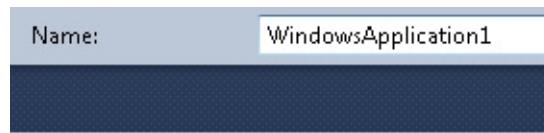


While convenient, the option has some side effects. For example, when you subsequently create a new project, the New Project dialog box does not show the usual “save” fields at the bottom of the dialog box.

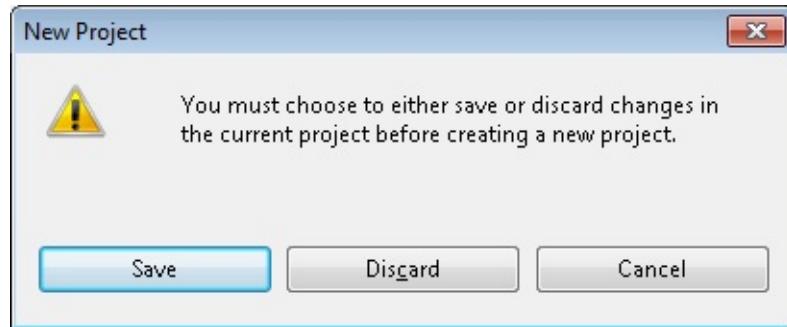
Before:



After:



The good news is that now you can create projects all day long but can choose to either save or discard the changes when the solution is closed.



#### NOTE

You can still save changes to a project—even a temporary project—anytime you like if you decide you want to keep the code around. When you decide to save, your AutoRecover settings take over. For more information, see vstipEnv0019 (01.05 AutoRecover, page 10).

## 02.13 Create Your Own Item Template

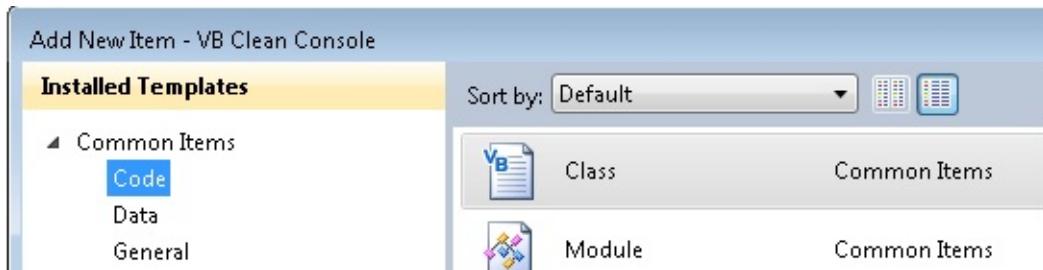
<b>WINDOWS</b>	Alt,F, E
<b>MENU</b>	File   Export Template
<b>COMMAND</b>	File.ExportTemplate
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipProj0013

Have you ever used or created a template in Microsoft Word, Excel, or PowerPoint? Unless you live in a cave, the answer is most likely “yes.” Just as with the Microsoft Office products, you can create and use your own templates in Visual Studio. This tip shows you how to make your own item template. Sometimes you just want to customize an individual item that you use frequently in projects. Class files are a perfect example of this type of scenario. Here’s an example.

Create a new project, and then add a class to it (Ctrl+Shift+A).

### NOTE

The process is the same regardless of which language you’re using.



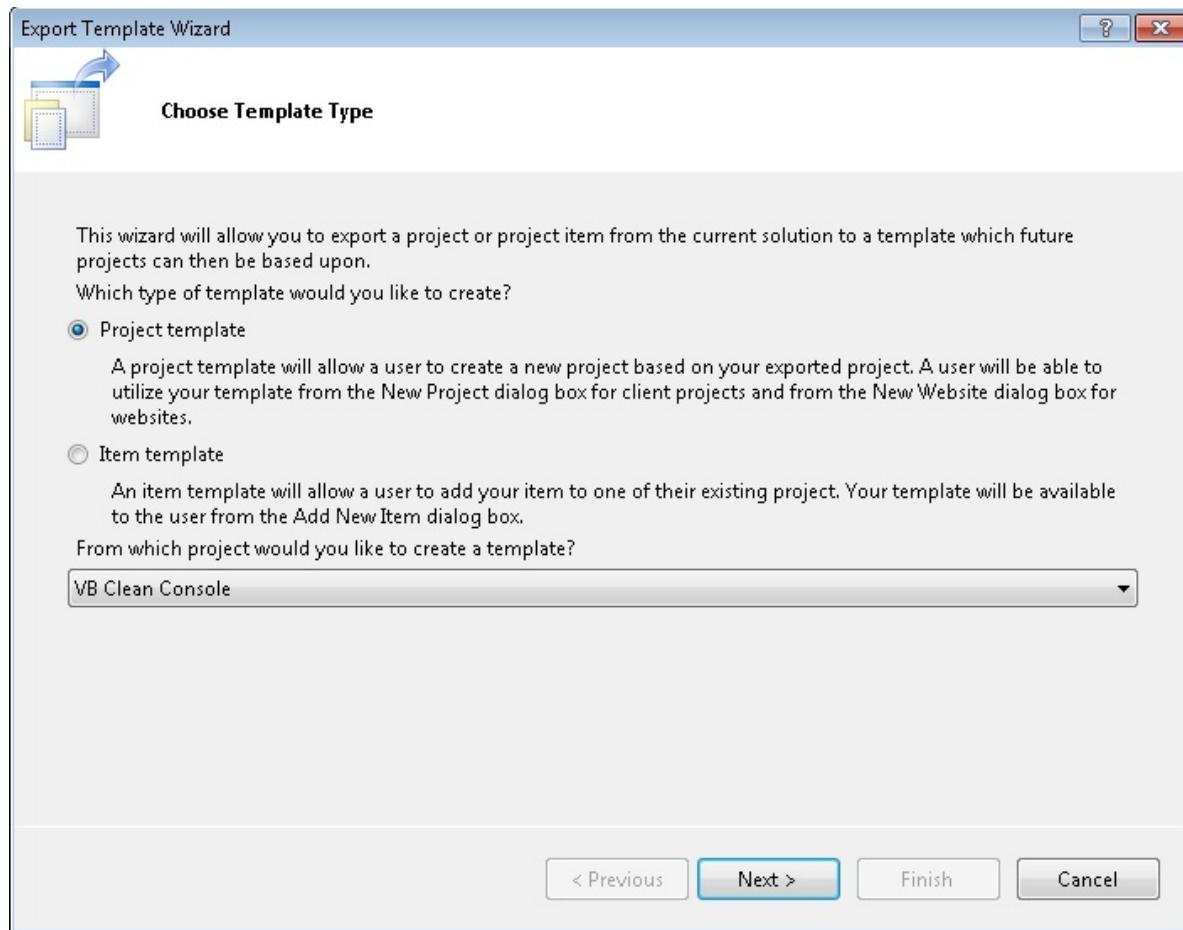
Now modify the class code so that it looks the way you would like your item template to look, and save your changes to the file.

```
3 Imports System.DirectoryServices  
4  
5 ' company required comments here  
6 Friend Class Class1  
7     ' todo put your connection string here  
8  
9 End Class  
10
```

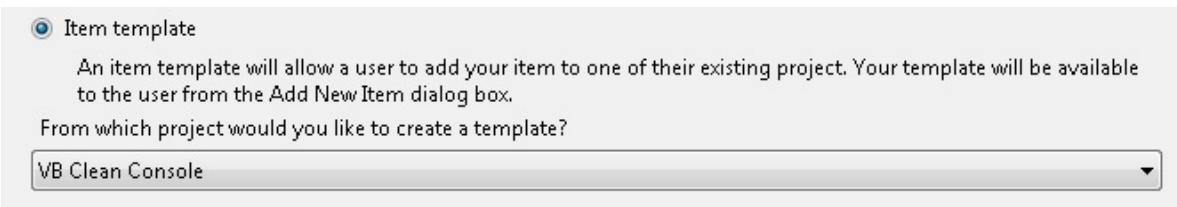
At this point, you can export the item template so that you can use it in future projects. Select File | Export Template to start the Export Template Wizard.

#### NOTE

You are prompted to save changes to your project if you haven't already done so.



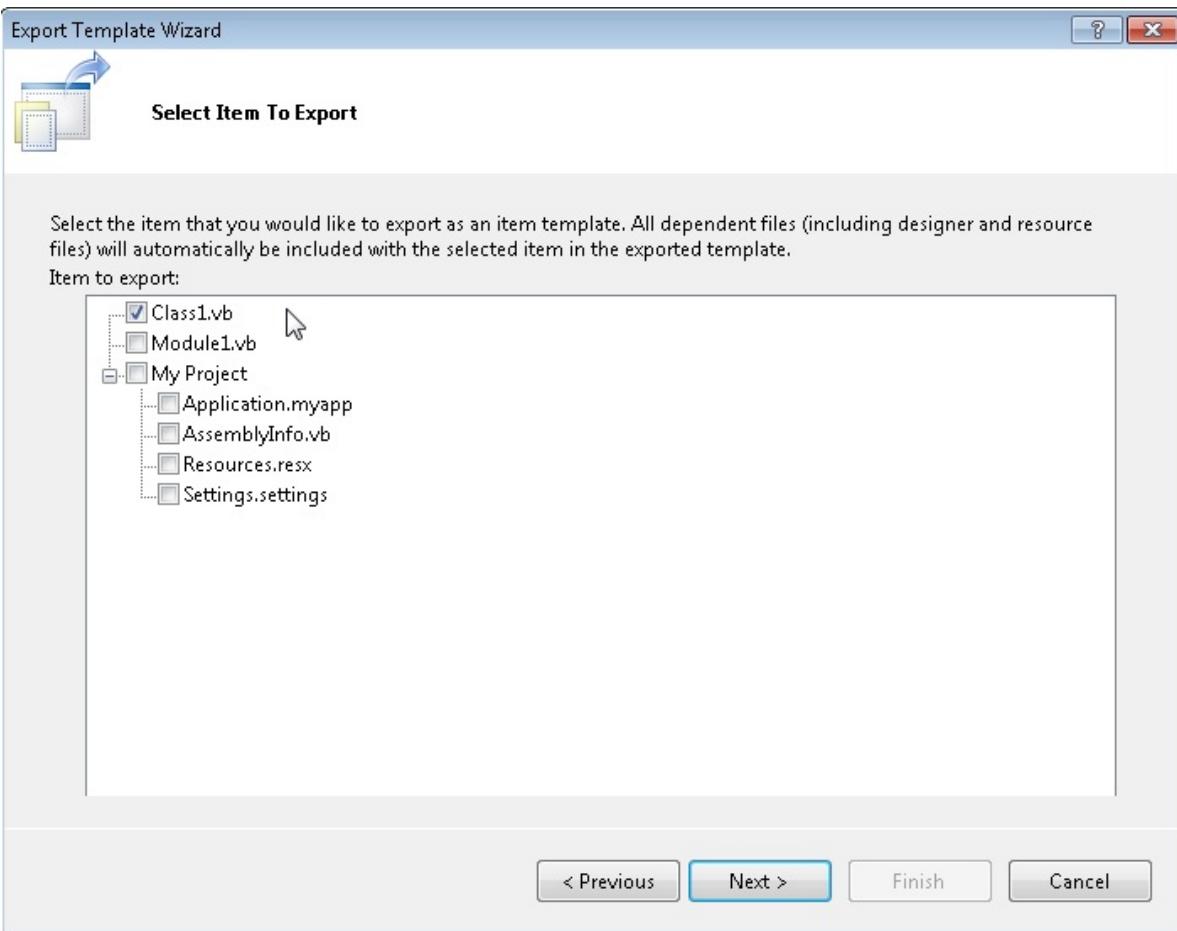
Select Item Template, and select the project that currently contains the item you want to export (if you have more than one project in your solution).



Click Next and then select the item to export as a template.

#### NOTE

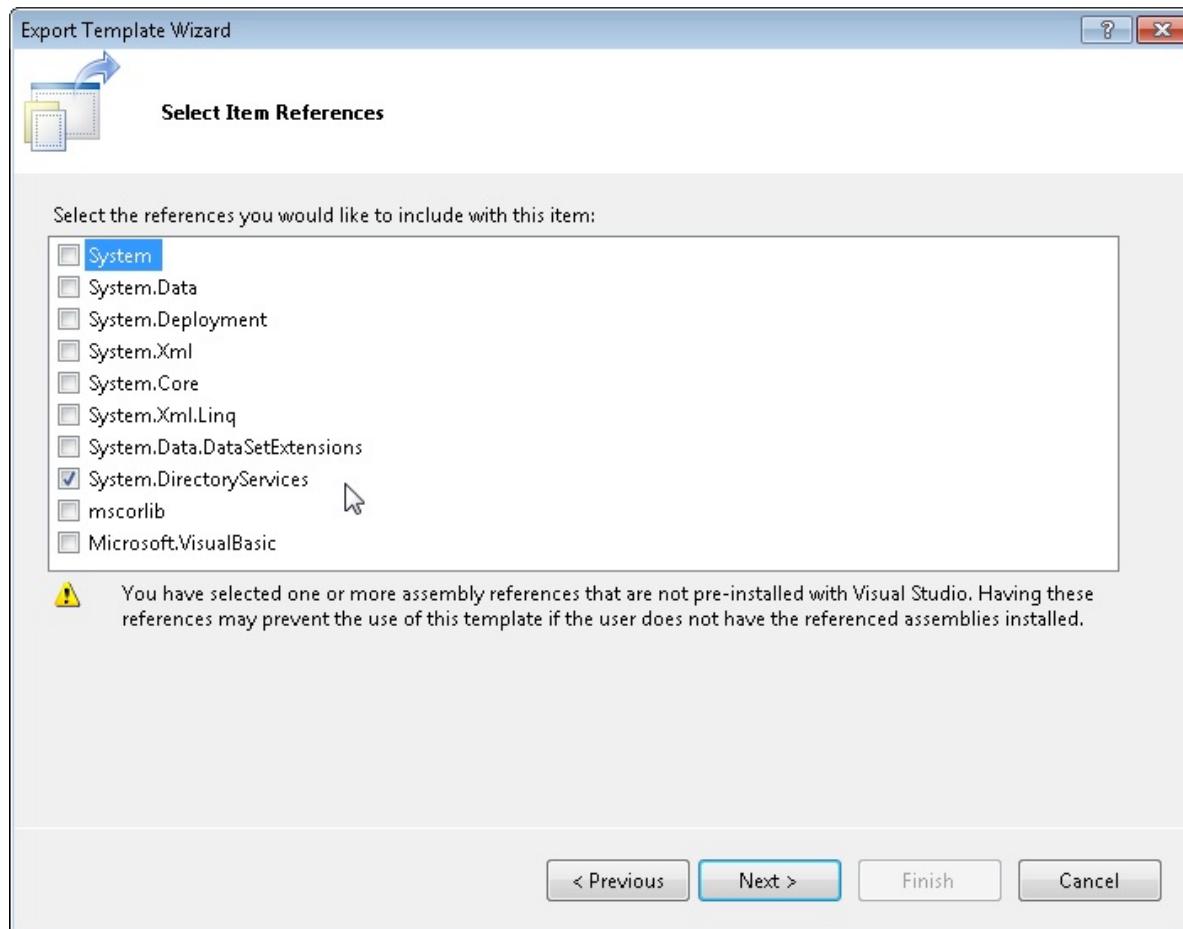
The wizard automatically selects any dependent files as needed based on your selection. Also, even though it looks like you can select more than one item here, you can only select a single item in this list.



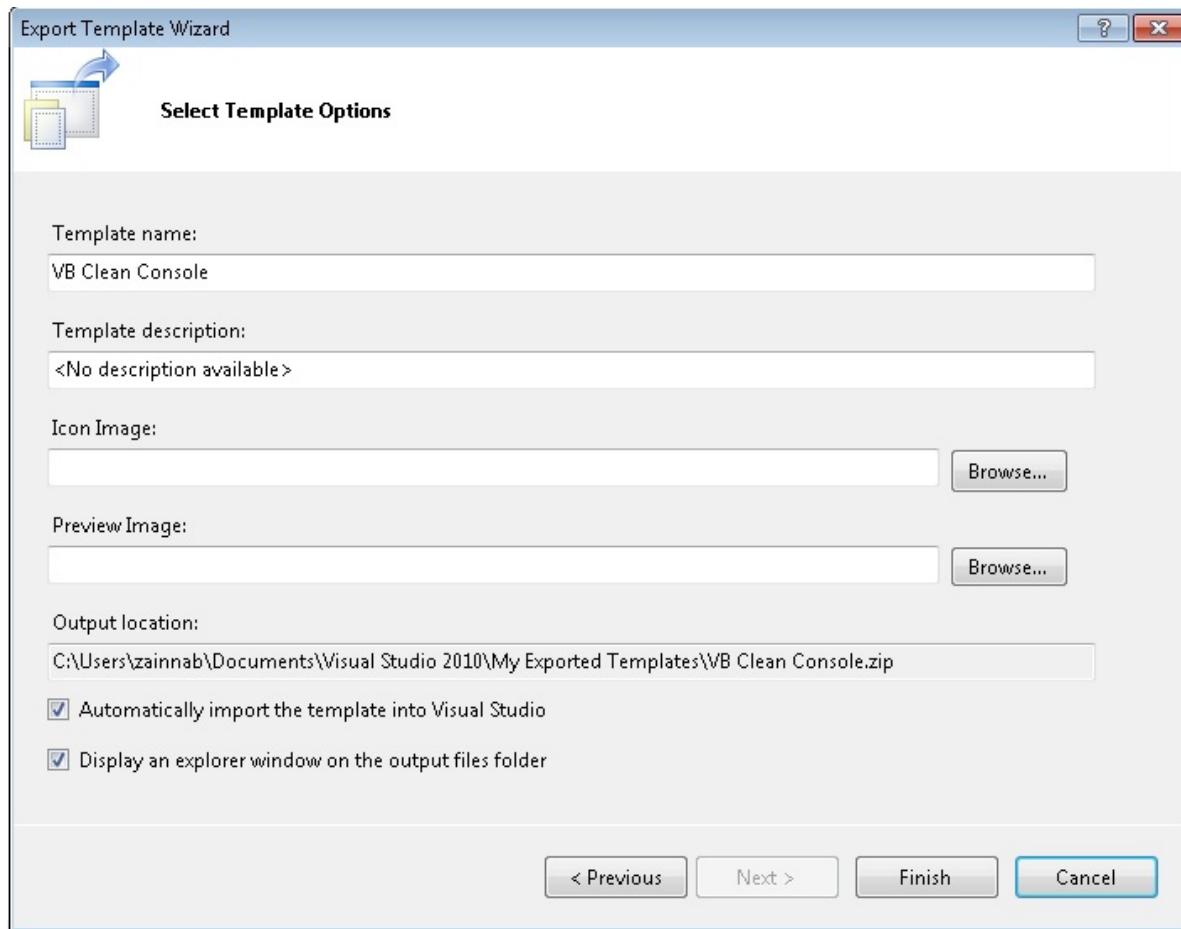
Click Next again. Now you can select any references that you want included with the item. If you have any Using or Imports statements, you need to pick the references here or the template will not work correctly.

#### NOTE

The wizard generates this list of assemblies from the assembly references in the current project. If the assembly you want to reference does not appear in the list, exit the wizard and add the reference to your project, and then run the wizard again.



Click Next. As you can see from the following illustration, you can add quite a bit of information.



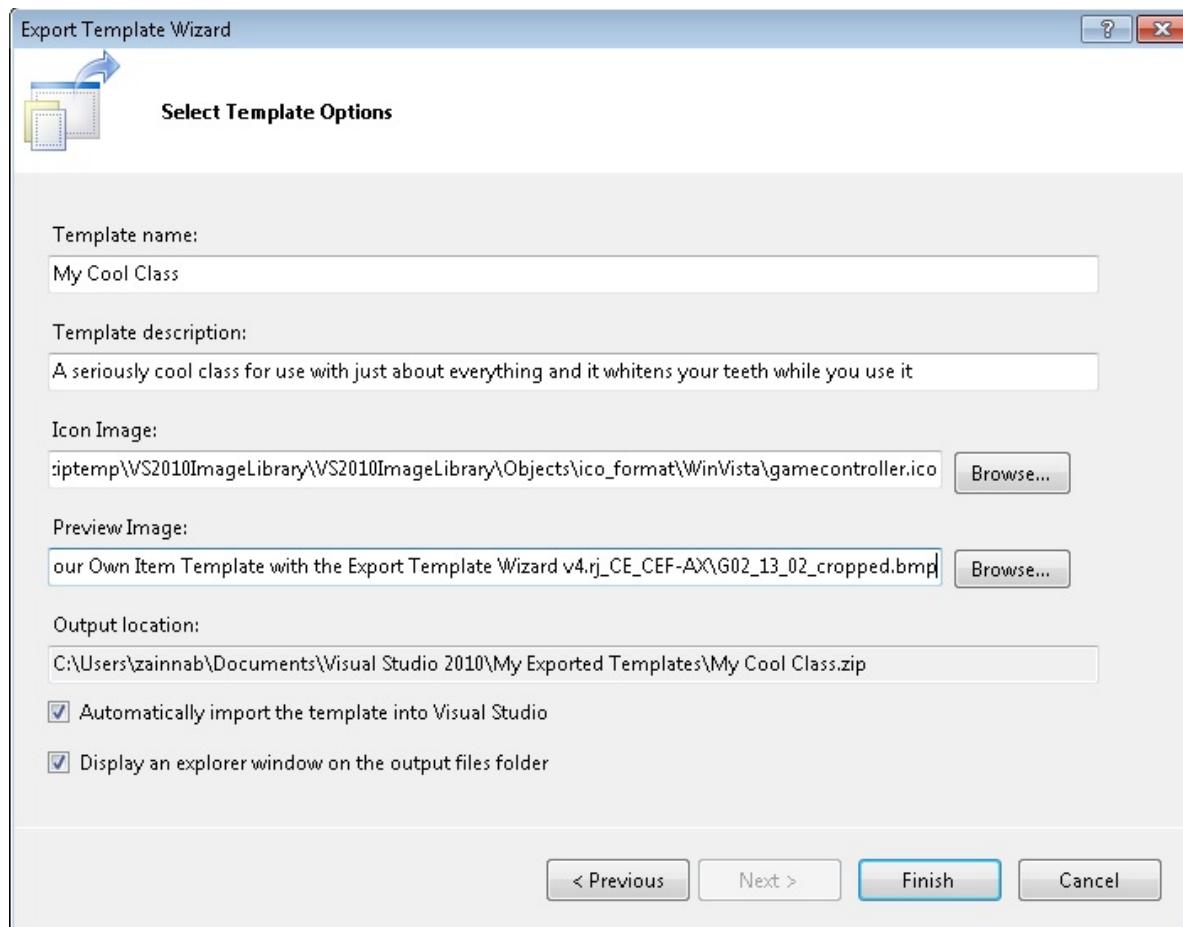
Here's a description of the information you can add:

- **Template Name** The friendly name for the template that Visual Studio displays in the list of templates. I suggest keeping this to around 50–60 characters. Don't get too verbose here.
- **Template Description** A short description that provides a little more detail about the template's purpose. In this text box, I want you to get very descriptive. This is your one and only chance to make it perfectly clear what this template should be used for, so don't skimp on detail.
- **Icon Image** A small image that represents the icon for the item. I suggest you just leave this blank.
- **Preview Image** A larger image that provides a preview of what the template looks like. As with the Icon Image field, I suggest you leave this blank.
- **Output Location** The location where the wizard stores exported items. This is the initial storage location of your templates. To be clear, they are not usable in Visual Studio when they are just created. To make them useable in Visual

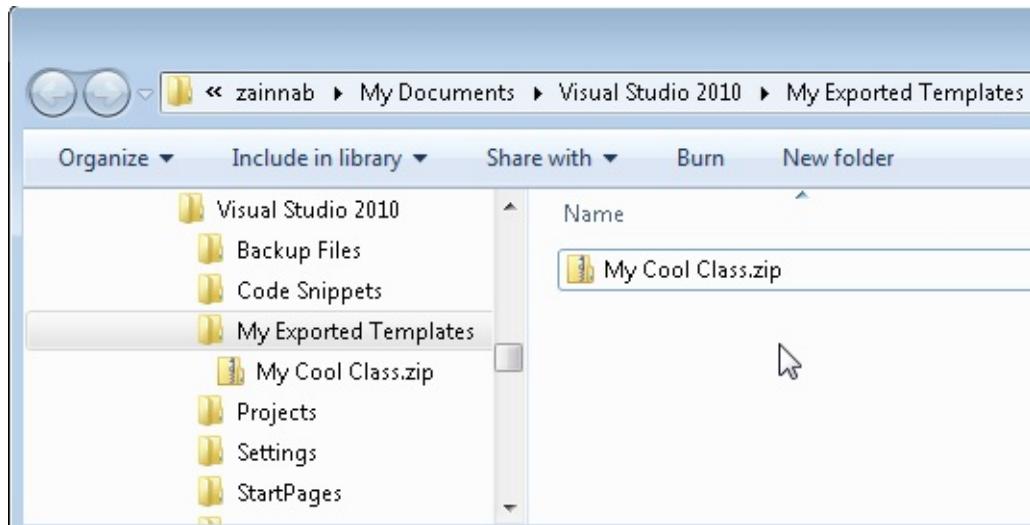
Studio, you need to check Automatically Import The Template Into Visual Studio. Leave this as-is unless you are storing your templates on a network share somewhere. If you do change this value, make sure you use the new location consistently when you create templates or you will wind up forgetting where you put them.

- **Automatically Import The Template Into Visual Studio** Lets you decide whether you want to import the template right away or want to do it manually later. This “import” is just a copy of the .zip file created in the appropriate location in My Documents\Visual Studio <version>\Templates\ItemTemplates. By doing this, the template immediately becomes usable in Visual Studio.
- **Display An Explorer Window On The Output Files Folder** Opens up the location where the template files are stored after they are created. This is useful when you want to see the .zip file that is created. It’s interesting the first few times you do it, but then it’s pretty much a waste of time. You will wind up turning off this option most of the time.

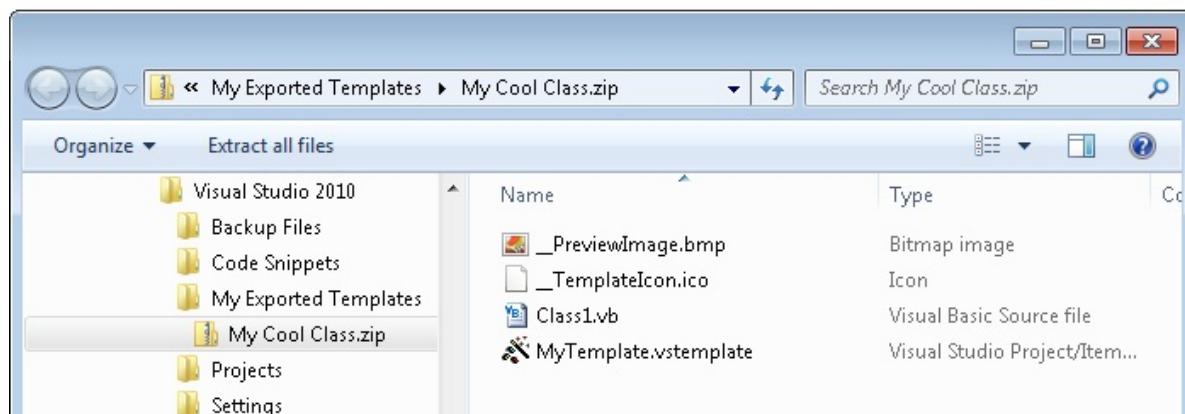
The following illustration shows the settings I used for this example.



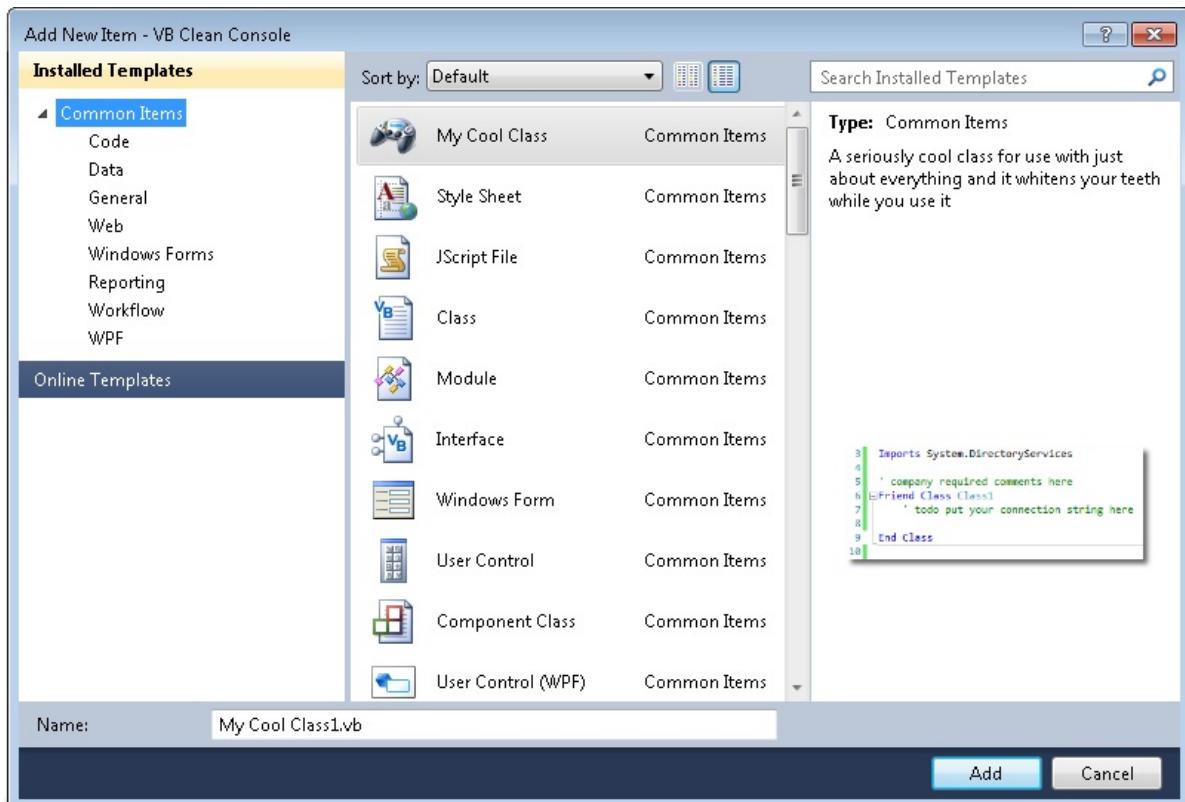
To complete the wizard, click Finish. The wizard closes and opens up the output file location, showing the .zip file that contains the exported templates.



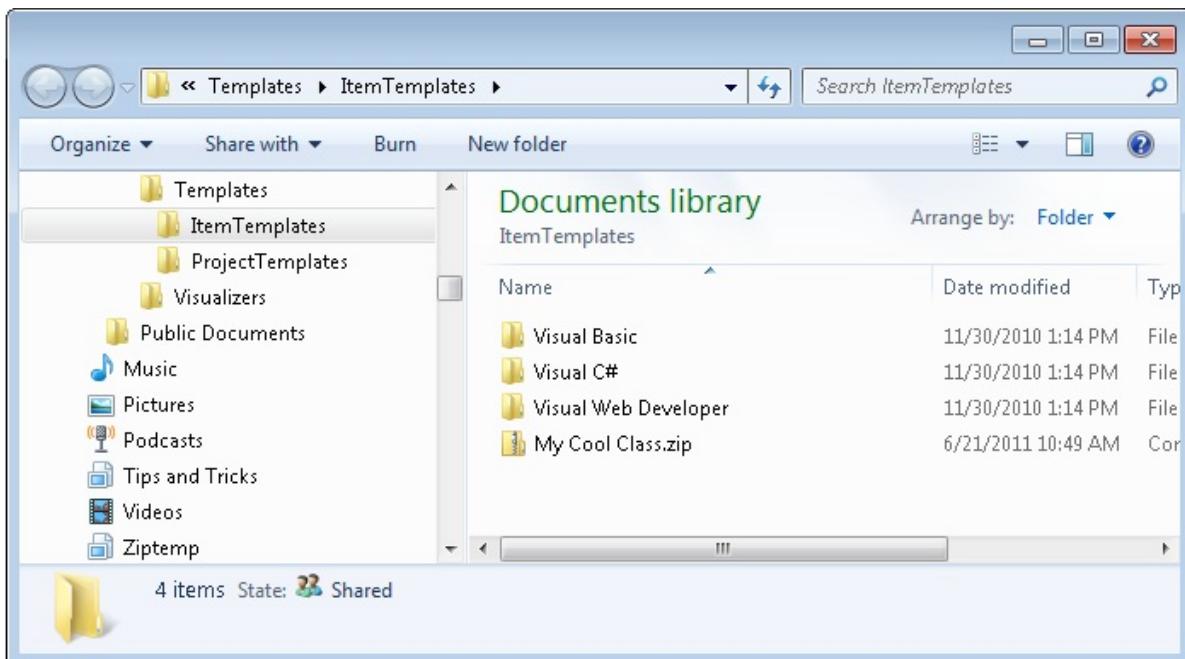
Although you aren't interested in the details right now, if you explore inside the .zip file, you can see the files that make up an item template.



Finally, test your new template. Create a new item (Ctrl+Shift+A), and you should see the new template. Notice the Icon Image next to the name of the item and the Preview Image (the Visual Studio 2010 logo in the lower-right) that is below the description. I feel that the names and descriptions are critical but really don't see a lot of value in the icons.



You can also see the template in the `My Documents\Visual Studio <version>\Templates\ItemTemplates\` folder.



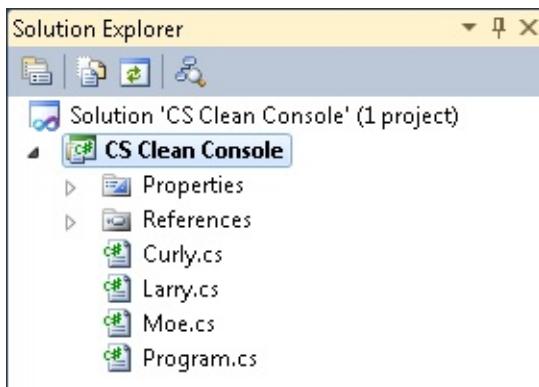
If you aren't happy with your new template, just delete the .zip file from this directory; it no longer shows up in the Add New Item dialog box.

## 02.14 Roll Your Own Project Template with the Export Template Wizard

<b>WINDOWS</b>	Alt,F, E
<b>MENU</b>	File   Export Template
<b>COMMAND</b>	File.ExportTemplate
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0004

Are you always adding the same extra files to projects when you create them? Ever wish you could have it all just “be there”? Well, you can when you become familiar with the Export Template Wizard.

First, set up an existing project template the way you want it. All changes (new files, code, interfaces, and so on) will be used in the template you create. In this simple example, I always want to include larry, curly, and moe C# class files with my console applications.

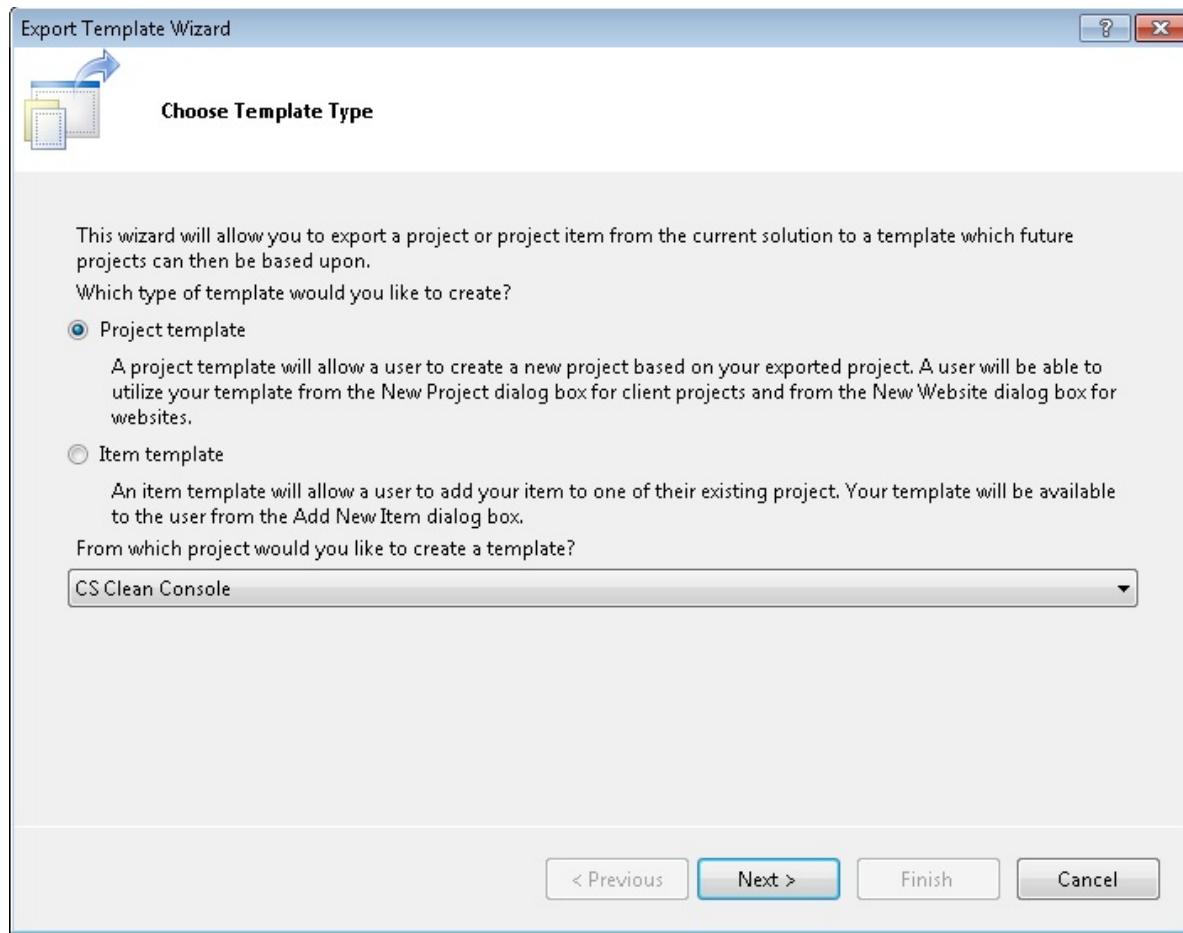


Now select File | Export Template from the menu bar.

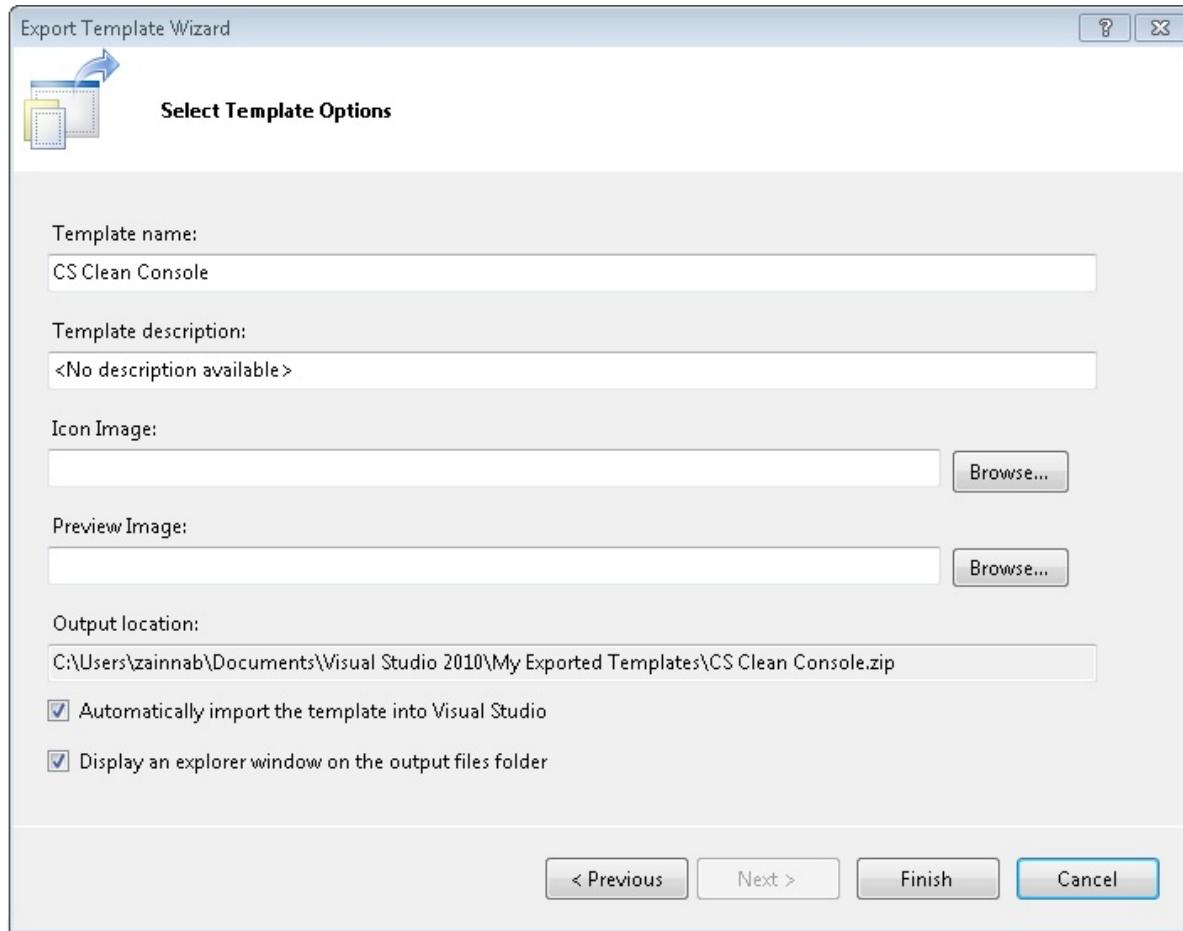
**NOTE**

Choosing Export Template prompts you to save any pending changes if you haven't already.

You'll see the Export Template Wizard. From this first screen, you can choose to make either a Project template or an Item template.



For this example, I selected Project Template and then clicked Next to continue to the Select Template Options screen, shown in the following illustration.



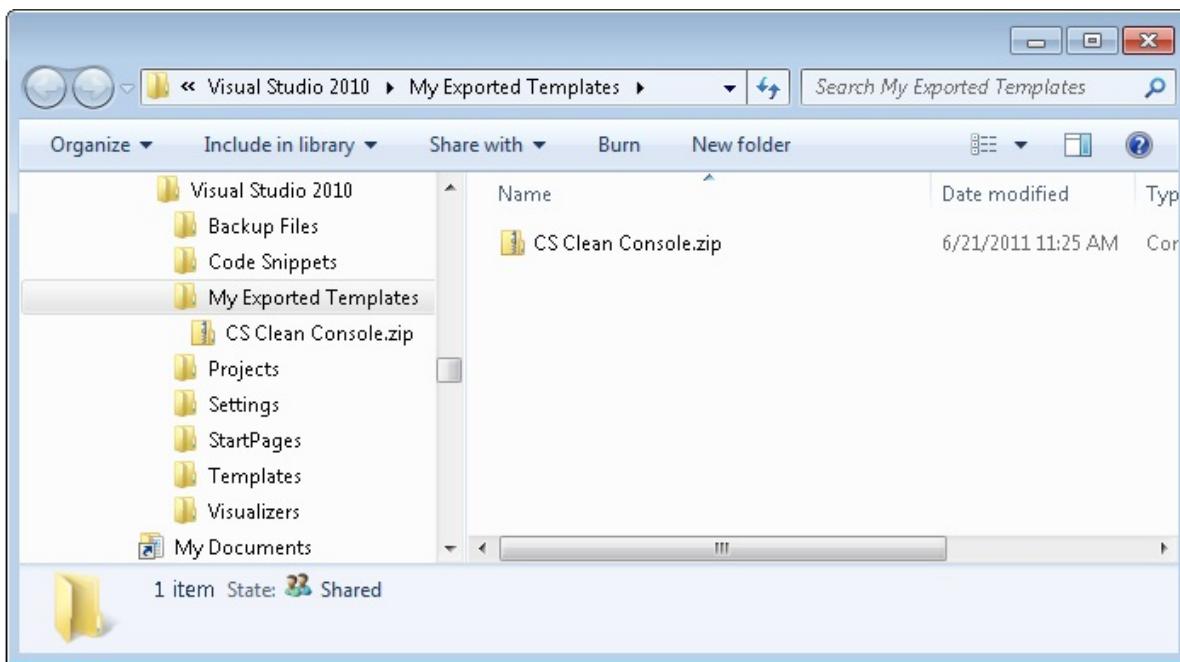
The following list provides a brief description of each option:

- **Template Name** The friendly name for the template that Visual Studio displays in the list of templates. I suggest keeping this to around 50–60 characters. Don't get too verbose here.
- **Template Description** A more complete explanation of how this template is intended to be used. This is your one and only opportunity for you to be very clear on the proper usage for this template.
- **Icon Image and Preview Image** The images used with the template name (icon image) and just below the description (preview image). I suggest you don't bother setting these because they don't have much use, in my opinion.
- **Output Location** The location where the wizard saves the .zip file that it creates. The default value is usually what you will stick with unless you have, say, a network share where you want your templates to stored.
- **Automatically Import The Template Into Visual Studio** Controls whether the wizard puts a copy of the new template to your templates directory: My

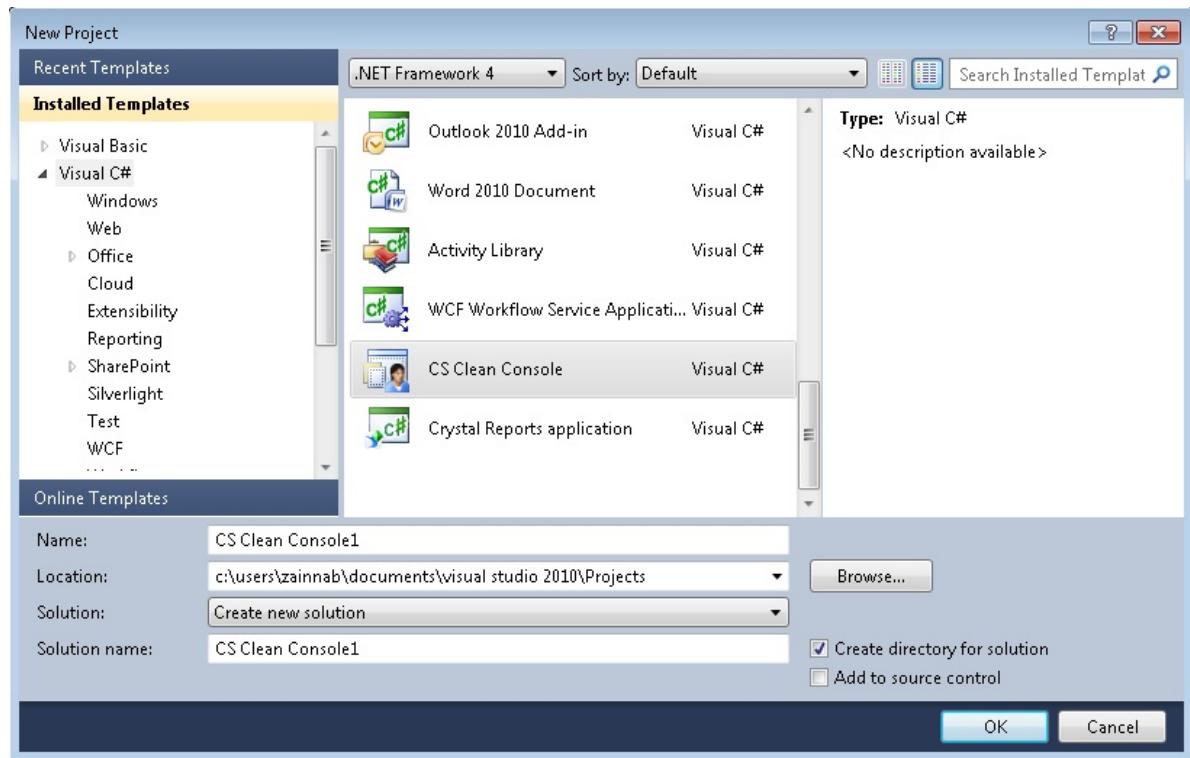
Documents\Visual Studio <version>\Templates\ProjectTemplates. If you want the template to be available the next time you create a new project, select this option. Most of the time you should leave this option selected.

- **Display An Explorer Window On The Output Files Folder** Controls what happens when you complete the wizard. When selected, it displays the folder containing the .zip file that the wizard saves. After the first few times you create templates, this option can get tiresome, so I usually turn it off. I suggest you leave it on the first few times you create templates to see the template that is created.

After filling out the wizard and clicking Finish, the wizard closes and opens up my Exported Templates (output) folder, where I can see the new .zip file containing the template files.



Now, when I create a new application, the new template appears, visible in the New Project dialog box.



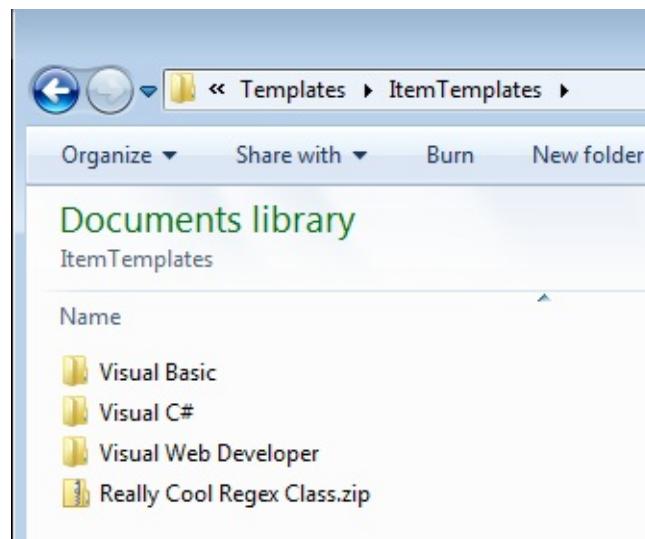
Obviously, this is a simple example; you can do a lot more with templates, and I suggest you visit the “Export Template Wizard” documentation, at <http://msdn.microsoft.com/en-us/library/ms185318.aspx>, for more detailed information about how to make good use of this feature.

## 02.15 Organizing Your Custom Item Templates

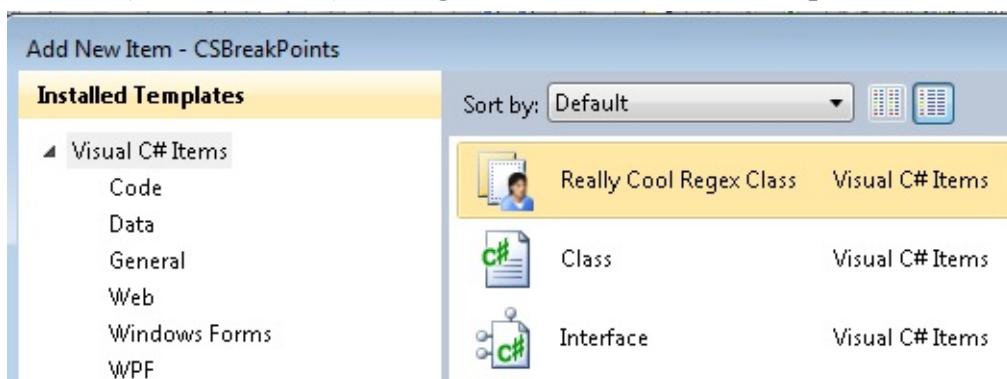
VERSIONS	2005, 2008, 2010
CODE	vstipProj0020

In vstipProj0013 ([02.14 Roll Your Own Project Template with the Export Template Wizard](#), page 57), I showed you how to create custom item templates but didn't show you how to organize them.

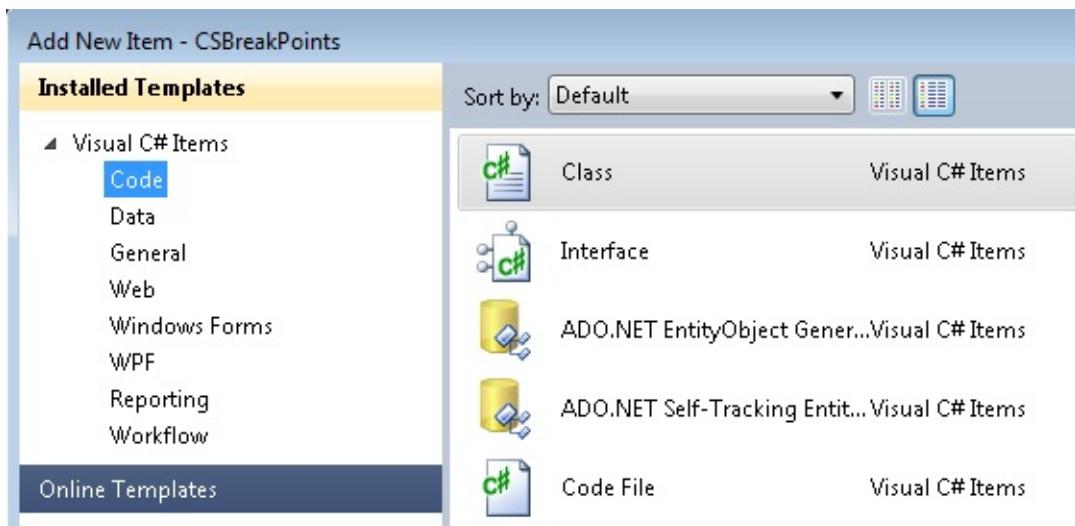
Fortunately, organizing them is pretty easy. After you have created your template(s), navigate to the folder My Documents\Visual Studio <version>\Templates\ItemTemplates. For example, on my machine, the path is My Documents\Visual Studio 2010\Templates\ItemTemplates.



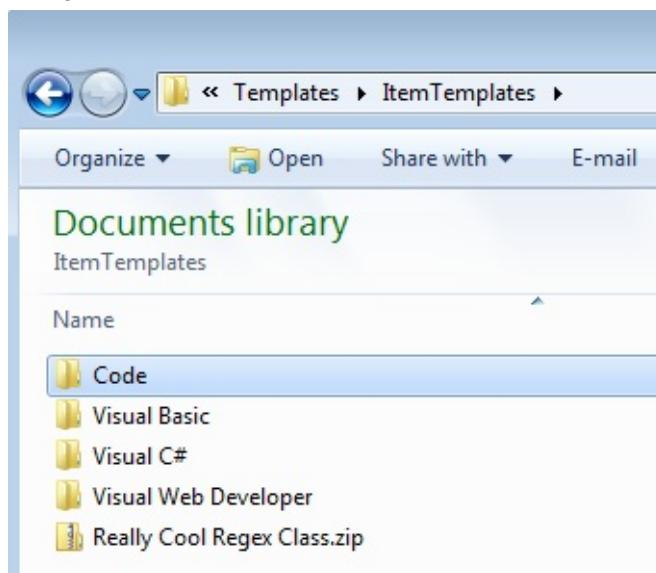
As you can see in the preceding illustration, I have a custom item—a class called “Really Cool Regex Class.” Unfortunately, when I want to use it and I bring up the Add New Item (Ctrl+Shift+A) dialog box, that class shows up in the root list.



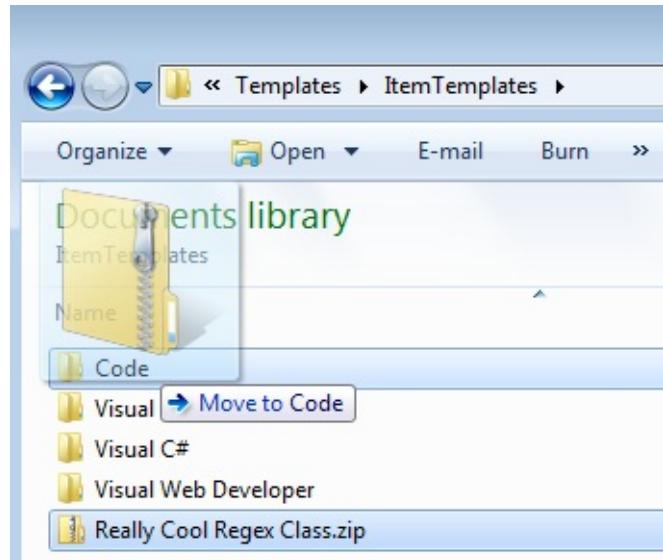
I want it to show up in the Code area, but it doesn't.



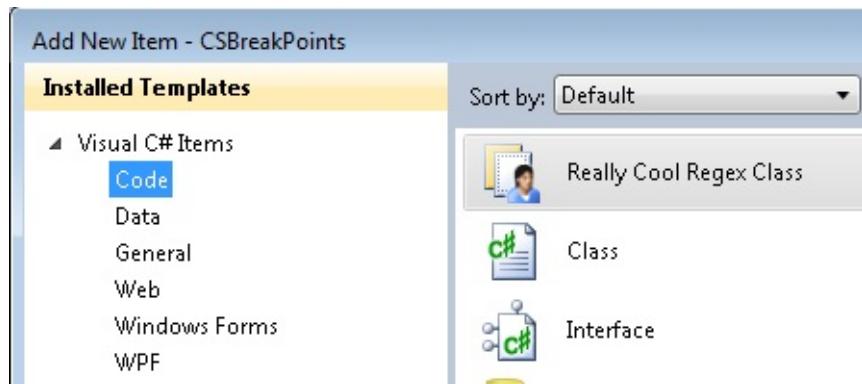
To get the custom template to show up in the Code area, you need to go back to the ItemTemplates directory and create a new folder named Code.



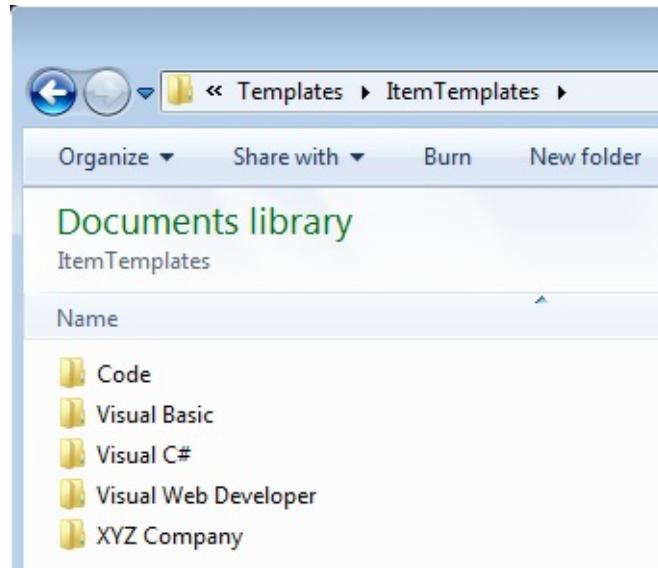
Then move the custom template into the Code folder.



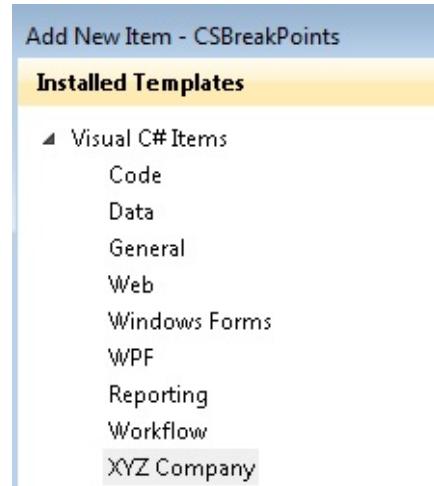
Now whenever I press Ctrl+Shift+A to add a new item, my custom template appears in the Code section.



In addition to working with existing folder names, you can create custom names as well. If, for example, you wanted an XYZ Company folder for your templates you would just create one and put your templates in there:



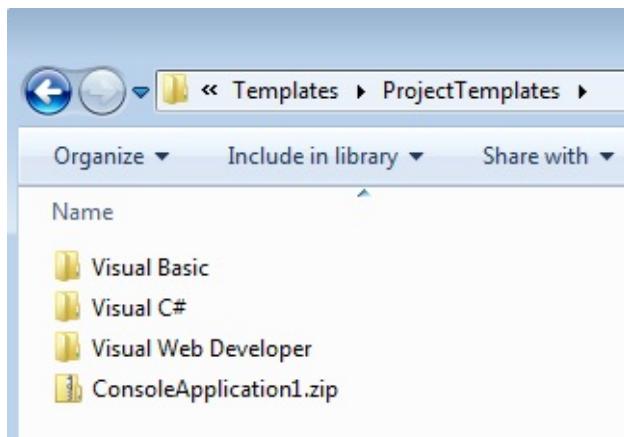
Now, when you go to add a new item, you will see your new folder in the dialog box:



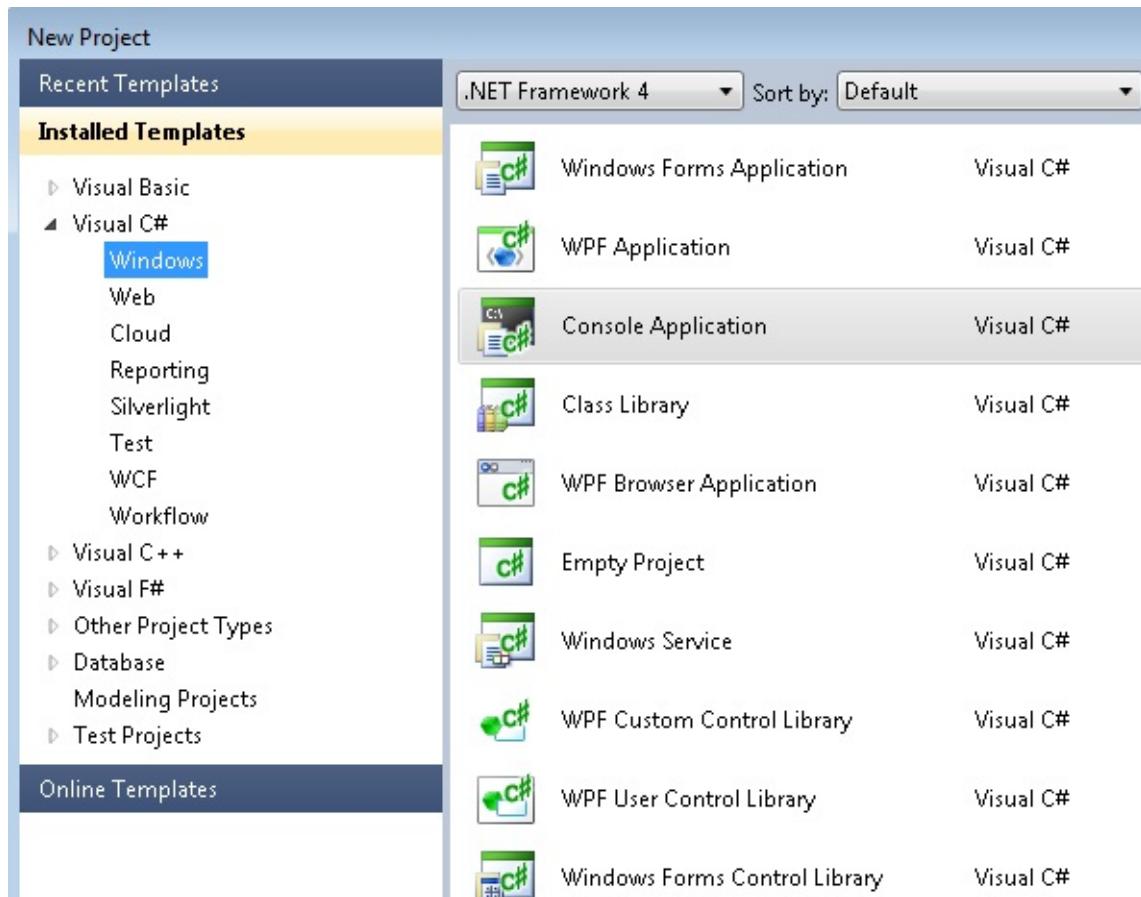
## 02.16 Organizing Your Custom Project Templates

VERSIONS	2005, 2008, 2010
CODE	vstipProj0019

In vstipProj0004 ([02.14 Roll Your Own Project Template with the Export Template Wizard](#), page 64), we discussed how to create custom project templates, but it doesn't show you how to organize them. Fortunately, that's pretty easy. After you have created one or more custom project templates, browse to My Documents\Visual Studio <version>\Templates\ProjectTemplates. For example, on my machine, the full path is My Documents\Visual Studio 2010\Templates\ProjectTemplates.

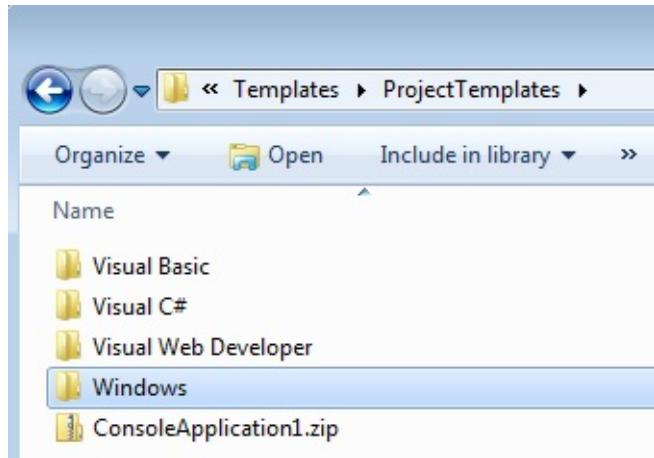


As the preceding illustration shows, I created a custom Console application project type for this example with the name ConsoleApplication1. By default, custom project templates don't show up in the project subfolders.

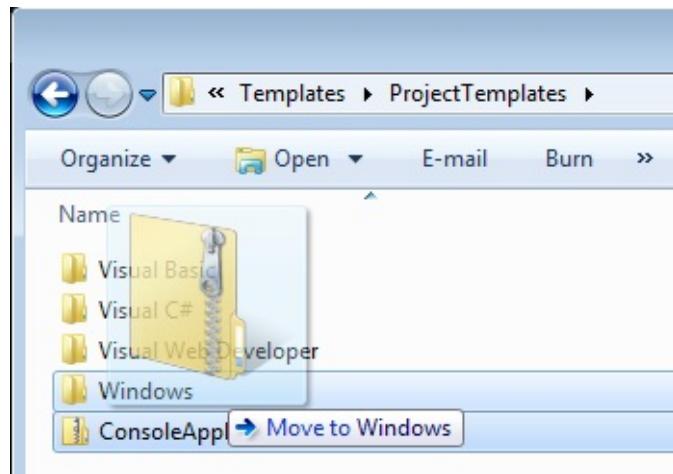


To get the custom templates to appear, the trick is to create a new folder in that directory with a name that matches where you want the template(s) to show up. You place custom templates in this new folder—and then they show up in the appropriate areas.

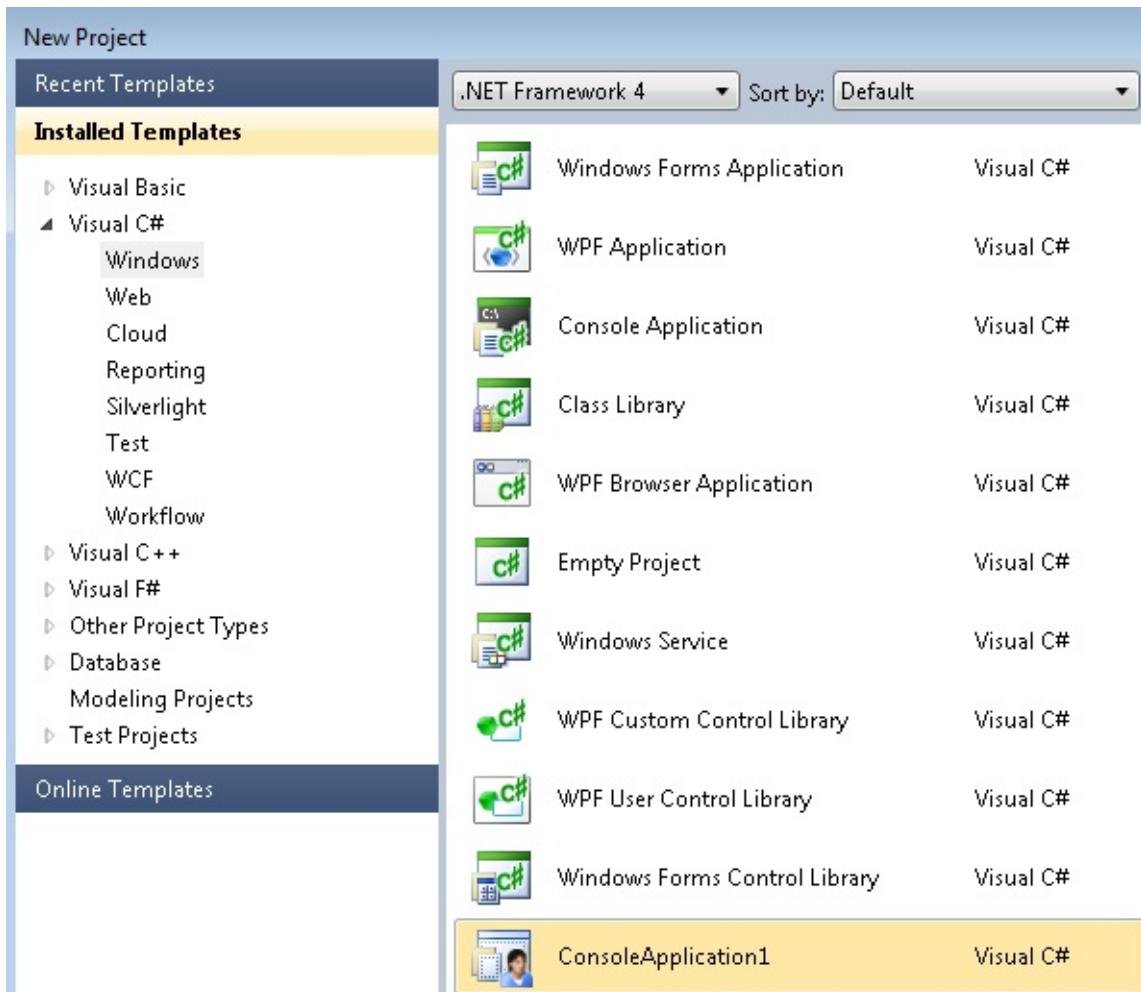
To do this, return to the `My Documents\Visual Studio <version>\Templates\ProjectTemplates` directory and create a new folder called Windows—to match the Windows area in the New Project dialog box, which is where we want the new Console template to appear.



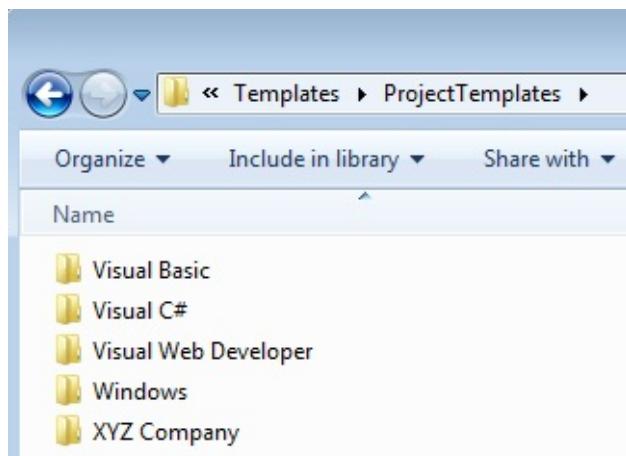
Move your template into the new folder. For this example, I moved ConsoleApplication1.zip into the Windows folder.



The next time you open up the New Project dialog box (Ctrl+Shift+N), it shows the project template in the proper area.



In addition to matching the existing folder names, you can create new ones. If you want a custom area for your company templates, for example, you would just create a folder with your company name and put at least one template in the folder:



The next time you bring up the New Project dialog box, it will show your new subfolder in the list:



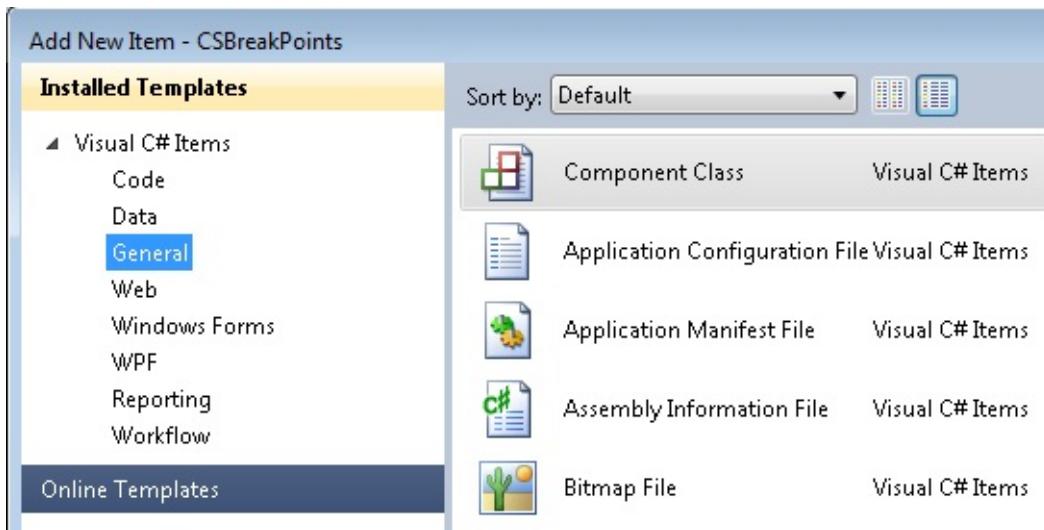
## 02.17 Reorganize the Default Item Templates

VERSIONS	2005, 2008, 2010
CODE	vstipProj0021

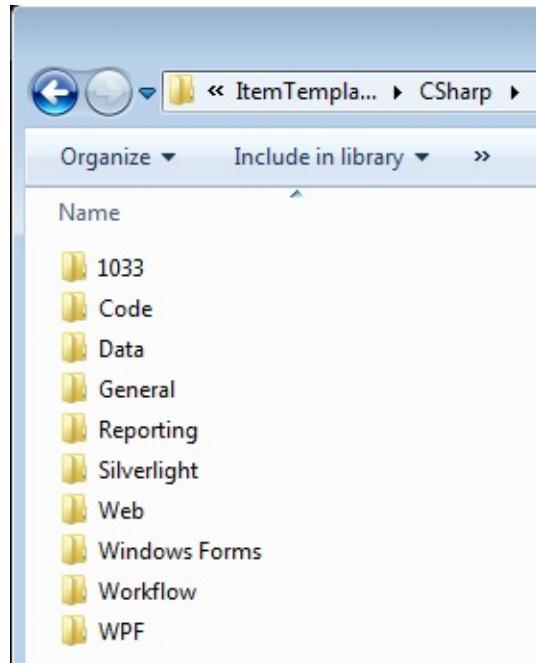
### WARNING

The procedures in this tip could cause your templates to disappear if you don't follow the instructions carefully. So do this at your own risk. You might want to back up your ItemTemplates folder just to be safe.

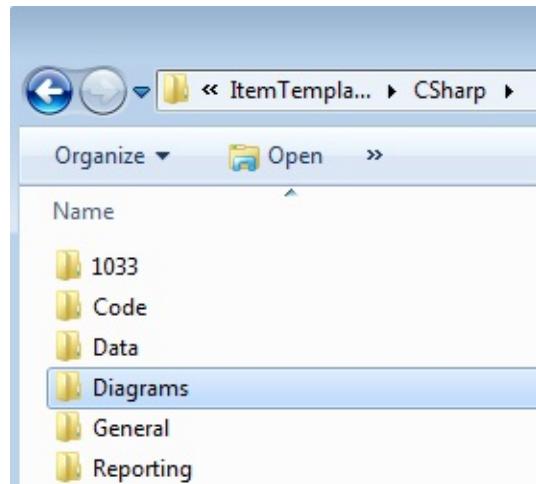
You've probably noticed that General section of the New Item dialog box contains a large number of items. If you want to organize those a bit more, this tip shows you how to create custom areas in which you can store the default item templates that ship with Visual Studio. This example creates a Diagram area for the diagram items that—by default—appear in the General section.



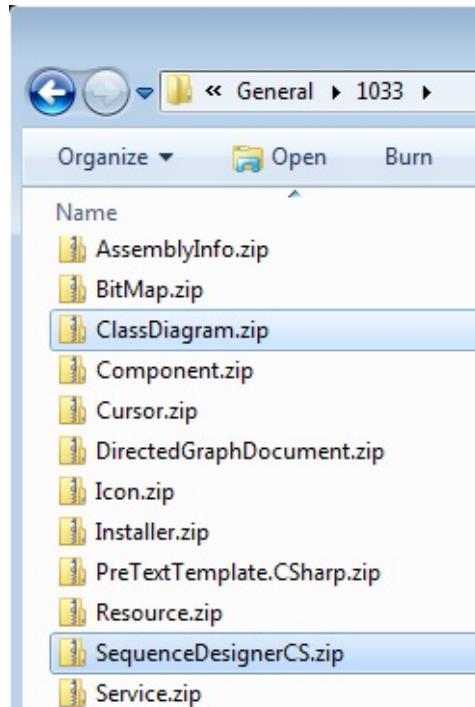
To get started, find where Visual Studio stores item templates on your machine. Typically, this is in `C:\Program Files\Microsoft Visual Studio <version>\Common7\IDE\ItemTemplates\<language>`. You might have to drill down into the file structure, depending on what items you're looking for, and the path might be slightly different on your machine, based on your Visual Studio version. In this case, the actual full path on my machine is `C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\ItemTemplates\CSharp`.



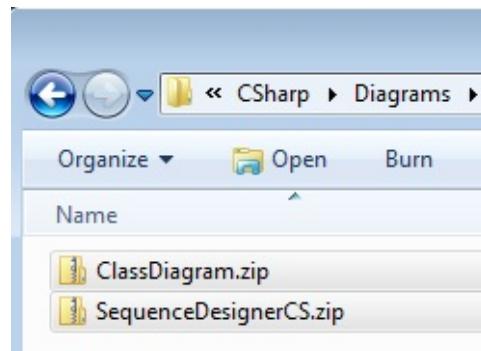
When you've found the template location for your language, create a new folder. For this example, I created a folder called Diagrams.



Now go into the General\1033 folder, and locate the diagram .zip files you want, as shown in the following illustration.



Now carefully move them to the new Diagrams folder.



Close all instances of Visual Studio, and then run the following command from the Visual Studio command prompt (must be run with administrative privileges):

`devenv.com /installvstemplates`

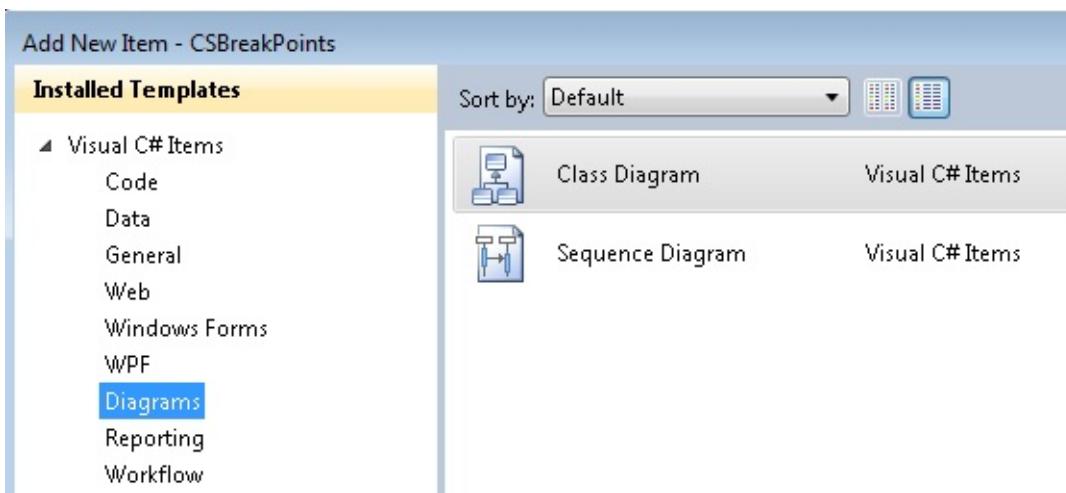
A screenshot of an Administrator: Visual Studio Command Prompt window. The title bar says 'Administrator: Visual Studio Command Prompt (2010)'. The command line shows 'c:\>devenv.com /installvstemplates'.

#### WARNING

Let this process complete without interfering. It is extremely important that you let the process finish. The `devenv.com` command runs without any user interface. You know it is done when another cursor shows up:

```
c:\>Administrator: Visual Studio Command Prompt (2010)
c:\>devenv.com /installvstemplates
c:\>
```

When the process completes, open up Visual Studio, and then open any project. Bring up the Add New Item dialog box (Ctrl+Shift+A). Notice your brand new Diagrams area, with the templates you moved there inside it.



## 02.18 Reorganize the Default Project Templates

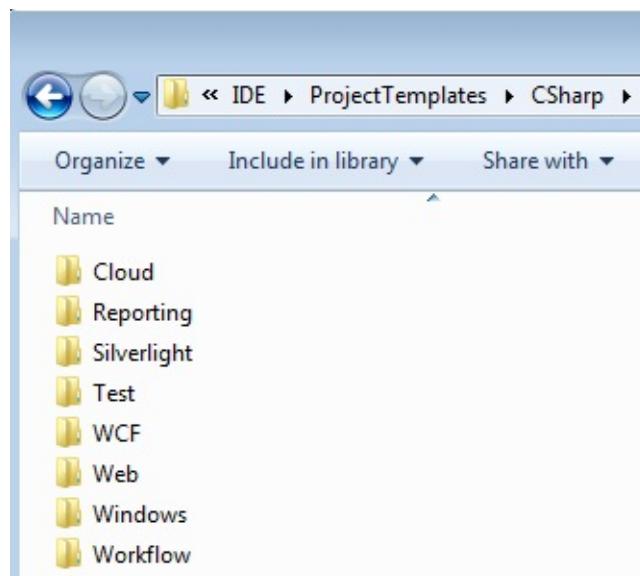
VERSIONS	2005, 2008, 2010
CODE	vstipProj0018

### WARNING

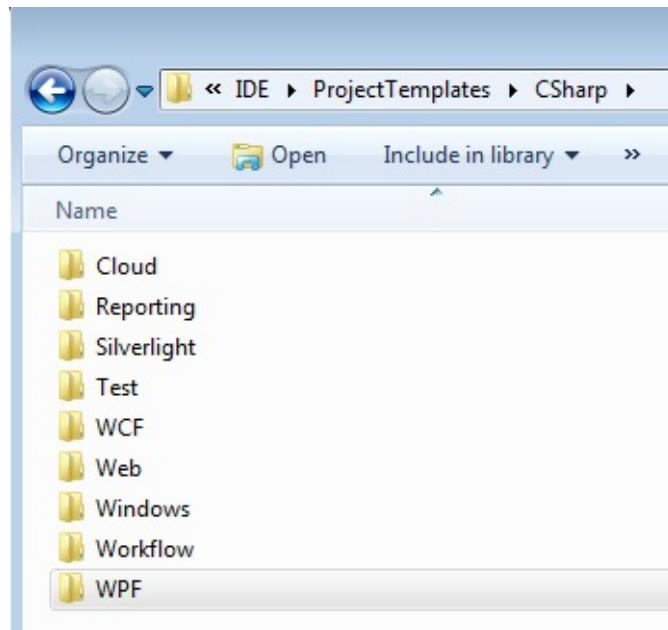
The procedures in this tip could cause some of your templates to disappear unless you follow the instructions carefully. So do this at your own risk. You might want to back up your ProjectTemplates folder just to be safe.

Maybe it's just me, but I get really annoyed that, for example, my WCF project templates are in a WCF section when I go to create a new project—but my WPF projects are under “Windows.” That just doesn’t seem intuitive. So this example shows you how you can create custom areas for the default project templates that ship with Visual Studio. In this example, you create a WPF area for your WPF project templates.

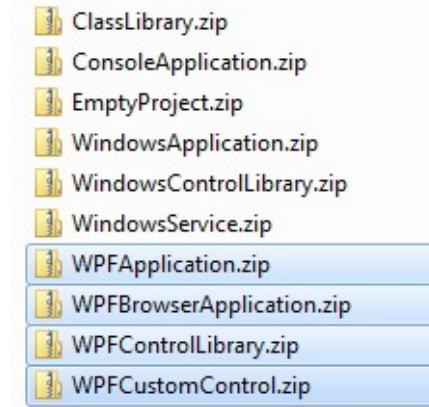
You need to find the location where your version of Visual Studio stores project templates. Typically, that’s in C:\Program Files\Microsoft Visual Studio <version>\Common7\IDE\ProjectTemplates\<language>. You might need to explore the file system to find the templates, and your path might be slightly different, based on which Visual Studio version you’re running. For example, the actual full path on my machine for C# templates is C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\ProjectTemplates\CSharp.



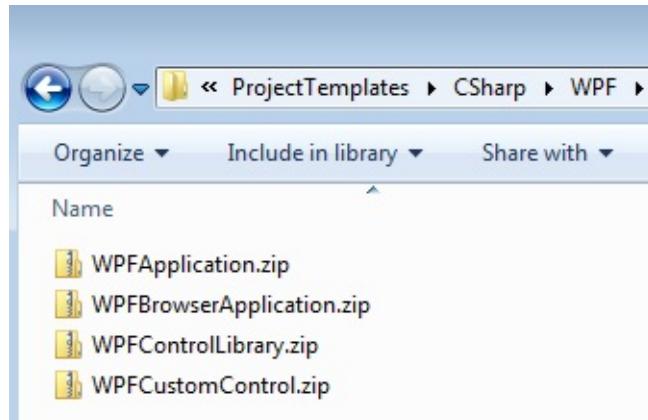
Now create a new folder, and name it “WPF.”



These next steps are potentially dangerous, so be careful. Navigate to the Windows folder (actually Windows\1033\), and locate the WPF templates.

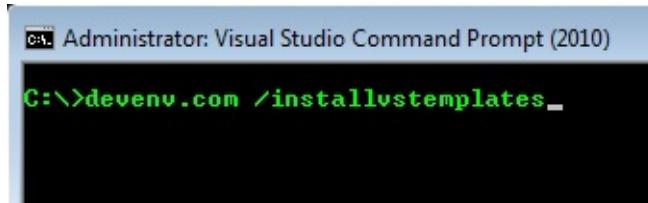


Move these .zip files into the new folder you just created.



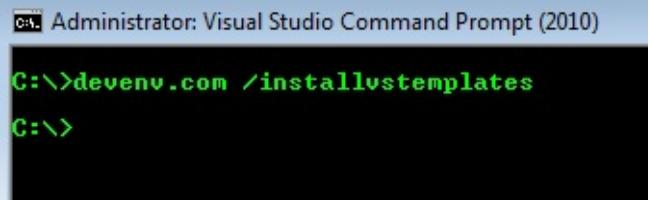
Close all instances of Visual Studio, and then run the following command from the Visual Studio command prompt, which you can find on your Start menu:

```
devenv.com /installvstemplates
```

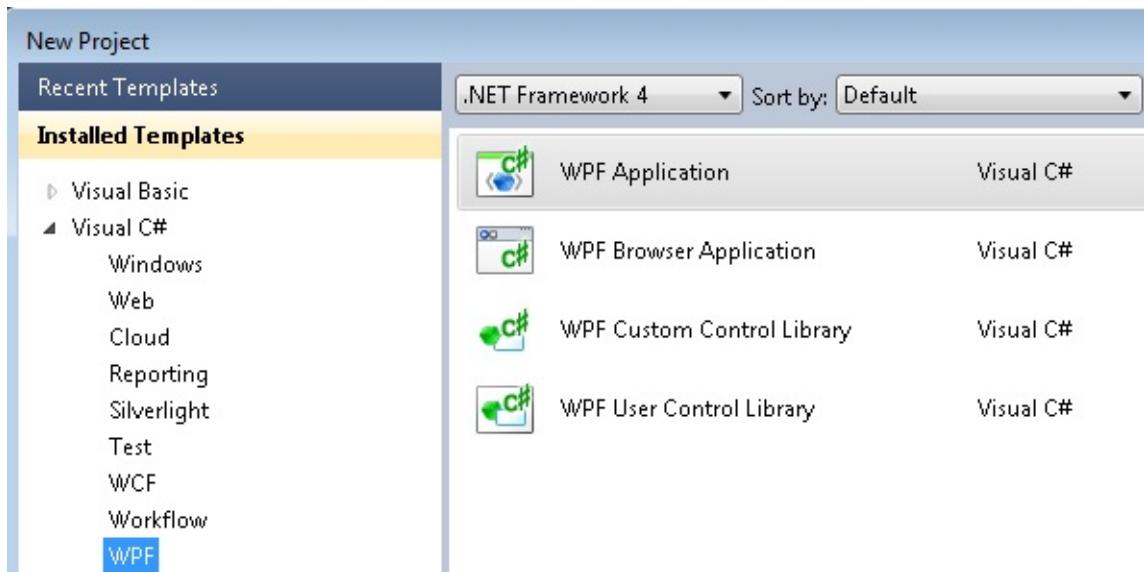


#### WARNING

It is very important that you let this process finish without interruption. The devenv.com command runs without any user interface. You know it is done when the command prompt shows up again:



When the process completes, open up Visual Studio, and open the New Project dialog box (Ctrl+Shift+N) to see your brand-new WPF area containing the WPF templates, as shown in the following illustration.



#### NOTE

While researching how to do this, I experimented with copying the templates instead of moving them. However, Visual Studio apparently detects duplicate template names and doesn't allow you to have multiple copies in different locations. So I wound up with an empty WPF section; the templates stayed in their original Windows section. I suspect this is dependent on load order—and my tests indicate Visual Studio loads the known default directories first, so having a folder called “Abacus” to beat the sort order doesn’t work.

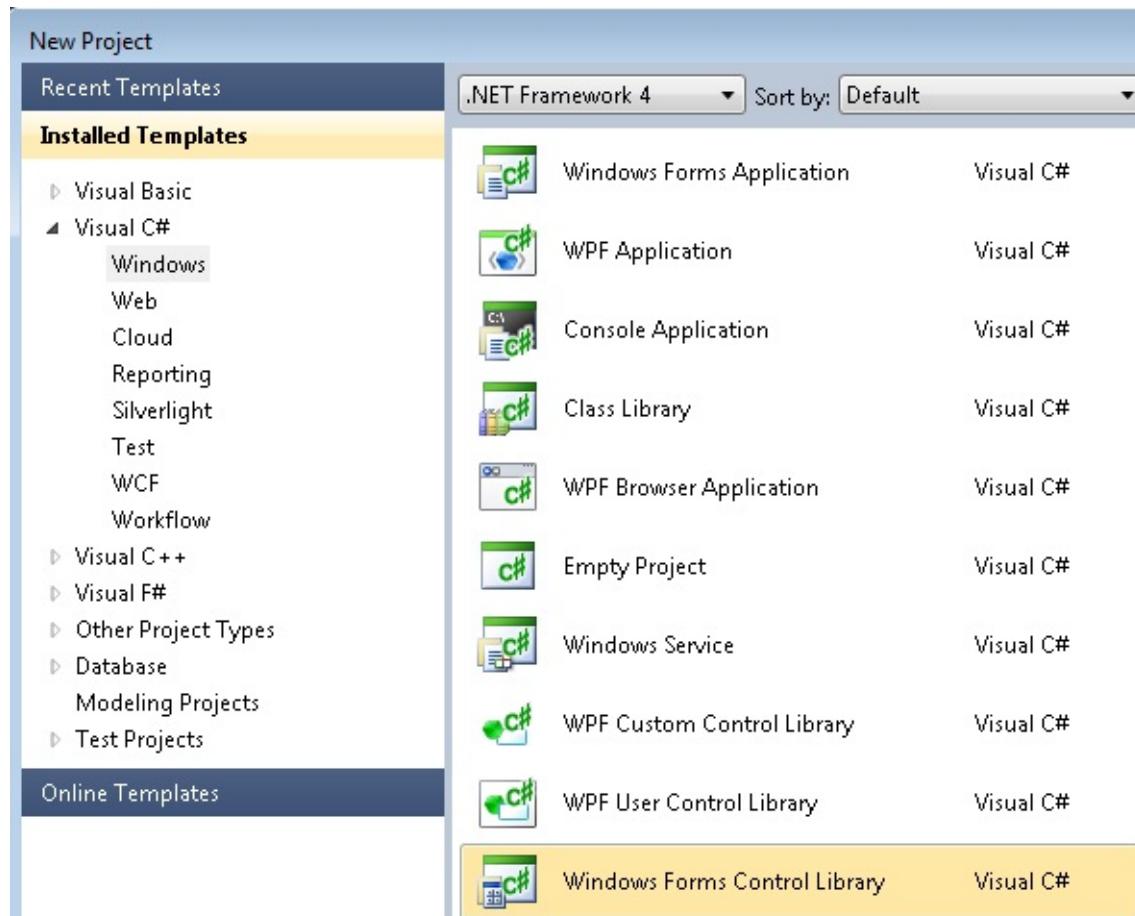
## 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes

VERSIONS	2008, 2010
CODE	vstipProj0017

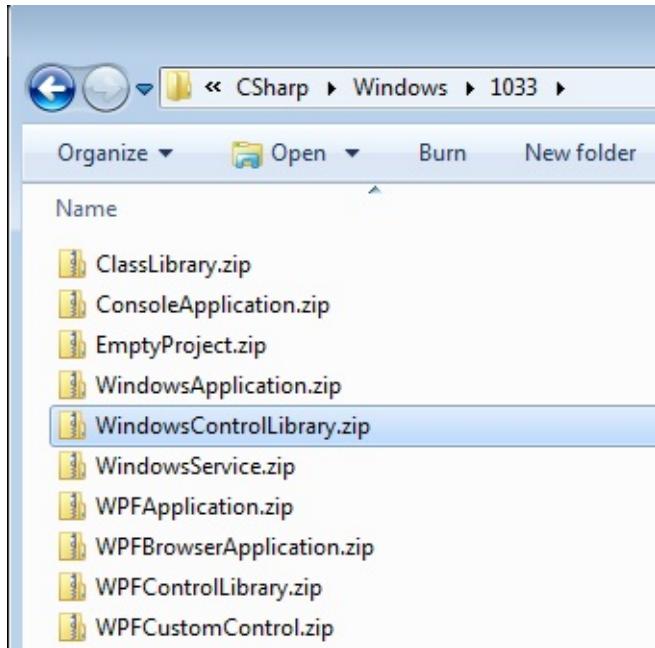
### WARNING

Manipulating templates as discussed here can cause serious problems if you don't know what you are doing. Use this information at your own risk. You should consider backing up your ProjectTemplates or ItemTemplates folders.

With all the great changes to the New Project and New Item dialog boxes, you might be perfectly happy with the list of things that Visual Studio presents by default. But for argument's sake, suppose you want to get rid of some of the entries. This example removes the C# Windows Forms Control Library from the New Project dialog box (Ctrl+Shift+N), but you can follow the same steps to remove or change items in the New Item dialog box.



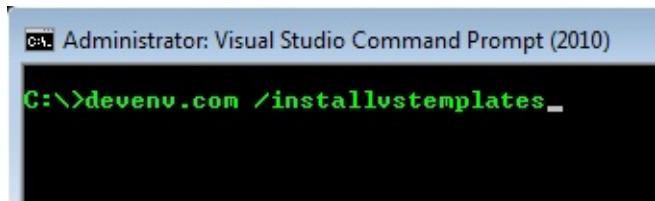
Navigate to the location where your version of Visual Studio stores templates for your selected language. Typically, that's C:\Program Files\Microsoft Visual Studio <version>\Common7\IDE\<Project or Item>Templates\<language>\<project category>. You might have to explore a little; your path might be slightly different, based on which Visual Studio version you're running. On my machine, the path to the Windows Forms Control Library .zip file is C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\ProjectTemplates\CSharp\Windows.



You don't want to simply delete the file—you might need it in the future. Instead, just move the .zip file to another directory. That way, you can always retrieve it from that location if you need it again.

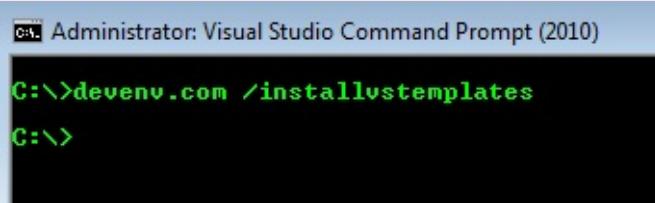
Close all instances of Visual Studio, and then run the following command from the Visual Studio command prompt, which you can find on your Start menu:

```
devenv.com /installvstemplates
```



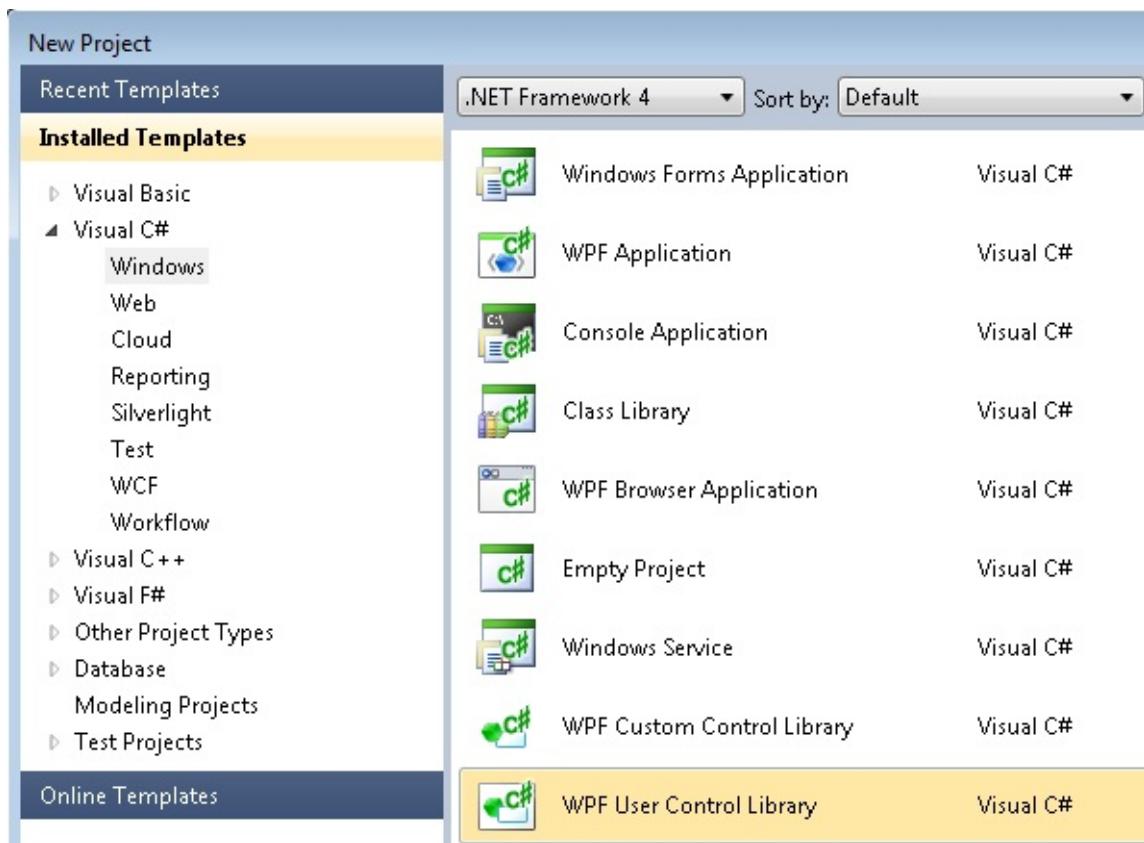
#### WARNING

It is very important that you let this process finish without interruption. The devenv.com command runs without any user interface. You know it is done when the cursor shows up again:



After the process completes, start Visual Studio again, and then create a new

project (Ctrl+Shift+N). You should see that the moved template is no longer in the list.



# Chapter 3. Getting to Know the Environment

*“A mobile robot has to devote a tremendous amount of processing time simply to avoid obstacles in the environment. Human beings do, too, but they’re never aware of it—until the lights go out. Then they learn painfully just how much processing is really required.”*

— Michael Crichton “*Prey*”

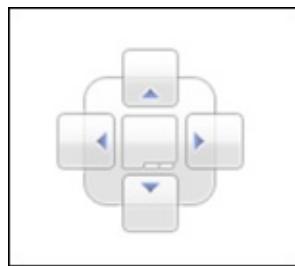
Too often we take our environment for granted—the little things that we see every day and, yet, fail to notice. This section is meant to awaken you to the possibilities in your Visual Studio environment.

Most notably, the purpose is to highlight how best to work with your window layouts, how to use the toolbox to your advantage, and how to work with commands properly, among other things. Take time to really explore the Visual Studio environment, and you can unlock the secrets to navigating that same environment successfully.

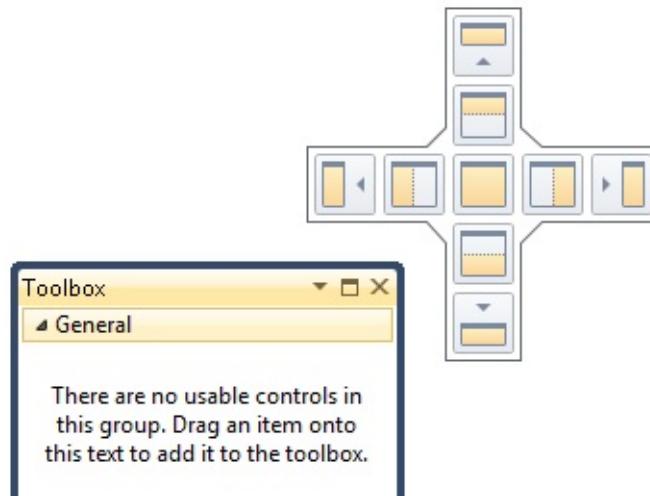
## 03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond

<b>DEFAULT</b>	[no shortcut]
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	[no shortcut]
<b>VISUAL C++ 2</b>	Alt+F6 (dock)
<b>VISUAL C++ 6</b>	[no shortcut]
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt,W, F (Float); Alt,W, K (Dock); Alt,W, T (Dock as Tabbed Document)
<b>MENU</b>	Window   Float; Window   Dock; Window   Dock as Tabbed Document
<b>COMMAND</b>	Window.Float; Window.Dock; Window.DockasTabbedDocument
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0008

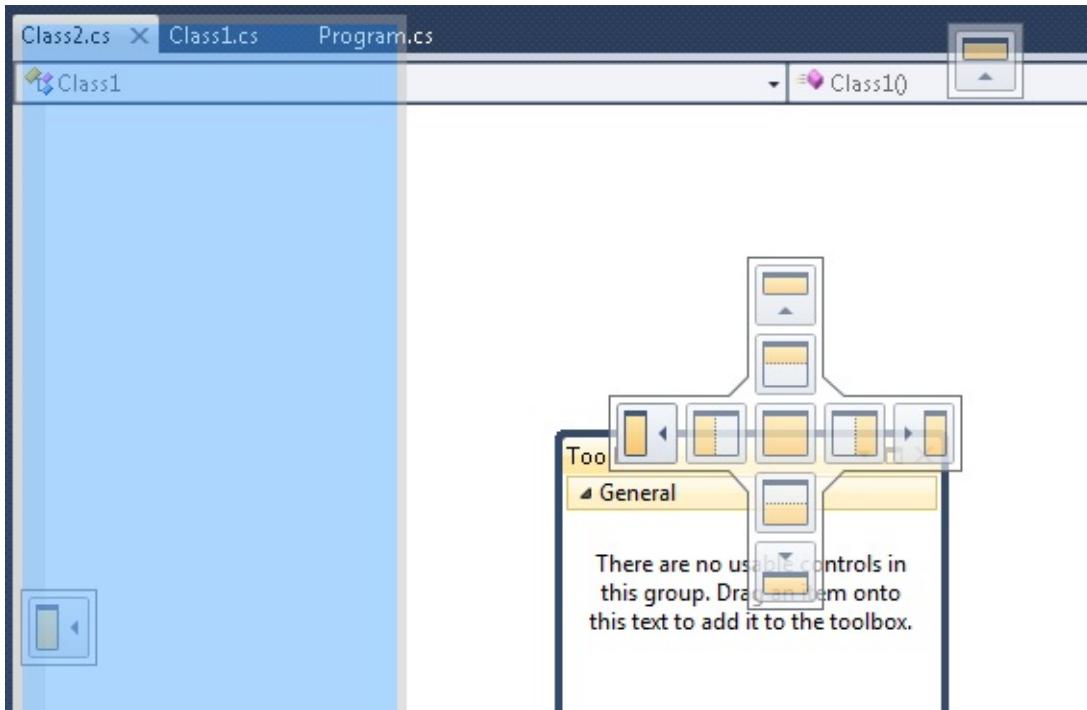
Docking and undocking windows in the IDE has always been interesting. In Visual Studio, we have a tool called the Guide Diamond that is used to assist our efforts. The following illustration shows the Guide Diamond in Visual Studio 2008.



Unfortunately, this doesn't really provide good visual cues to help determine the final position of a window. Visual Studio 2010 provides a new and improved Guide Diamond that makes docking much easier, as shown in the following illustration.



Now you can clearly see how your docked window will look based on the image in the diamond. Just drag the title bar of your window over one of the previews in the Guide Diamond.

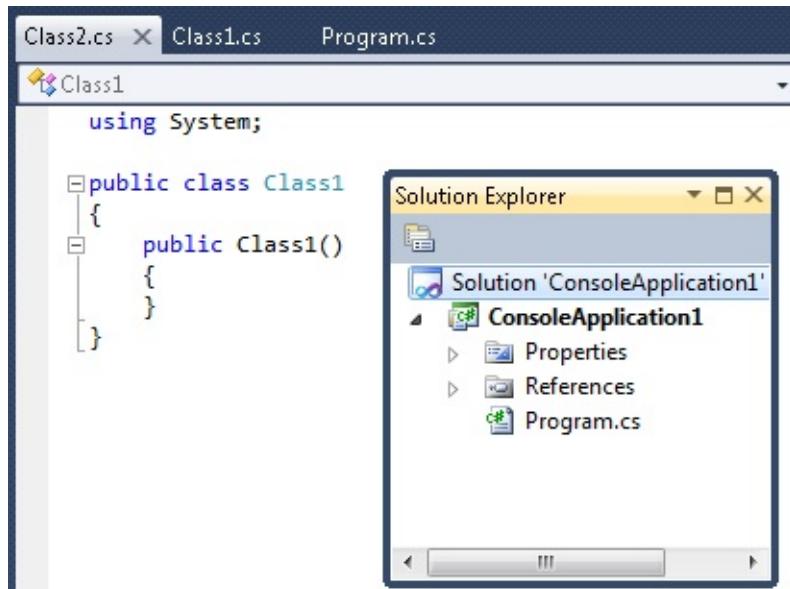


When the preview matches where you want your window to go, just release the mouse and the window docks at that location.

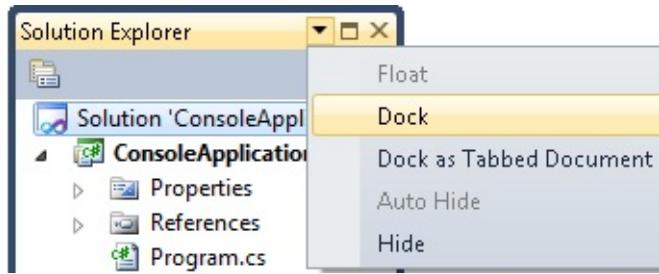
## 03.02 Dock a Floating Tool Window Back to Its Previous Location

<b>DEFAULT</b>	[no shortcut]
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	[no shortcut]
<b>VISUAL C++ 2</b>	Alt+F6
<b>VISUAL C++ 6</b>	[no shortcut]
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt,W, K
<b>MENU</b>	Window   Dock
<b>COMMAND</b>	Window.Dock
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0036

You can easily dock a floating tool window back to its previous docked location.



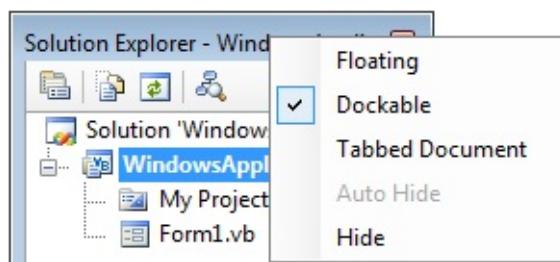
Just right-click the Title Bar, and choose Dock, as shown in the following illustration.



### NOTE

In Visual Studio 2008, you can also double-click the title bar to dock the window back to its previous location.

In Visual Studio 2005, to place a tool window back to its previous docking location without using the docking guides, you must double-click the title bar. Additionally, the title bar menu is slightly different for 2005. The word "Dockable" is used in place of the word "Dock."



## 03.03 Cycle Through Your Open Tool Windows

<b>DEFAULT</b>	Alt+F6 (next); Alt+Shift+F6 (previous)
<b>VISUAL BASIC 6</b>	Alt+F6 (next); Alt+Shift+F6 (previous)
<b>VISUAL C# 2005</b>	Alt+F6 (next); Alt+Shift+F6 (previous)
<b>VISUAL C++ 2</b>	F6 (next); Shift+F6 (previous)
<b>VISUAL C++ 6</b>	Alt+F6 (next); Alt+Shift+F6 (previous)
<b>VISUAL STUDIO 6</b>	Alt+F6 (next); Alt+Shift+F6 (previous)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Window.NextPane; Window.PreviousPane
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0038

In vstipTool0023, [04.08 Using the IDE Navigator](#), page 160, you saw how to get around among your open tool windows. Here's how to get around among your open tool windows without using the IDE Navigator.

Press Alt+F6 (next) or Alt+Shift+F6 (previous) to begin going through your open tool windows. It's important to understand what the word "open" means in this context. An "open" tool window is one whose tab appears in the IDE. For example, suppose you have the following view:

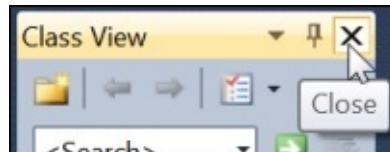


The Error List, Command Window, and Breakpoints are "open" tool windows—even though the tool windows are hidden. This is an important distinction as you use this tip, because it explains why you cycle through all the open tool windows—whether or not they are hidden.

## 03.04 Closing Tool Windows

<b>DEFAULT</b>	Shift+Esc
<b>VISUAL BASIC 6</b>	Shift+Esc
<b>VISUAL C# 2005</b>	Shift+Esc
<b>VISUAL C++ 2</b>	Shift+Esc
<b>VISUAL C++ 6</b>	Shift+Esc
<b>VISUAL STUDIO 6</b>	Shift+Esc
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Window.CloseToolWindow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0039

Eventually, after using your tool windows, you will want to close one or more of them. You can always do this by clicking the Close button (the “X” in the upper-right corner).

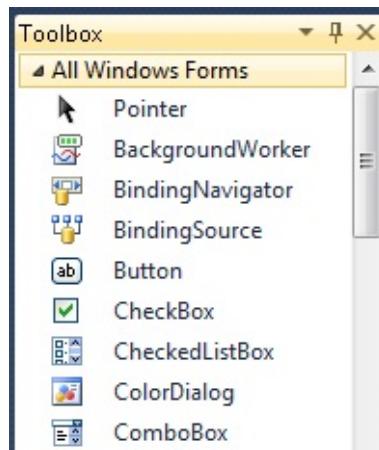


Using the keyboard, you can simply press Shift+Esc to close the current tool window.

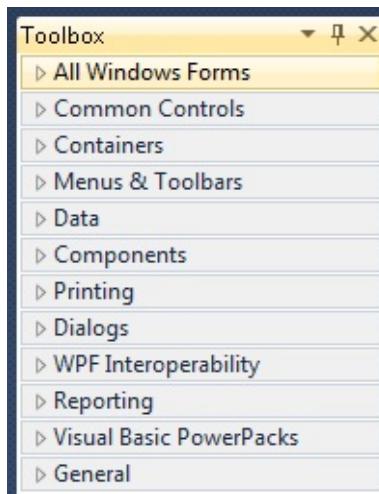
## 03.05 Expand and Collapse All in the Toolbox

<b>WINDOWS</b>	* (expand all); / (collapse all)
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0050

You can quickly expand the entire Toolbox by pressing the asterisk (\*) when the Toolbox is active.



You can also collapse the entire Toolbox by pressing the forward slash (/) when the Toolbox is active.

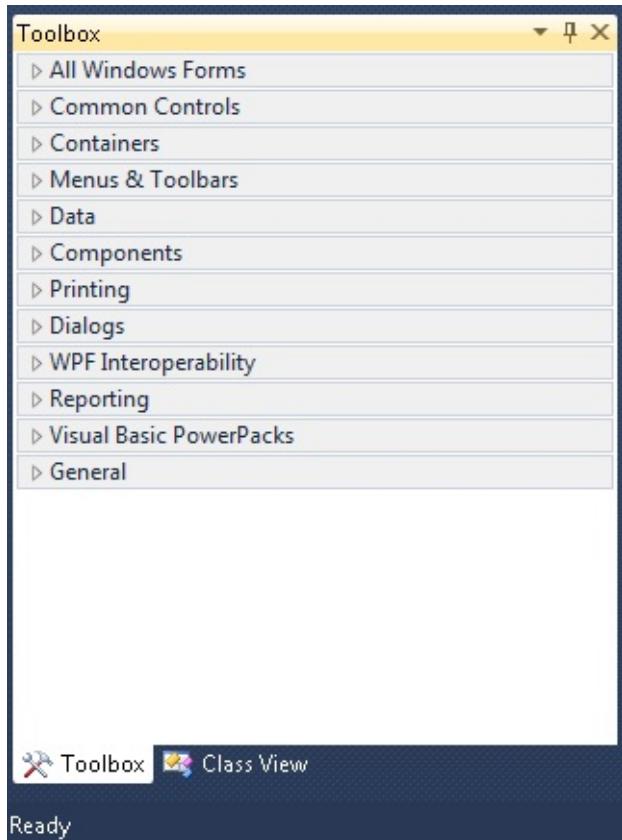


## 03.06 Searching in the Toolbox

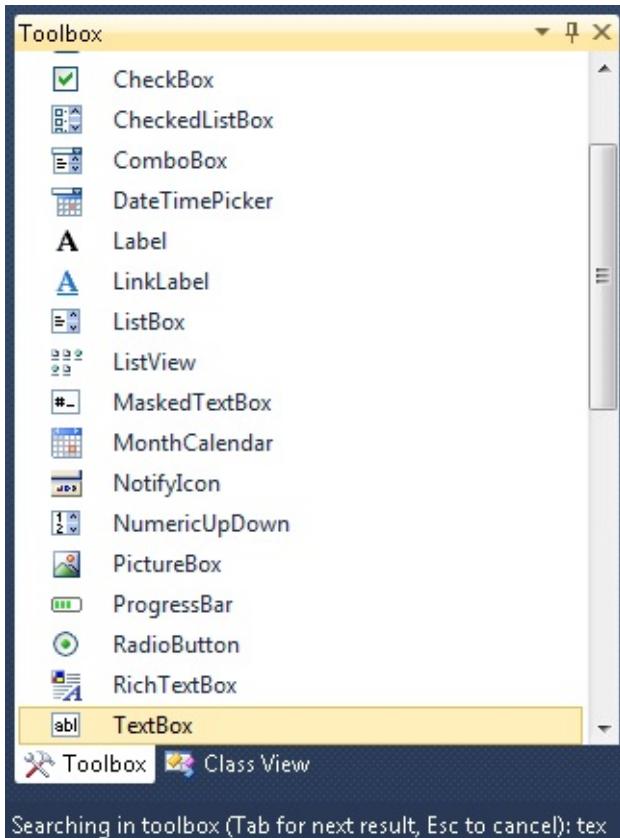
<b>DEFAULT</b>	Ctrl+Alt+X (view toolbox)
<b>VISUAL BASIC 6</b>	Ctrl+Alt+X (view toolbox)
<b>VISUAL C# 2005</b>	Ctrl+Alt+X (view toolbox); Ctrl+W, X (view toolbox); Ctrl+W, Ctrl+X (view toolbox)
<b>VISUAL C++ 2</b>	Ctrl+Alt+X (view toolbox)
<b>VISUAL C++ 6</b>	Ctrl+Alt+X (view toolbox)
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+X (view toolbox)
<b>WINDOWS</b>	Alt,V, X (view toolbox); TAB (next result); ESC (cancel)
<b>MENU</b>	View   Toolbox
<b>COMMAND</b>	View.Toolbox
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0114

This tip provides a much-requested and much-anticipated feature: searching the Toolbox.

Simply switch focus to the Toolbox (Ctrl+Alt+X), as shown in the following illustration.



Now start typing the name of the control you are looking for. In the following example, I'm looking for the TextBox control by typing **tex**:



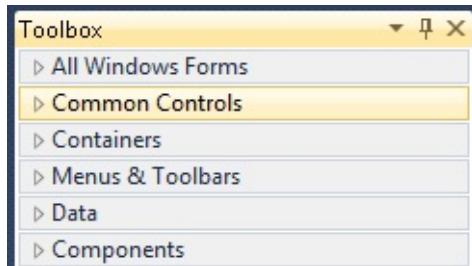
The letters you are typing appear in the Status Bar, and you are even provided instructions either for looking for the next item or for cancelling the search.

Press Tab to go the next result, or press Esc to cancel. Also, you can actively use the Backspace key to delete letters from the search when you want to quickly retype new characters.

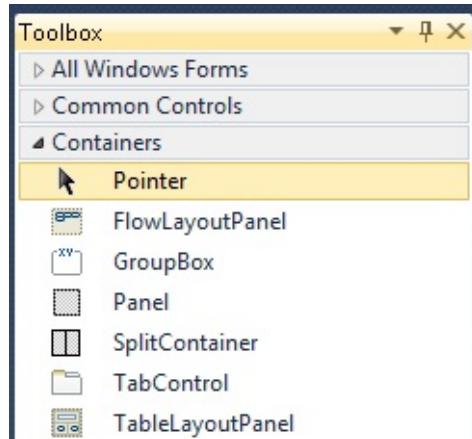
## 03.07 Navigate Among Tabs in the Toolbox

<b>KEYBOARD</b>	Ctrl+[Up, Down] Arrow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0051

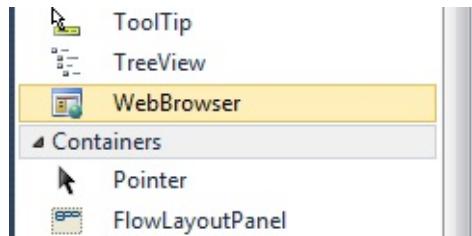
You can jump between tabs in the Toolbox.



Just press Ctrl+[Up, Down] Arrow to navigate. When you use Ctrl+Down Arrow, it expands the next tab and jumps to the first item in that group, as shown in the following illustration.



When you press Ctrl+Up Arrow it jumps to the last item in the previous control group:

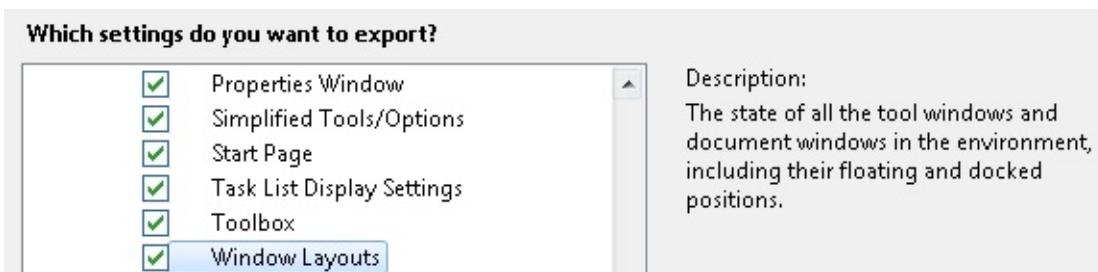


## 03.08 Window Layouts: The Four Modes

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0051

Ever wonder why the windows seem to shift around when you go from Design to Debug Mode? The answer is simple: window layouts.

You might have seen them if you have ever tried to export your window layouts. You can find it under General Settings | Window Layouts, as shown in the following illustration.



The four window layout modes in Visual Studio are as follows.

### Design View

This view is the one you see when you start up Visual Studio. It's what most people refer to as the "normal" view.

### Debugging View

This is the view that you get when you enter Debug Mode as you are stepping through your code.

### Full Screen

This is the view that you get when you go to View | Full Screen (Shift+Alt+Enter).

### File View

This is the lesser-known view that you can get when you open up a file via devenv.exe [filename].

```
\ConsoleApplication1>devenv.exe class1.cs
```

The thing to remember here is that your tool windows and your command bar customizations are saved separately for each state. There is no way to tell Visual Studio to use one state for all modes at this time. Additionally, when you shut down Visual Studio in any state, all four states are saved.

## 03.09 Window Layouts: Design, Debug, and Full Screen

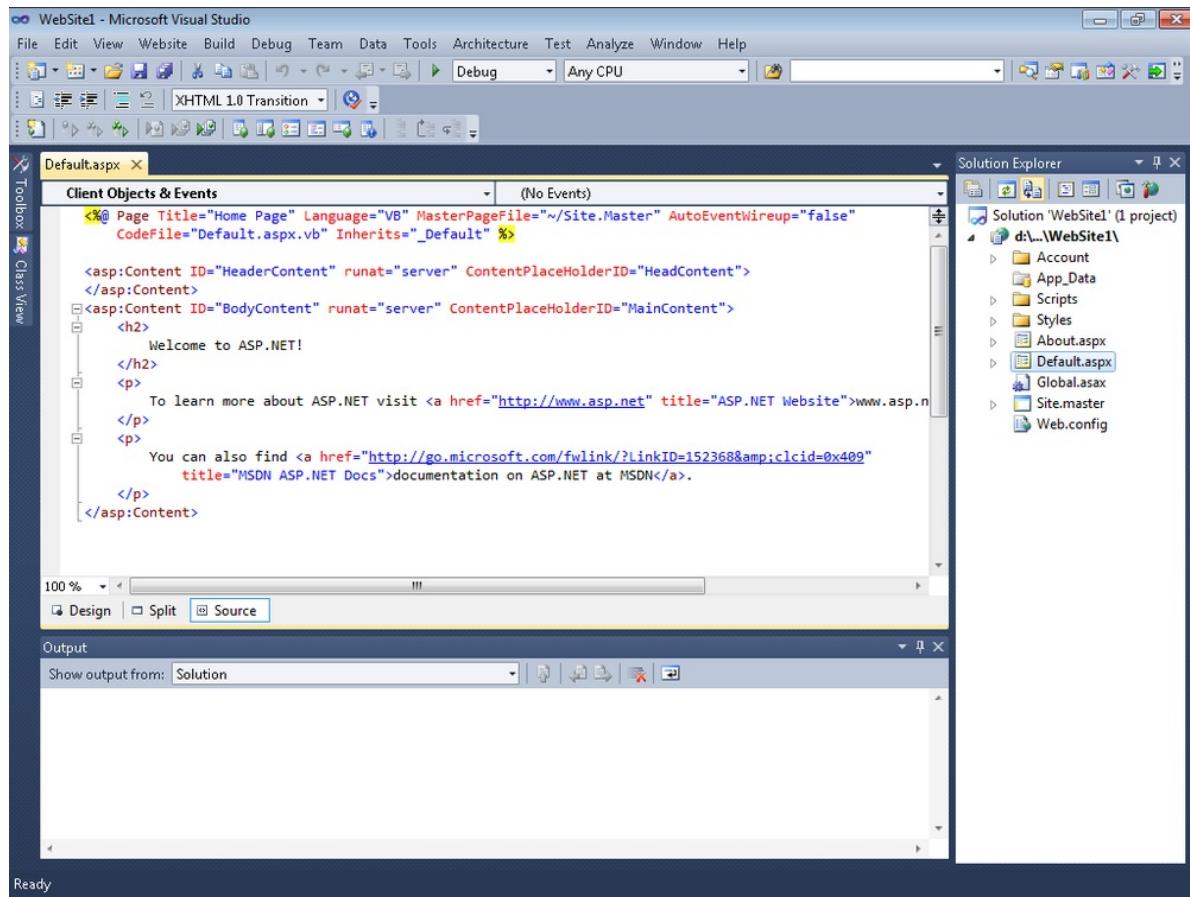
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0052

In vstipEnv0051, [03.08 Window Layouts: The Four Modes](#), page 90, I discussed the four layout modes in Visual Studio. I thought it would be instructive to demonstrate the three most common modes together here.

As we discuss these modes, keep in mind that each has its own layout that can be customized to your needs. For example, you might clearly need some windows in Design Mode (for example, the Pending Changes window) that perhaps aren't necessary in Debug Mode.

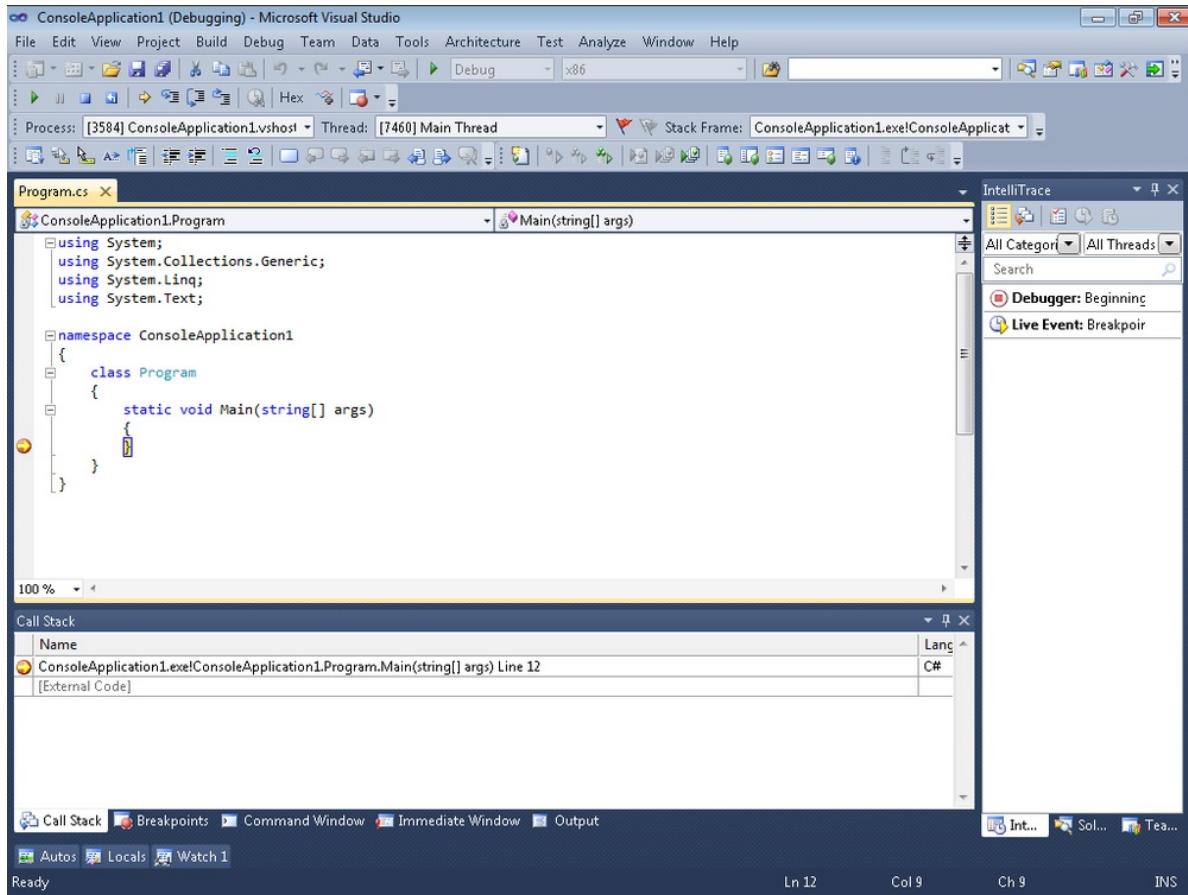
### Design Mode

This is the mode you see when you first start up Visual Studio. It is one of the two most common modes you will find yourself in. The following illustration shows a view of my Design Window Layout for a website.



## Debug Mode

When I enter Debug Mode, the second most common mode, I see my Debug Window Layout, as shown in the following console application.



## Full Screen Mode

I addressed this mode in vtipEnv0024, **AX.20 Full Screen Mode**, in Appendix B (<http://go.microsoft.com/fwlink/?LinkId=223758>). You can get here by pressing Shift+Alt+Enter. An example of what it looks like is shown in the following illustration.

The screenshot shows the Microsoft Visual Studio IDE interface. The menu bar includes File, Edit, View, Website, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, Help, and Full Screen. The title bar says "Default.aspx". The main area is titled "Client Objects & Events" and shows "(No Events)". Below this is the ASP.NET source code:

```
<%@ Page Title="Home Page" Language="VB" MasterPageFile("~/Site.Master" AutoEventWireup="false"
CodeFile="Default.aspx.vb" Inherits="_Default" %>

<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
    <h2>
        Welcome to ASP.NET!
    </h2>
    <p>
        To learn more about ASP.NET visit <a href="http://www.asp.net" title="ASP.NET Website">www.asp.net</a>.
    </p>
    <p>
        You can also find <a href="http://go.microsoft.com/fwlink/?LinkId=152368&clcid=0x409"
            title="MSDN ASP.NET Docs">documentation on ASP.NET at MSDN</a>.
    </p>
</asp:Content>
```

The status bar at the bottom shows "100 %", "Design | Split | Source" (with Source selected), "Ln 1", "Col 1", "Ch 1", and "INS".

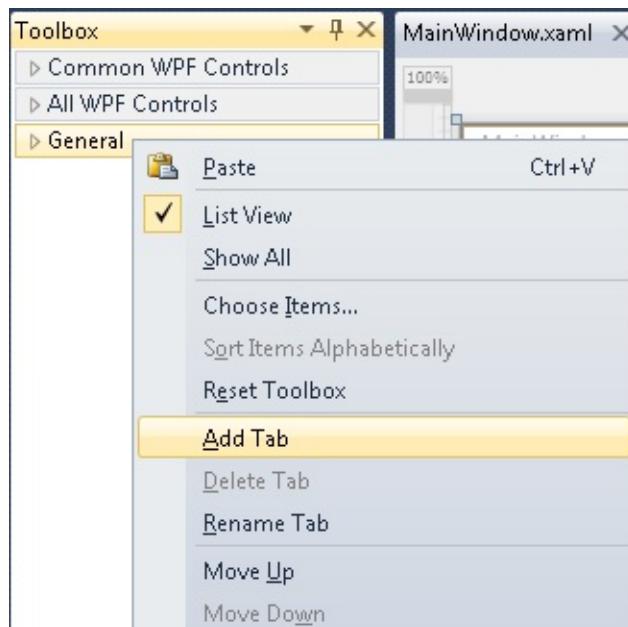
## 03.10 Working with Tabs in the Toolbox

<b>WINDOWS</b>	Shift+F10, A (with the Toolbox selected)
<b>MENU</b>	[Right-click the Toolbox]   Add Tab
<b>COMMAND</b>	Tools.AddTab
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0054

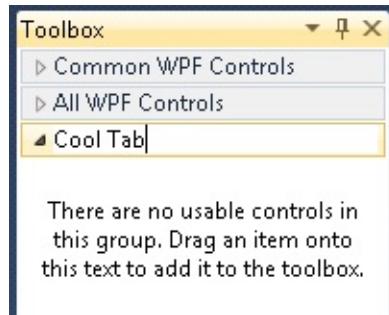
The Toolbox is a pretty cool place, and one of its best features is the ability to organize items by using tabs.

### Creating Tabs

To create a custom tab, right-click inside the Toolbox and choose Add Tab, as shown in the following illustration.

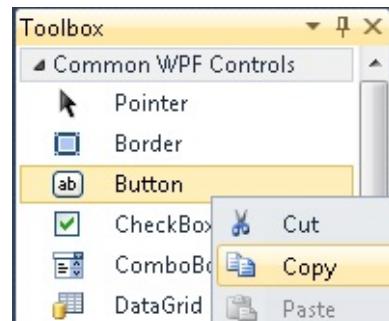


Just type in a name for your new tab, and press Enter.

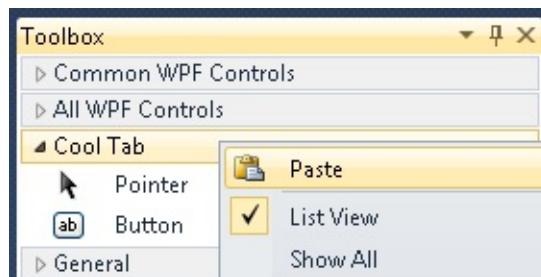


## Adding Items

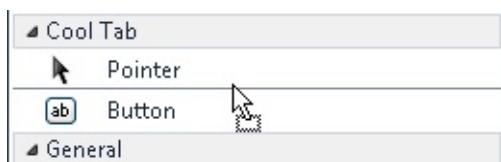
You can add items to your custom tab as you see fit. For example, to add controls to this new tab from existing tabs, just pick the control you want and copy it.



Then go to your customized tab, and paste inside it to get a copy of that control for your use.

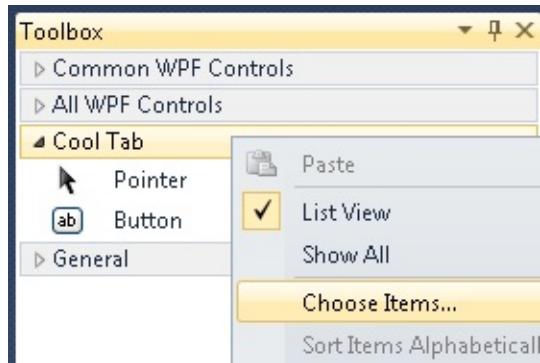


You can also click and drag items onto new tabs.



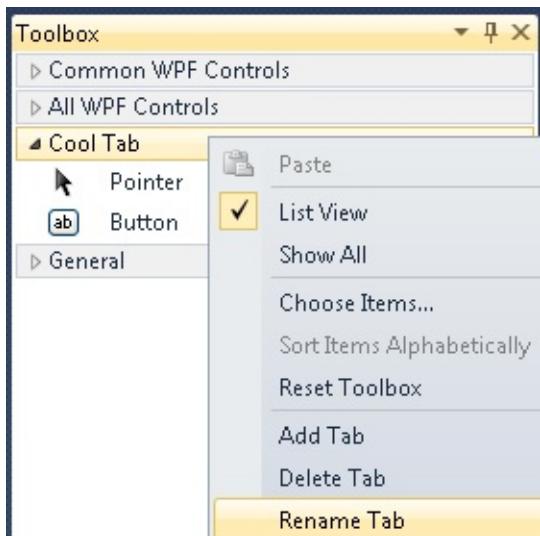
## Custom Controls

Of course, if you want custom controls, you can always right-click in the custom tab and select Choose Items, as shown in the following illustration.



## Renaming Tabs

If you don't like a tab name, you can always rename it:



## Deleting Tabs

Also notice the option to delete the tab, shown in the preceding illustration. If you choose this option, the following dialog box appears. When you click OK in this dialog box, you lose the tab as well as all the items on it.



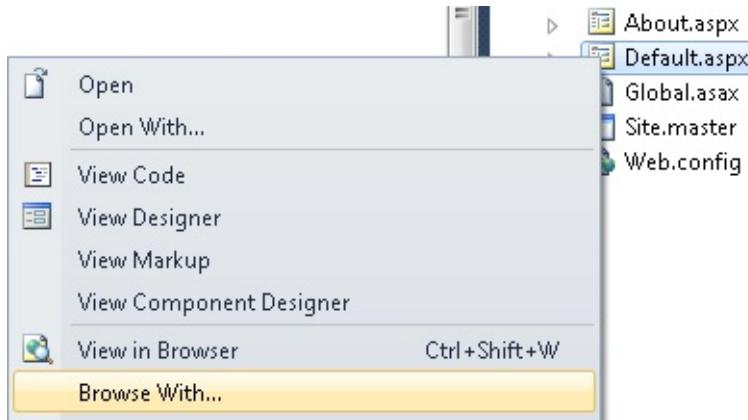
This action is just an organizational mechanism and doesn't permanently delete any controls from your system, so you can add them as needed to any future tabs.

## 03.11 Using Additional Browsers for Web Development

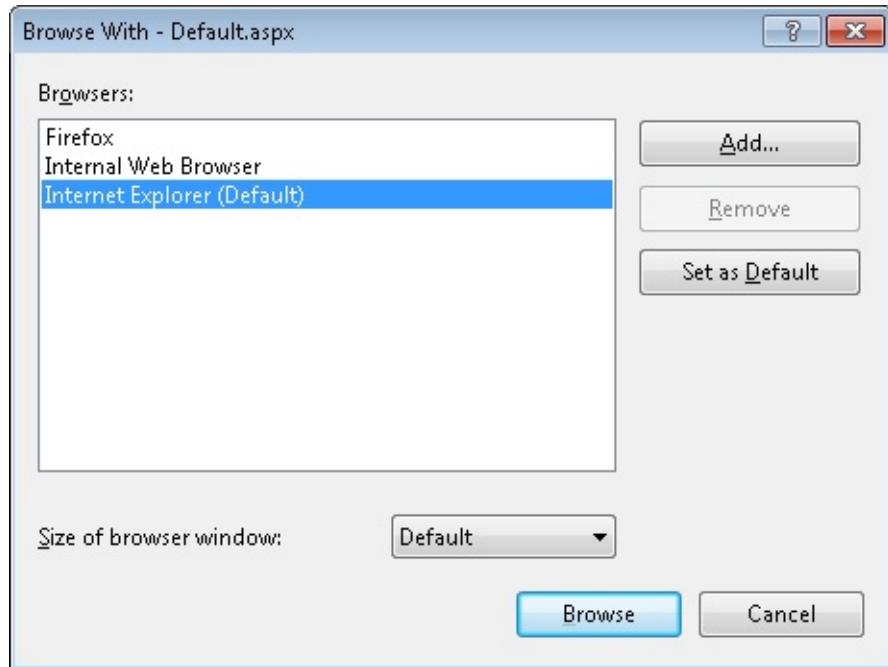
<b>WINDOWS</b>	Alt,F, H (with file selected in Solution Explorer)
<b>MENU</b>	File   Browse With (with file selected in Solution Explorer)
<b>COMMAND</b>	File.BrowseWith (with file selected in Solution Explorer)
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0057

When you are doing web development in Visual Studio, you might want to use a different default browser than you are currently using. You can do this by using the Browse With dialog box.

Getting to this dialog box is a little interesting because it is context sensitive. It is best to have either your web project or a webpage selected in Solution Explorer to see the Browse With option on the File Menu or when you right-click:

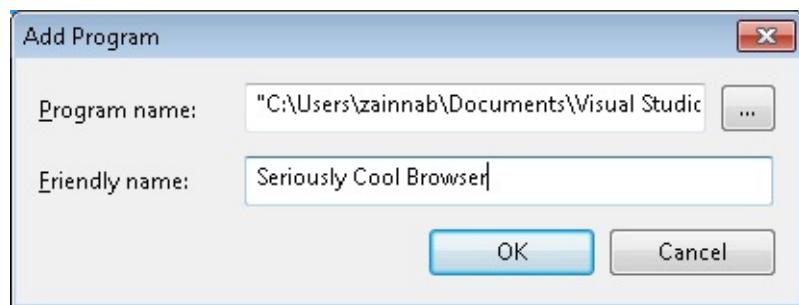


The Browse With dialog box appears as shown in the following illustration.



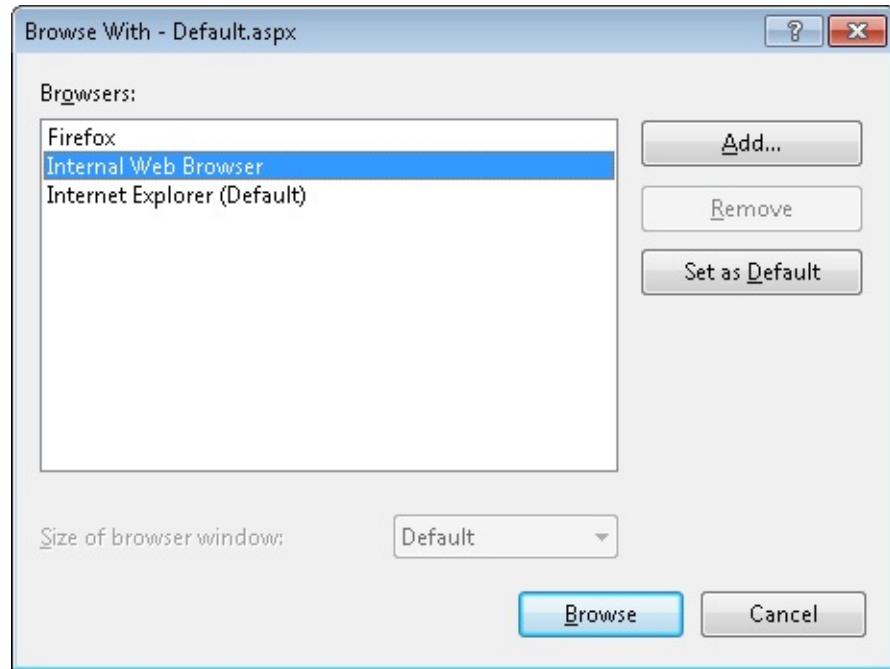
## Adding New Browsers

Visual Studio automatically detects some browsers. For example, I installed Firefox and the preceding dialog box automatically detected it. However, if you don't see your browser in the Browse With dialog box, you can click Add, enter the path to the executable in the Program Name field, and enter a friendly name for your browser in the Friendly Name field, as shown in the following illustration.



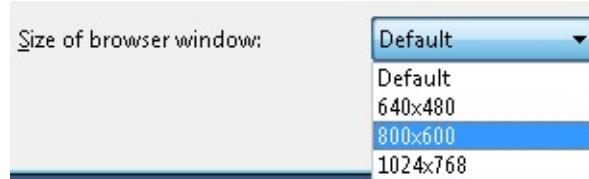
## Changing the Default Browser

You can also change the default browser by choosing a browser in the list and then clicking Set As Default:



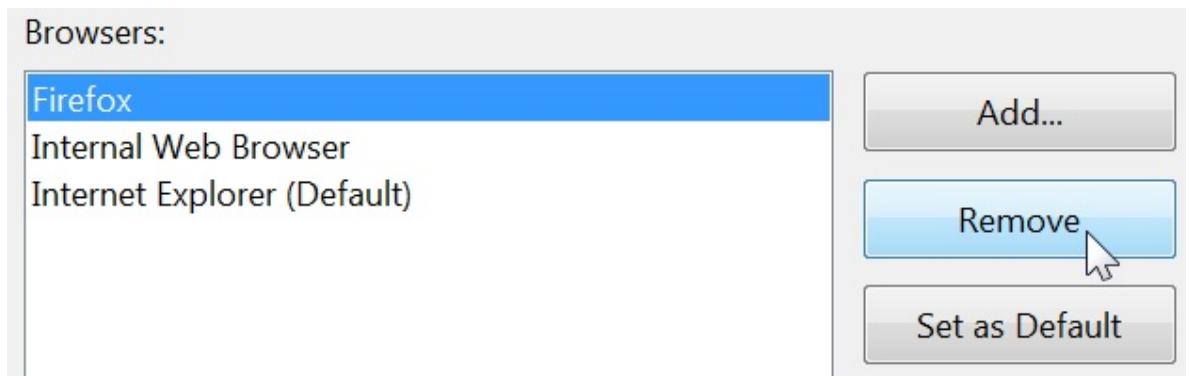
## Browser Window Size

Choose the window size you want for your browser by using the Size Of Browser Window drop-down list:



## Removing Browsers

Eventually, you might want to get rid of some of your browser choices. Simply select the browser in the list, and click Remove:



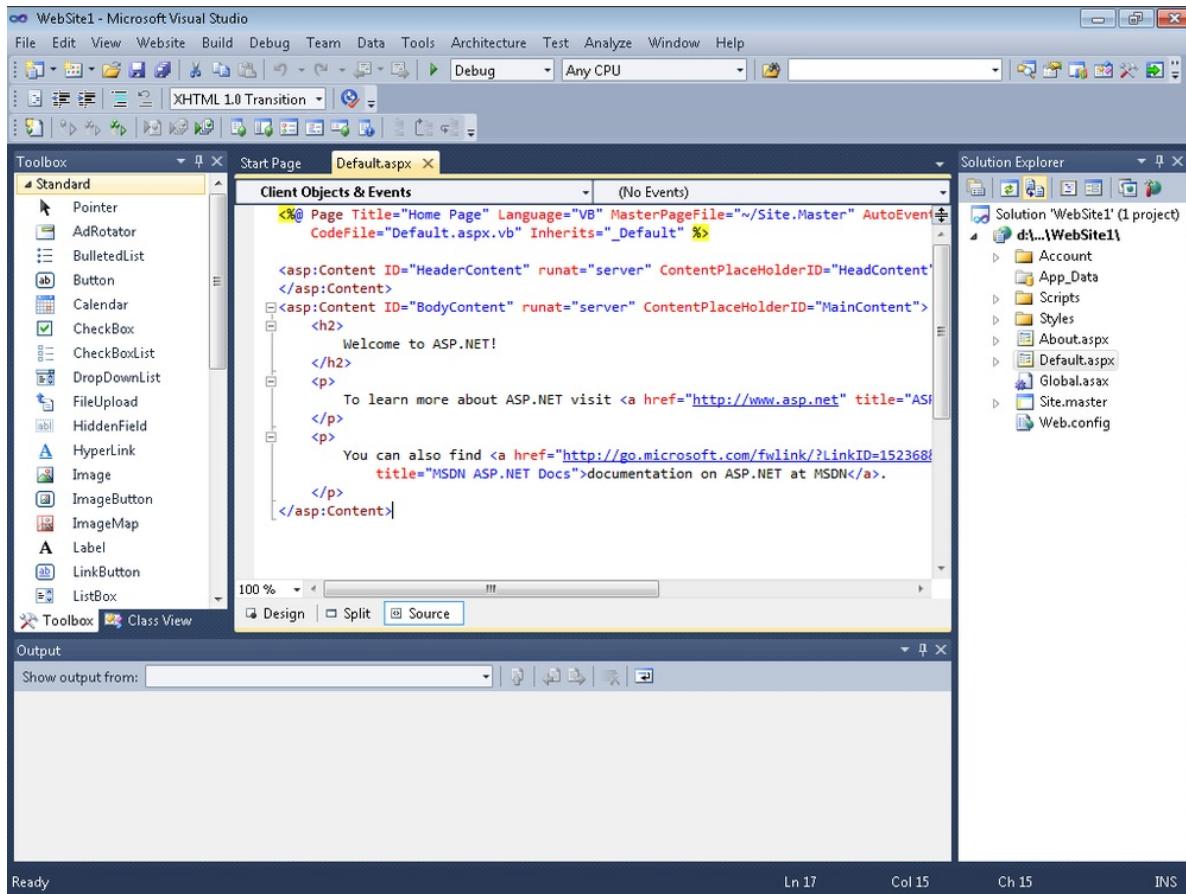
## 03.12 Auto-Hide All Tool Windows

<b>WINDOWS</b>	Alt,W, U
<b>MENU</b>	Window   Auto Hide All
<b>COMMAND</b>	Window.AutoHideAll
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0034

### WARNING

While this is a great tip, there is no way to “un-auto-hide” all tool windows, so you have to bring your tool windows back individually.

So let's say you have a crowded space with lots of tool windows open, as shown in the following illustration.

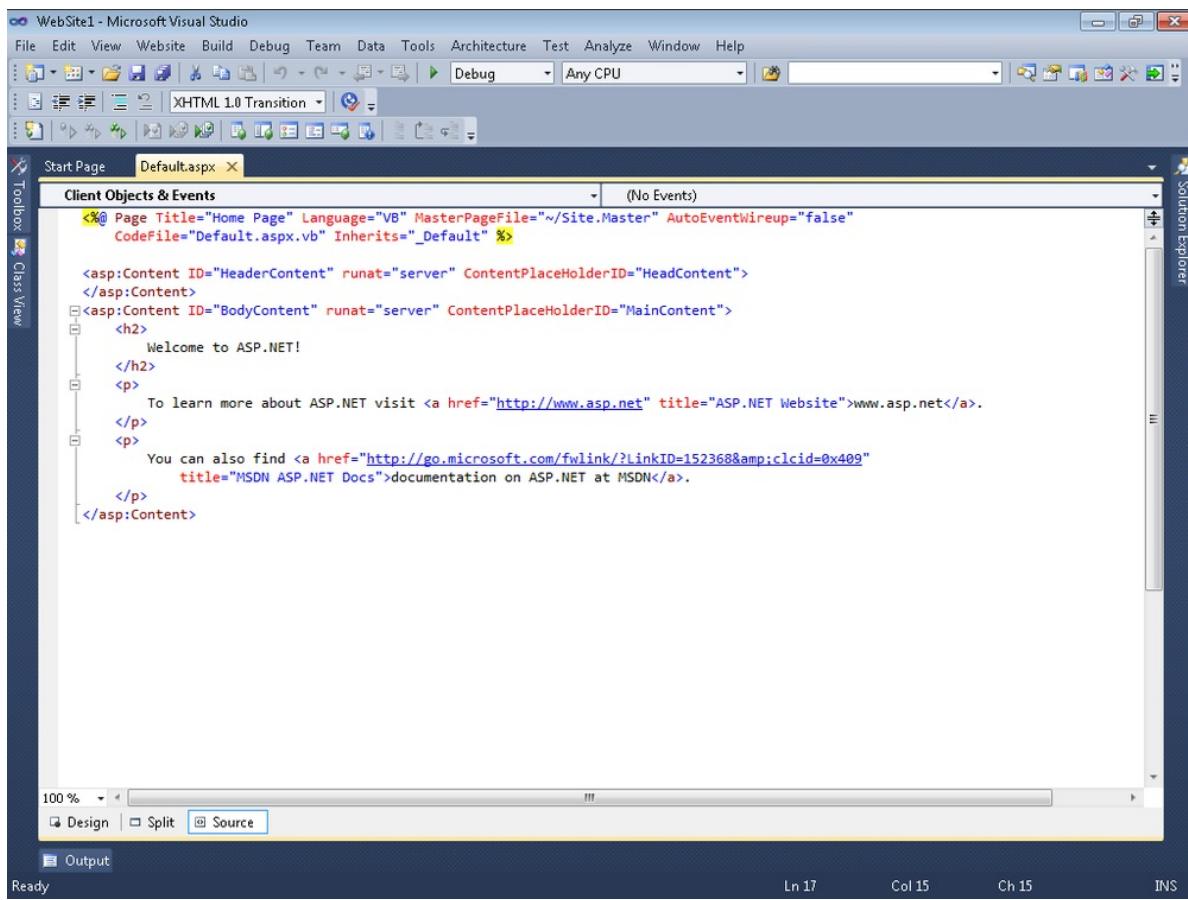


You can make all the tool windows go away quickly—just go to Window | Auto

## Hide All:



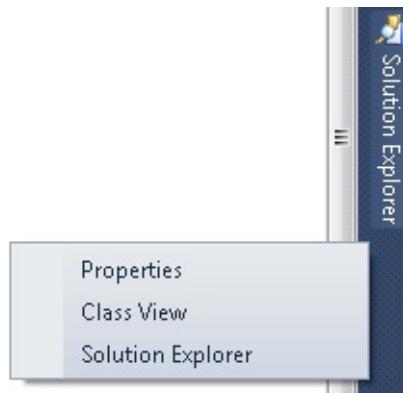
All the tool windows are automatically hidden.



## 03.13 Showing Hidden Tool Windows with the Auto Hide Channel

VERSIONS	2005, 2008, 2010
CODE	vstipTool0037

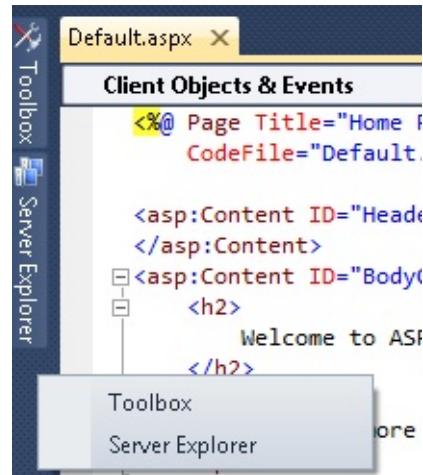
You can use a special place called the Auto Hide Channel to see what tool windows are hidden. Just go to any area that has hidden tool windows, and then right-click the bar where the tabs are to see a list of the hidden tool windows.



The best part is that this works on the bottom channel.



And it works on the channel to the left as well.



For best results, click in the empty space in the channel, beyond any tabs, as shown in the following illustration.



And, of course, to show any hidden window, just select it from the list.



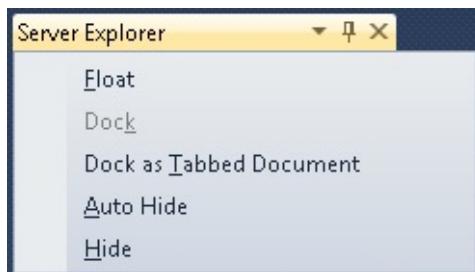
## 03.14 Moving Tool Windows Around with Your Keyboard

<b>DEFAULT</b>	Alt+- (dock menu)
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	Alt+- (dock menu)
<b>VISUAL C++ 2</b>	Alt+- (dock menu)
<b>VISUAL C++ 6</b>	Alt+- (dock menu)
<b>VISUAL STUDIO 6</b>	Alt+- (dock menu)
<b>WINDOWS</b>	Alt,Space (floating tool windows)
<b>MENU</b>	Window   [Float, Dock, etc.] (dock menu)
<b>COMMAND</b>	Window.ShowDockMenu
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0041

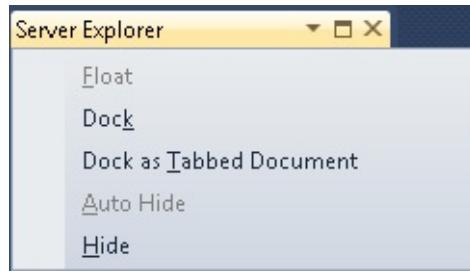
### NOTE

The minus sign (-) used in the keyboard shortcut is from the top row of numeric keys on your keyboard, not the minus sign on the numeric keypad.

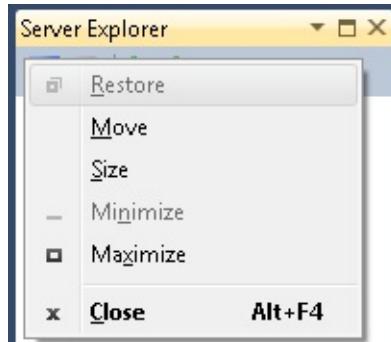
With the rewrite of the IDE for Visual Studio 2010, you'll find some changes in how you control the tool windows. When an active tool window is docked, you can use Alt+minus (-) to bring up the Dock menu and then use your arrow keys to pick an item from the menu.



This works for floating tool windows as well.



But active floating tool windows have a System menu that you can access *only* by pressing Alt+Space.

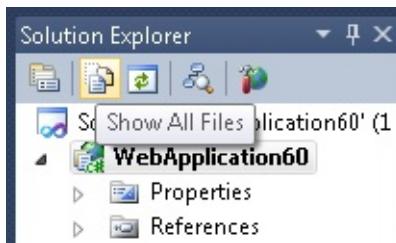


These commands should be familiar to just about everyone, and they give you full control over moving, resizing, and other window manipulations.

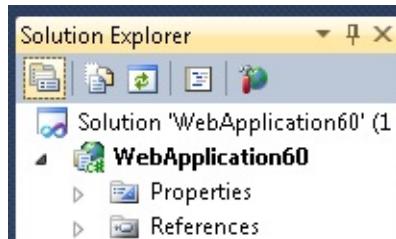
## 03.15 Keyboard Access to a Tool Window's Toolbar

<b>WINDOWS</b>	Shift+Alt; Tab
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0042

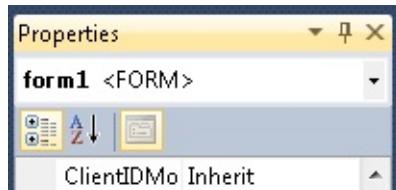
Sometimes, when you have an active tool window, you just want access to the toolbar in the window without having to reach over and use your mouse.



A little-known fact is that you can do this very easily for any active tool window by pressing Shift+Alt for some or by pressing Tab for others. For example, to get access to the Solution Explorer Toolbar, just press Shift+Alt.



In the Properties Tool Window, you would use press the Tab key to gain access to the toolbar.



Now you can use your arrow keys to move between toolbar items, and use the Enter key to “press” the button you choose.

## 03.16 Command Prompt History

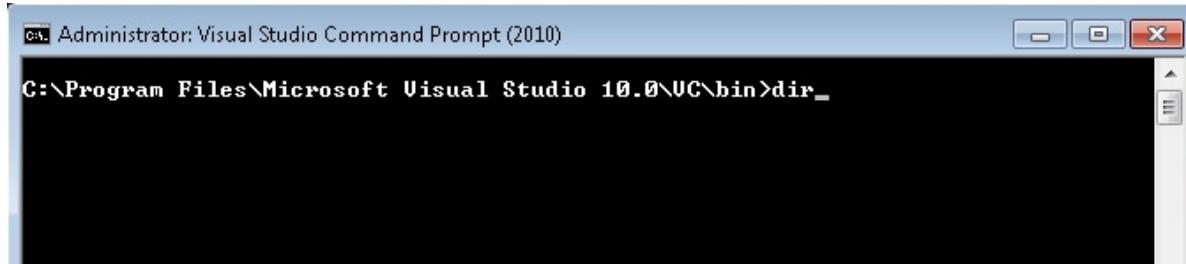
WINDOWS	F7 (history window); [Up / Down] Arrow (history)
CODE	vstipTool0055

Many people like to use the command prompt; I thought we might explore one of the oldest features around: command history.

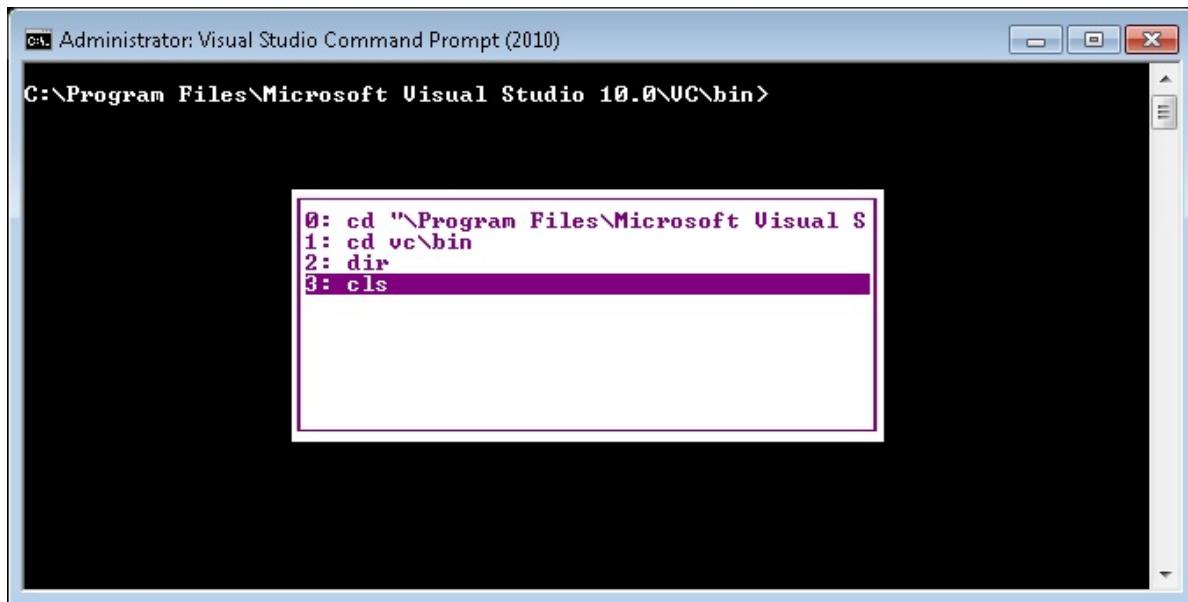
### NOTE

The following examples will not work unless you have typed some commands into the Command Window already, so type a few commands and then clear the screen by typing `cls` and pressing Enter.

There are two main ways to get commands you've typed in previously. First, you can just press the Up Arrow key to start going through your history at the prompt itself.



The advantage here is that you can quickly edit the command to change it if needed. However, if you just want to run a command from your history, you can use a very old trick by pressing F7.



The screenshot shows an Administrator Visual Studio Command Prompt window titled "Administrator: Visual Studio Command Prompt (2010)". The command prompt path is "C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>". A command history menu is displayed in a white box with a black border, containing the following entries:

- 0: cd "\Program Files\Microsoft Visual S
- 1: cd vc\bin
- 2: dir
- 3: cls

The entry "3: cls" is highlighted with a thick black horizontal bar.

Pressing F7 runs the command you selected from the list by using your Up or Down Arrow keys.

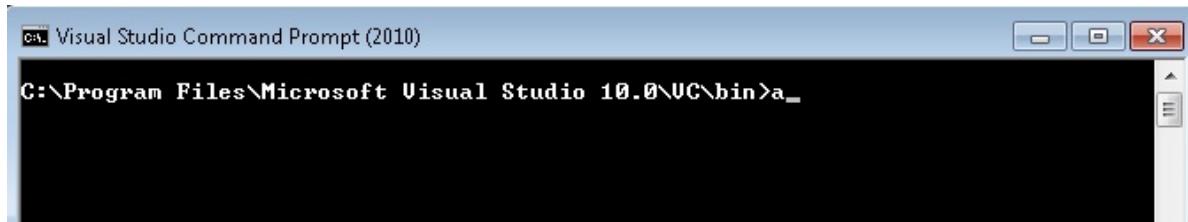
## 03.17 Command Prompt Tab Completion

WINDOWS	Tab
CODE	vstipTool0056

When using the Visual Studio command prompt (or any command prompt), you have several ways to use tab completion.

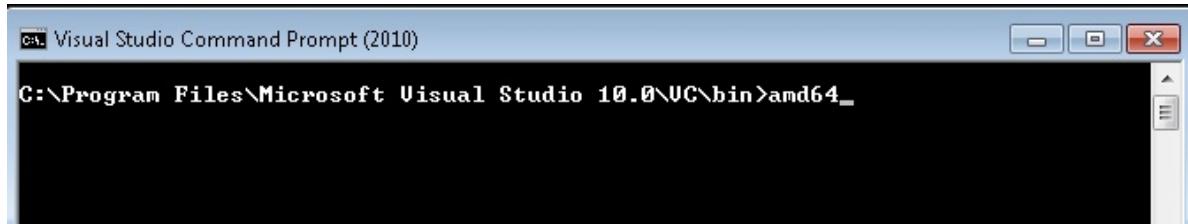
### Simple Search

You can type the first letter of a file, as shown in the following illustration.



A screenshot of a Windows Command Prompt window titled "Visual Studio Command Prompt (2010)". The window shows the command line: "C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>a\_". The cursor is positioned at the end of the letter 'a'.

Then press Tab one or more times to see all the files that begin with that letter.

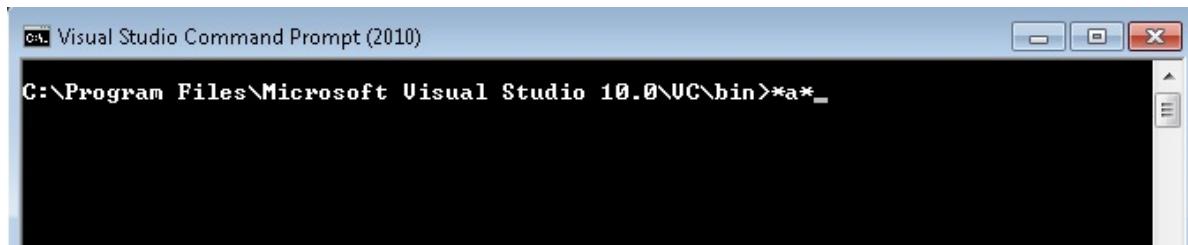


A screenshot of a Windows Command Prompt window titled "Visual Studio Command Prompt (2010)". The window shows the command line: "C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>amd64\_". The cursor is positioned at the end of "amd64\_".

### Wildcard Search

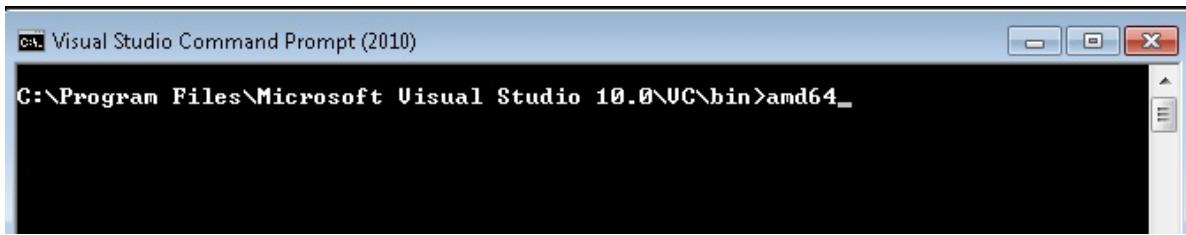
Not as well-known is the ability to use wildcards to match characters. You can use an asterisk (\*) to represent any number of characters and a question mark (?) to represent a single character.

So if you want to find a file name that has the letter “a” anywhere in it, you would use \*a\* as shown in the following illustration.



A screenshot of a Windows Command Prompt window titled "Visual Studio Command Prompt (2010)". The window shows the command line: "C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>\*a\*". The cursor is positioned at the end of "\*a\*".

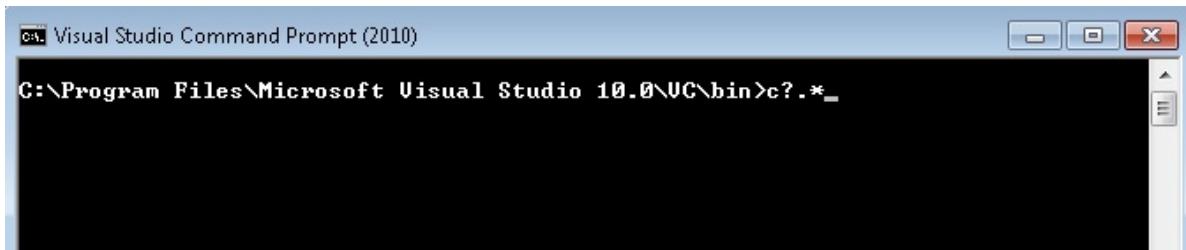
Then press Tab to get the first result.



```
Visual Studio Command Prompt (2010)
C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>amd64_
```

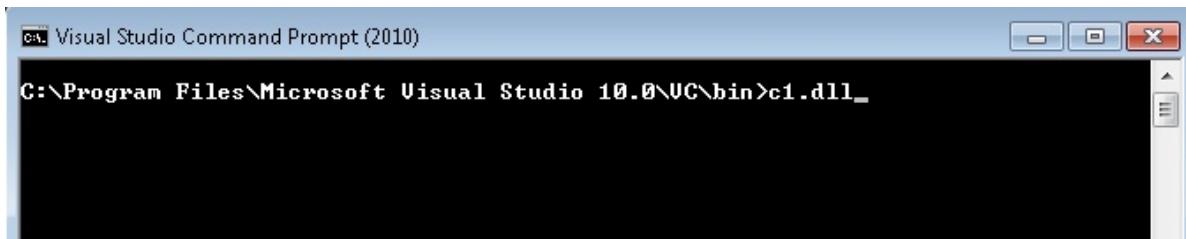
Press Tab several times, and notice that each file name listed contains the letter “a” somewhere in it.

What about a two-letter file that begins with a “c” but can have any other character and any extension? Use **c?.\*** and press Tab.



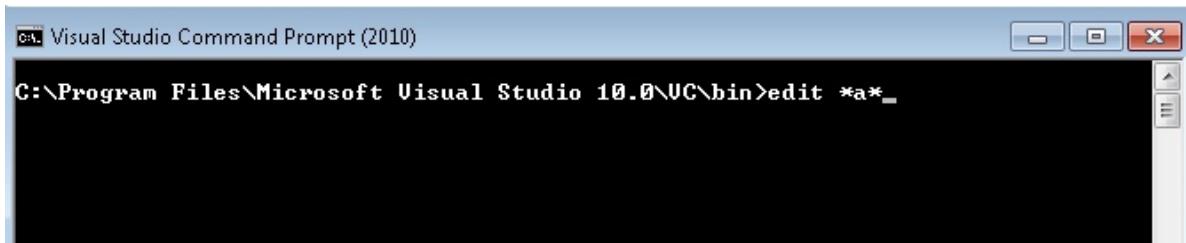
```
Visual Studio Command Prompt (2010)
C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>c?.*
```

Press Tab again.



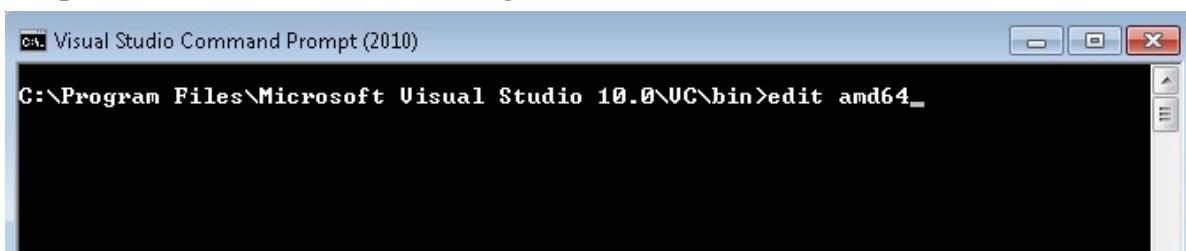
```
Visual Studio Command Prompt (2010)
C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>c1.dll
```

This feature extends to any commands you want to use as well. You can type something like **edit \*a\***, as shown in the following illustration.



```
Visual Studio Command Prompt (2010)
C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>edit *a*
```

Then press Tab to see the following result:



```
Visual Studio Command Prompt (2010)
C:\Program Files\Microsoft Visual Studio 10.0\VC\bin>edit amd64_
```

If I keep pressing tab, it continues to cycle through all file names that contain an “a”.

## Finally

As you can see, tab completion is a very useful and powerful feature with the command prompt. You definitely want this skill in your tool belt.

## 03.18 Undock and Dock a Single Tool Window in a Group

<b>DEFAULT</b>	[no shortcut]
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	[no shortcut]
<b>VISUAL C++ 2</b>	Alt+F6 (dock)
<b>VISUAL C++ 6</b>	[no shortcut]
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt,–, F (float, VS2010 Only); Alt,–, K (dock, VS2010 Only); Alt, W, F (float); Alt, W, K (dock)
<b>MENU</b>	Window   Float; Window   Dock
<b>COMMAND</b>	Window.Float; Window.Dock
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0099

Docking and undocking tool windows is a common activity. In this tip, we look at the different techniques you can use to accomplish these tasks.

### Undock

You have multiple ways to undock a single tool window in a group.

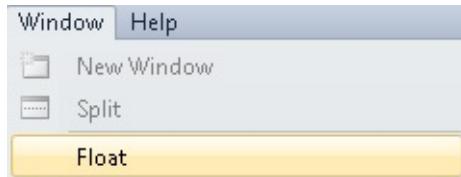
#### Click and drag

With the mouse, you can click and drag the tab out of the group.



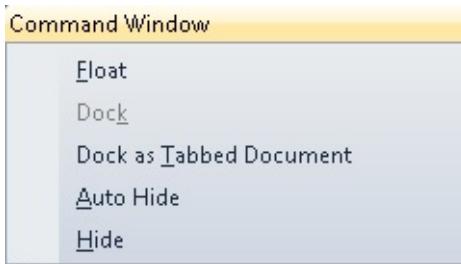
#### Menu

With the tool window active, you can select Window | Float from the menu bar, as shown in the following illustration.



## Control box (Visual Studio 2010 only)

Press Alt+minus (-) to get the tool window menu.



Then press “F” to make the tool window float.

## Result

Whichever method you use, the result is the same: You wind up with an undocked tool window.

```
</p>
</asp:Content>
```

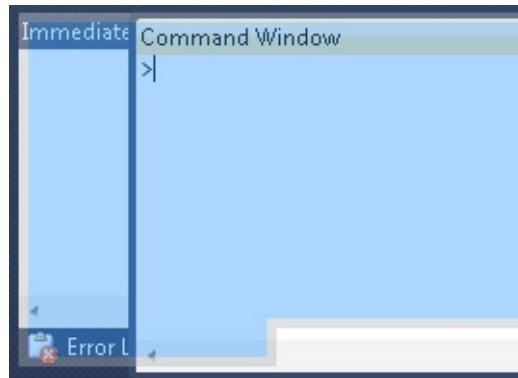


## Dock

To dock a floating tool window back into a group (assuming it came from a group), you also have multiple options.

### Click and drag

By far, the hardest option is to click and drag the tool window back into the group. The best way to do this is to drag the tool window title bar over the title bar of another tool window in the group you want it to join, as shown in the following illustration.



Notice how the target shading looks like a tab being put onto the existing group? That is what you look for when doing it this way. Of course, I would do it this way only if I were taking a tab from one group to another, *not* if I were returning a floating tool window back to its original group.

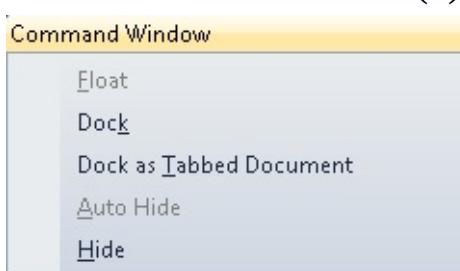
## Menu

The menu option is pretty easy: Just go to Window | Dock, as shown in the following illustration.



## Control box (Visual Studio 2010 only)

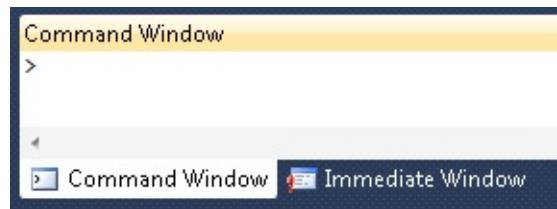
And then there is the Control menu. Press Alt+minus (-) to get the menu.



Then press “K” to make the tool window dock back to its original location.

## Result

You can use any of these methods to put the tool window back into the group it came from, with the notable exception of the click and drag method, which can be used to put the tool window anywhere.



## 03.19 Understanding Commands: Simple Commands

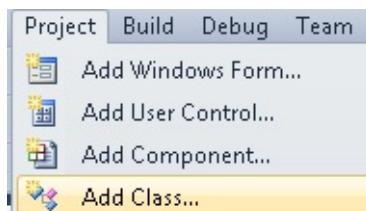
VERSIONS	2005, 2008, 2010
CODE	vstipTool0067

Just about everything you do in Visual Studio comes with an associated command. But what exactly is a command?

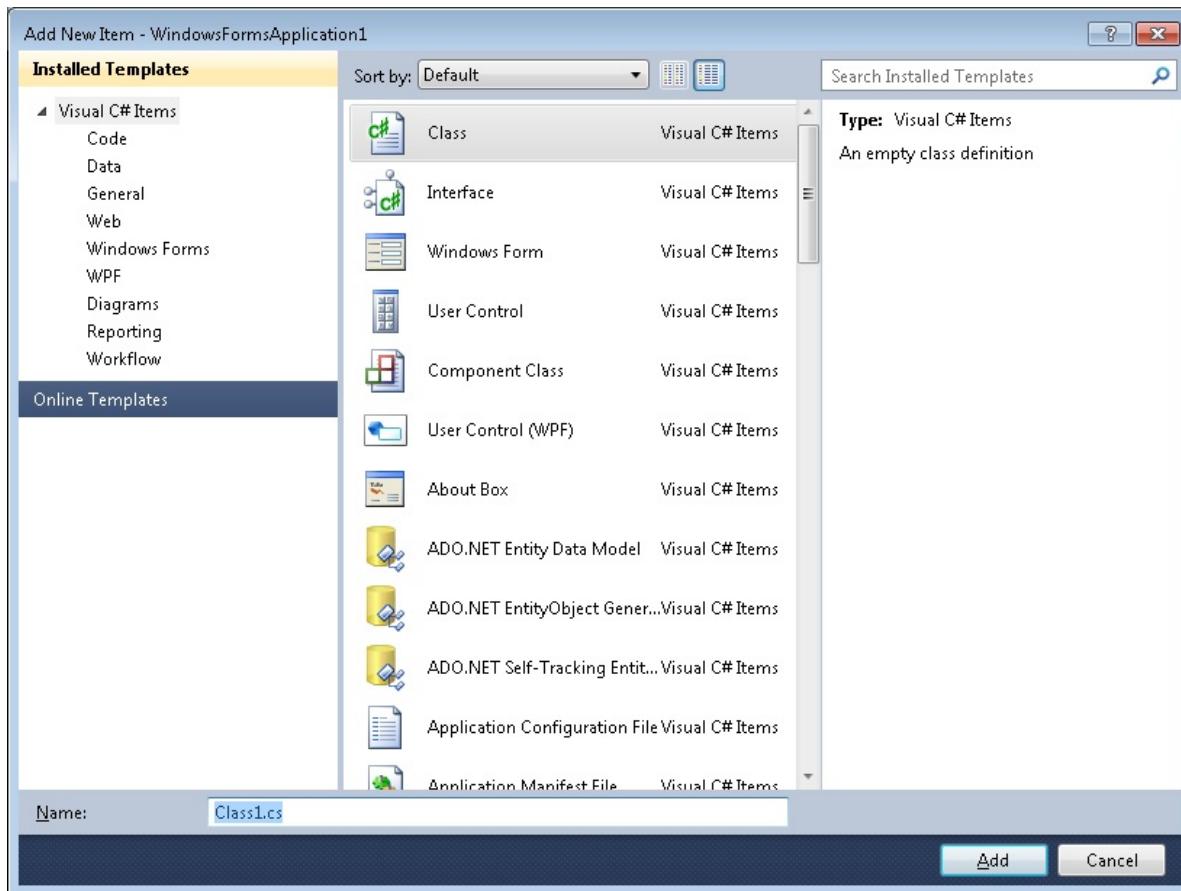
According to the MSDN documentation (<http://msdn.microsoft.com/en-us/library/kcc7tke7.aspx>), commands “allow direct interaction with the integrated development environment (IDE) from the keyboard. Dialog boxes, windows, and other items within the IDE have a command equivalent that you can type into the Command window or Find/Command box to display and, in some cases, execute the item.”

In plain English, commands allow you to perform actions in Visual Studio. Let’s take adding a class as an example.

First, let’s examine the typical way you add a class. Normally, you would just go to Project | Add Class.



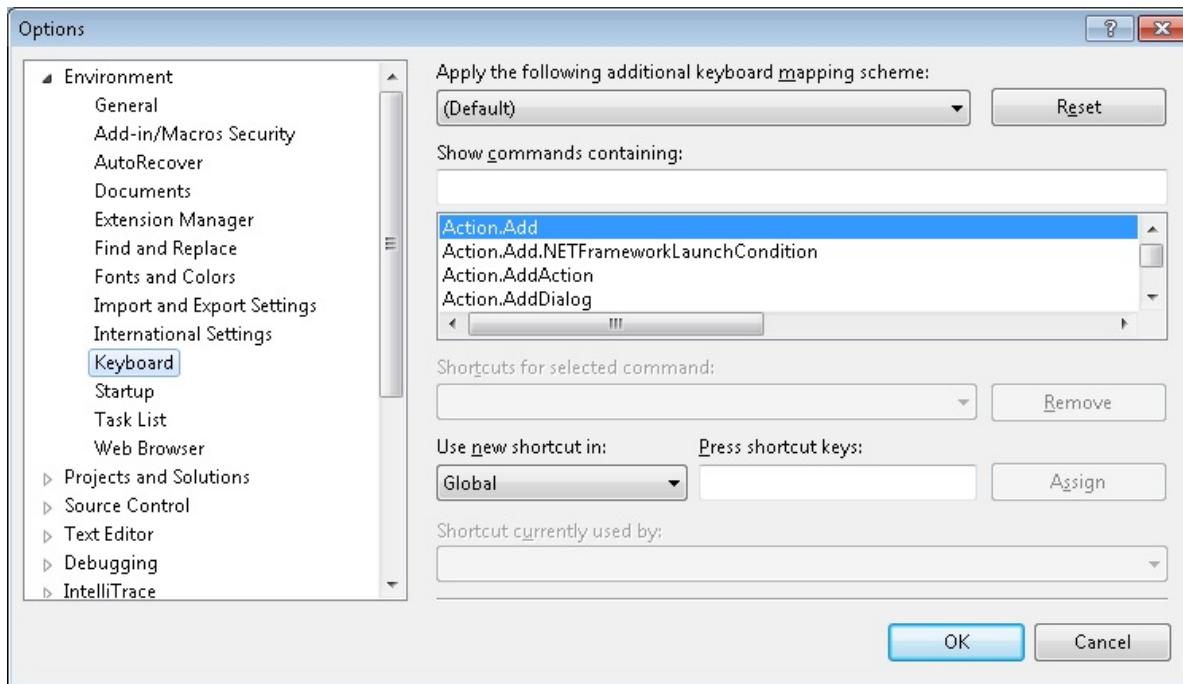
Obviously, this command is used to add a new class to your project and shows the Add New Item dialog box.



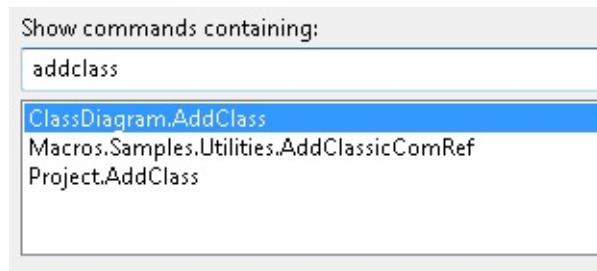
That's too much extra work for something I like to use all the time. I want to search and see whether a specific command is associated with this action. To do this, I'll use what most people refer to as the "command well," because it is a deep "well" of commands. It's where all the commands that you can use are located. To get there, go to Tools | Options | Environment | Keyboard, as shown in the following illustration

#### NOTE

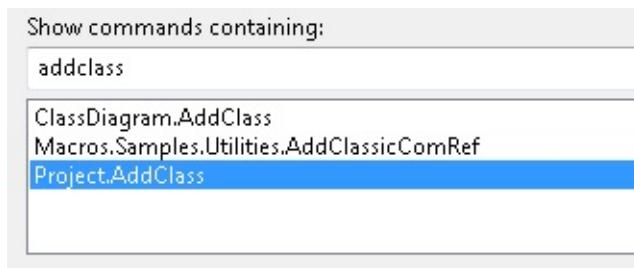
For this example, I'm using the General keyboard settings, so your settings might have a shortcut key assigned already. You can still follow this tip to create a new shortcut.



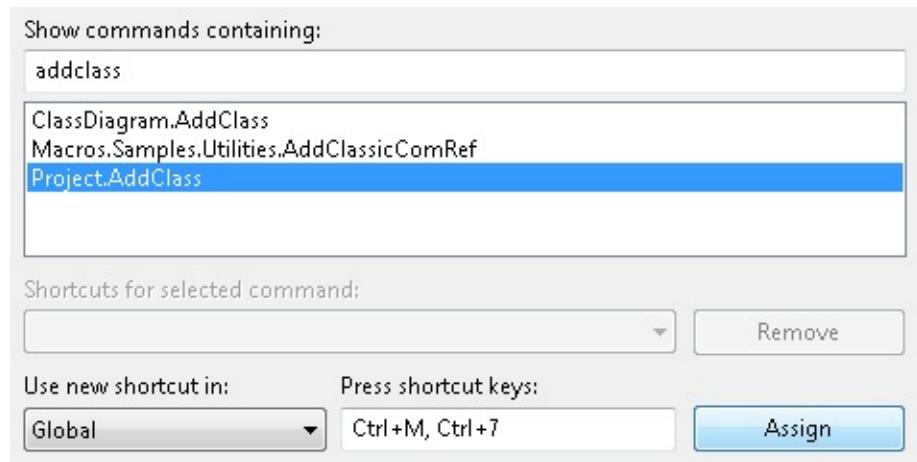
I'll type a keyword in the Show Commands Containing area to narrow down the command list. In this case, the keywords Add Class are used in the menu item, so I will use them here. In the following illustration, you can see how I have removed the spaces for the command.



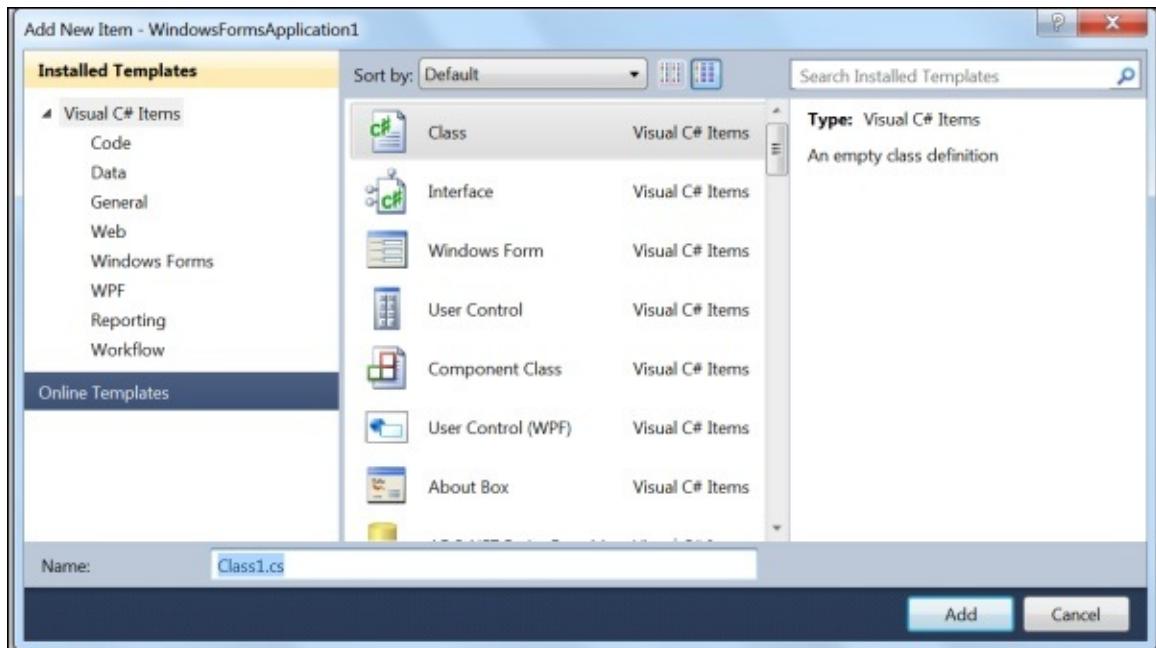
Notice that among the available commands is the Project.AddClass command. It's common to find a command that follows the menu structure, and this one is no exception.



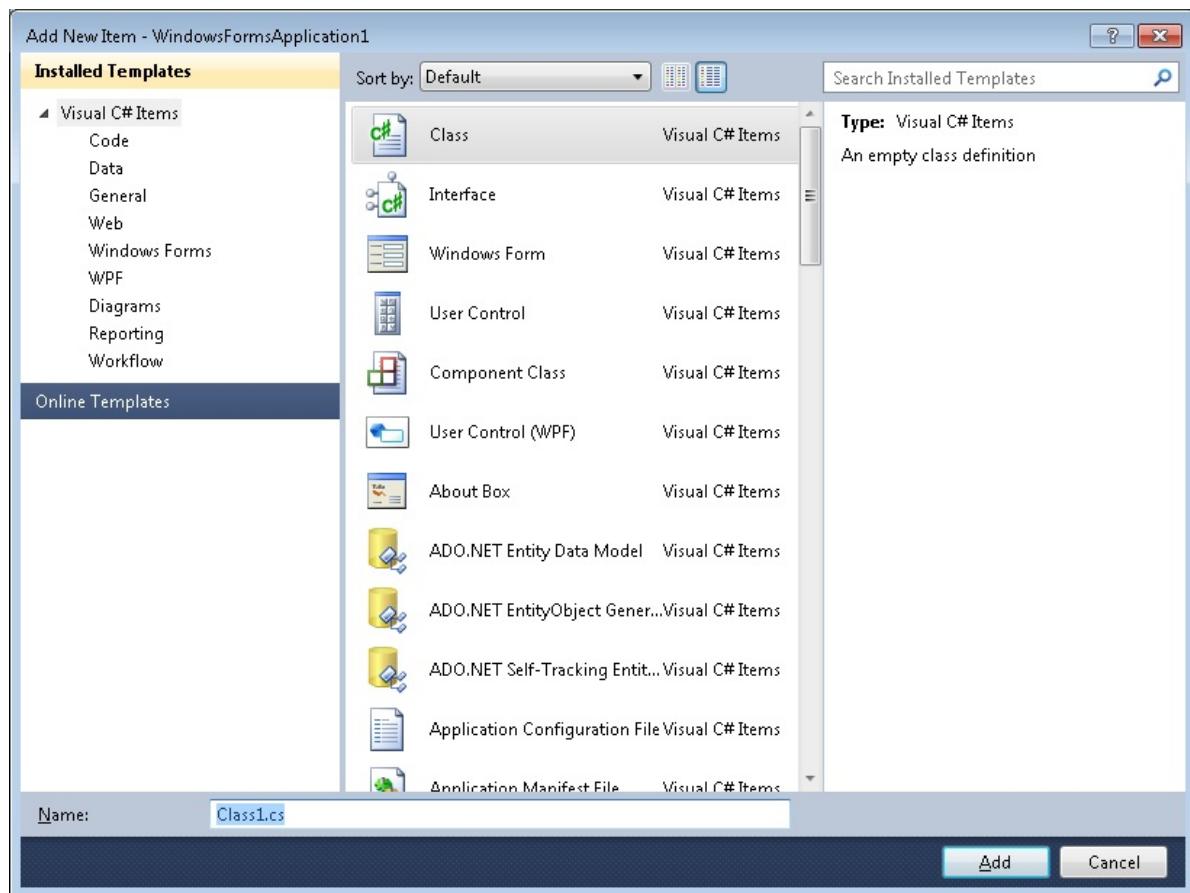
Also notice that no shortcuts are associated with the command. We can now add one. For this example, let's use Ctrl+M, Ctrl+7 as the shortcut key to be assigned.



For now, I won't get into the nuances of assigning shortcut keys, but you can get the details at vstipTool0063, [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), page 127. Assuming the key was assigned correctly, you can click OK and then press Ctrl+M, Ctrl+7 to see the Add New Item dialog box pop up, as shown in the following illustration.



Now you understand the power of commands. They can be quite useful, and after you assign a shortcut key, you can see it in the menu as well (if applicable).



## 03.20 Understanding Commands: Aliases

<b>DEFAULT</b>	Ctrl+Alt+A
<b>VISUAL BASIC 6</b>	Ctrl+Alt+A
<b>VISUAL C# 2005</b>	Ctrl+Alt+A; Ctrl+W, A; Ctrl+W, Ctrl+A
<b>VISUAL C++ 2</b>	Ctrl+Alt+A
<b>VISUAL C++ 6</b>	Ctrl+Alt+A
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+A
<b>WINDOWS</b>	Alt,V, E, C
<b>MENU</b>	View   Other Windows   Command Window
<b>COMMAND</b>	View.CommandWindow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0068

It is sometimes cumbersome to type in a full command. You can use aliases to quickly use a command without having to type in the full command syntax. If you want a list of the current aliases, just open the Command Window (Ctrl+Alt+A) and type **alias**.

```
Command Window
>alias
alias ? Debug.Print
alias ?? Debug.QuickWatch
alias AddProj File.AddNewProject
alias alias Tools.Alias
alias autos Debug.Autos
alias bl Debug.Breakpoints
alias bp Debug.ToggleBreakpoint
alias callstack Debug.CallStack
alias ClearBook Edit.ClearBookmarks
alias close File.Close
alias CloseAll Window.CloseAllDocuments
```

By the way, you can clear the Command Window out at any time by typing **cls**, just in case you get a lot of clutter in the window.

Let's take a simple alias as an example. How about the “Debug.ToggleBreakpoint” command? Notice in the preceding list that the alias is “bp”. Let's find a line of code.

Now we can go to the Command Window and type **bp**, as shown in the following illustration.



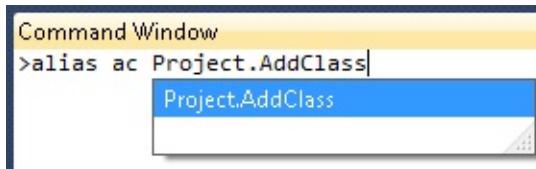
```
Command Window
>bp
>
```

It puts a breakpoint on the line.



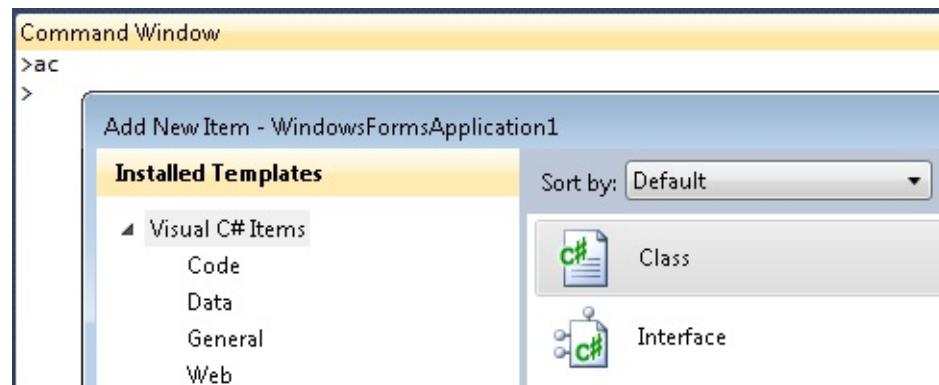
## Create a New Alias

In addition to the aliases that are already there, you can create new ones. Let's create one for the Project.AddClass command. Simply type **alias [alias to use] [command]**. In our case, let's put in **alias ac Project.AddClass**, as shown in the following illustration.



```
Command Window
>alias ac Project.AddClass
Project.AddClass
```

Anytime we want to add a class, we can type the command alias **ac** and get the Add New Item dialog box.



## Viewing Assigned Aliases

To show what command an alias is assigned to, just type **alias [alias]**. So, in this case, I would put in **alias ac** to see what command "ac" is bound to.

```
Command Window
>alias ac
alias ac Project.AddClass
>
```

## Delete an Alias

To get rid of an alias, type **alias [alias] /d[elete]**. To get rid of the alias we just made, we would type **alias ac /d**.

```
Command Window
>alias ac /d
>
```

You can confirm the alias is gone by typing **alias ac**. You should see the result shown in the following illustration.

```
Command Window
>alias ac /d
>alias ac
No alias 'ac' is defined
>
```

## 03.21 Understanding Commands: Arguments and Switches

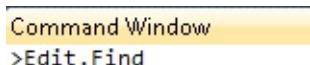
<b>DEFAULT</b>	Ctrl+Alt+A
<b>VISUAL BASIC 6</b>	Ctrl+Alt+A
<b>VISUAL C# 2005</b>	Ctrl+Alt+A; Ctrl+W, A; Ctrl+W, Ctrl+A
<b>VISUAL C++ 2</b>	Ctrl+Alt+A
<b>VISUAL C++ 6</b>	Ctrl+Alt+A
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+A
<b>WINDOWS</b>	Alt,V, E, C
<b>MENU</b>	View   Other Windows   Command Window
<b>COMMAND</b>	View.CommandWindow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0069

Some commands take arguments and switches so that you can quickly execute them without having to deal with user interface elements. You can get a list of commands that take arguments by going to the MSDN Documentation article entitled “Visual Studio Commands with Arguments,” at <http://msdn.microsoft.com/en-us/library/c338aexd.aspx>.

The best way to learn is by doing, so let’s use the Edit.Find command. If you want to know more about what Find can do, take a look at vstipFind0007, [05.02 Using Quick Find](#), on page 172.

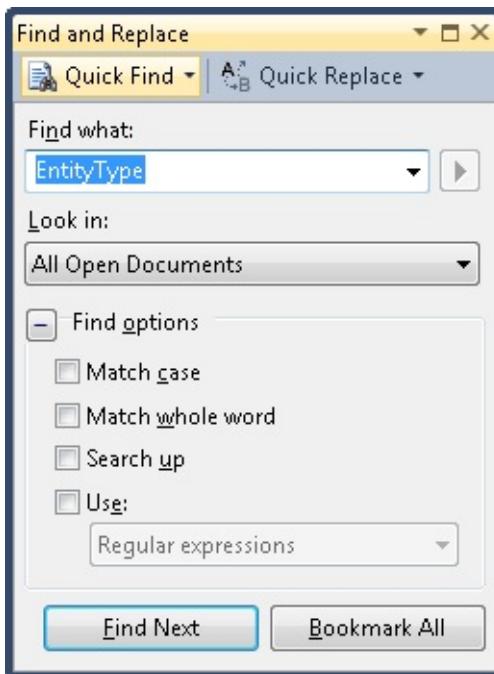
### Basic Use

First, open up the Command Window (Ctrl+Alt+A) and run the command without any arguments, as shown in the following illustration.



The preceding command opens the Find And Replace dialog box, shown in the

following illustration.



## Arguments and Switches

According to the “Find Command” documentation at <http://msdn.microsoft.com/en-us/library/295dhke9.aspx>, the Edit.Find command takes one argument and 12 possible switches. The general syntax for the command is as follows:

```
Edit.Find findwhat [/case] [/doc | /proc | /open | /sel] [/markall] [/options]
[/reset] [/up] [/wild | /regex] [/word]
```

I’ll resist the urge to copy and paste from the documentation here and just focus on the items we are going to use.

### Argument

- **findwhat**—Required. The text to match.

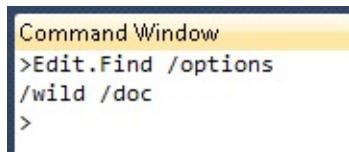
### Switches

- **/doc** or **/d**—Optional. Searches the current document only. You can use only one of the available search scopes: /doc, /proc (procedure), /open (all open documents), or /sel (current selection).
- **/markall** or **/m**—Optional. Places a [bookmark] on each line that contains a search match within the current document.

- **/wild** or **/l**—Optional. Uses predefined special characters in the `findwhat` argument as notations to represent a character or sequence of characters.

## List Current Options

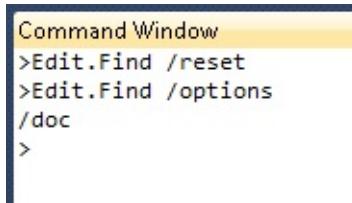
You can list out the current options that are set for the `Edit.Find` command by typing **Edit.Find /options**, as shown in the following illustration.



```
Command Window
>Edit.Find /options
/wild /doc
>
```

## Reset Options

You can reset the options to the default values by typing **Edit.Find /reset**.



```
Command Window
>Edit.Find /reset
>Edit.Find /options
/doc
>
```

## Using the Arguments and Switches

Let's put this command to the test. We want to bypass the Quick Find dialog box and just find things. We will use the following command:

```
Edit.Find *c* /wild /doc /markall
```

This command finds any line in the current document (/doc) that has the letter “c” anywhere in it, using wildcards (/wild) and placing a bookmark (/markall) on each line.

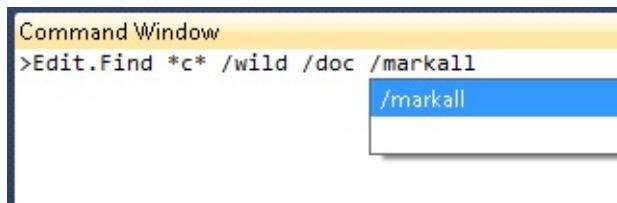
The following illustration shows the code we are going to use before we run the command.

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9
10 namespace WindowsFormsApplication1
11 {
12     public partial class Form1 : Form
13     {
14         public Form1()
15         {
16             InitializeComponent();
17         }
18     }
19 }
20

```

We run our command.



The result is shown in the following illustration.

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9
10 namespace WindowsFormsApplication1
11 {
12     public partial class Form1 : Form
13     {
14         public Form1()
15         {
16             InitializeComponent();
17         }
18     }
19 }
20

```

And now we have a working command that bypasses the Quick Find dialog box and just finds things.

## Make an Alias

What if we want to use this all the time? We can make an alias out of the command. In this case, we type **alias findc Edit.Find \*c\* /wild /doc /markall**, as shown in the following illustration.

```
Command Window
>alias findc Edit.Find *c* /wild /doc /markall
```

More information on aliases can be found in vstipTool0068 ([03.20 Understanding Commands: Aliases](#), page 113). You can double-check the alias assignment by typing **alias findc**.

```
Command Window
>alias findc
alias findc Edit.Find *c* /wild /doc /markall
>
```

From now on, you just type **findc** in the Command Window, and it performs the predefined search.

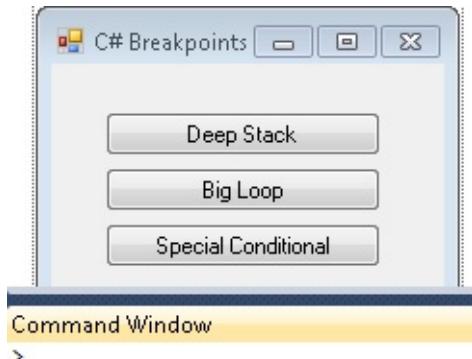
## 03.22 Testing a Command

<b>DEFAULT</b>	Ctrl+Alt+A
<b>VISUAL BASIC 6</b>	Ctrl+Alt+A
<b>VISUAL C# 2005</b>	Ctrl+Alt+A; Ctrl+W, A; Ctrl+W, Ctrl+A
<b>VISUAL C++ 2</b>	Ctrl+Alt+A
<b>VISUAL C++ 6</b>	Ctrl+Alt+A
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+A
<b>WINDOWS</b>	Alt,V, E, C
<b>MENU</b>	View   Other Windows   Command Window
<b>COMMAND</b>	View.CommandWindow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0065

Throughout these tips, I include the command when available. But you might be wondering how to test a command to see how it works. Let's take a look at one quick way.

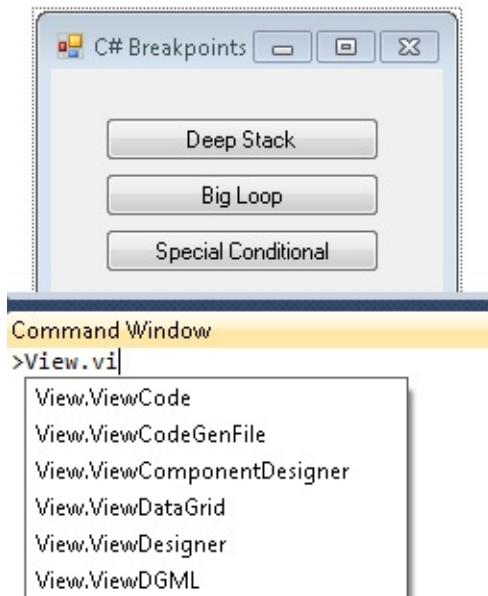
First, figure out what command you want to test. In our case, let's test “View.ViewCode”.

Press Ctrl+Alt+A to bring up the Command Window.

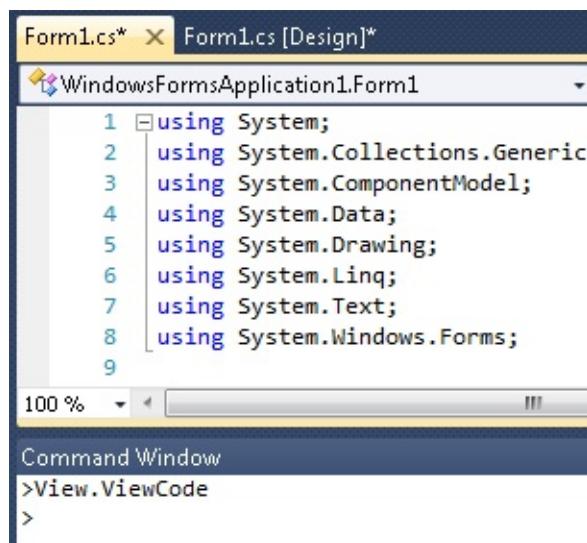


Because we're testing “ViewCode”, we have a Design window open so that it can switch to the code. Now start typing the command we want to test (case doesn't

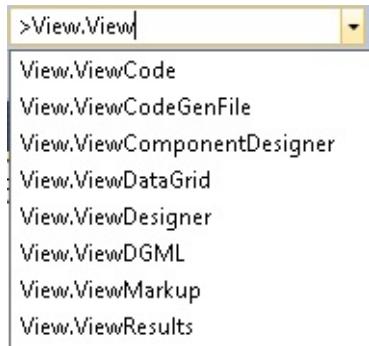
matter).



As you can see in the preceding illustration, we have IntelliSense helping us with the command. Now we just press Enter to see the command in action.



Now you have tested a command. Feel free to try various commands to see what they do, but make sure to set up the context as we did in this example to ensure that the command works properly. Some commands can't be run from the Command Window and might require that you assign a shortcut key or use the Find Combo box.



Refer to vtipTool0070 ([03.23 Understanding Commands: Running Commands](#), on the next page) for more information.

## 03.23 Understanding Commands: Running Commands

<b>DEFAULT</b>	Ctrl+Alt+A (command window); Ctrl+Alt+I (immediate window); Ctrl+/ (find combo box with command symbol)
<b>VISUAL BASIC 6</b>	Ctrl+Alt+A (command window); Ctrl+Alt+I; Ctrl+G (immediate window); (no shortcut for find combo box)
<b>VISUAL C# 2005</b>	Ctrl+Alt+A; Ctrl+W, A; Ctrl+W, Ctrl+A (command window) Ctrl+Alt+I; Ctrl+D, I; Ctrl+D, Ctrl+I (immediate window) Ctrl+/ (find combo box)
<b>VISUAL C++ 2</b>	Ctrl+Alt+A (command window); Ctrl+Alt+I (immediate window); Ctrl+/ (find combo box with command symbol)
<b>VISUAL C++ 6</b>	Ctrl+Alt+A (command window); Ctrl+Alt+I (immediate window); Ctrl+/ (find combo box with command symbol)
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+A (command window); Ctrl+Alt+I (immediate window); Ctrl+/ (find combo box with command symbol)
<b>WINDOWS</b>	Alt,V, E, C
<b>MENU</b>	View   Other Windows   Command Window; Debug   Windows   Immediate Window
<b>COMMAND</b>	View.CommandWindow; Debug.Immediate; Edit.GoToFindCombo; Tools.GoToCommandLine
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0070

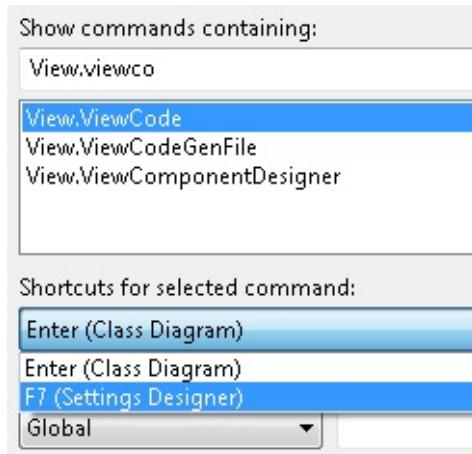
Whenever you work with commands, you have four main ways you can run them. For example, not all commands will run from the Command Window, so it is a good idea to familiarize yourself with the other options. Let's take a look at each way.

### Shortcuts

The easiest way to run a command is when a shortcut is attached to it. For example, View.Code has a couple of shortcut keys attached to it.

#### NOTE

For more information about shortcut keys, see vstipTool0061 (03.24 Find Keyboard Shortcuts, on the next page).



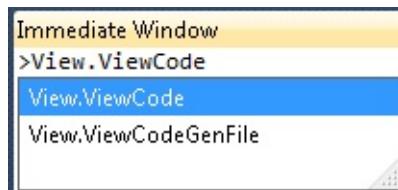
## Command Window

The Command Window (Ctrl+Alt+A) is specifically designed to run commands. Just type in the command, and press Enter.



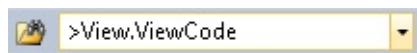
## Immediate Window

You can run many commands from the Immediate Window (Ctrl+Alt+I) by typing a greater-than sign (>). Then type any command and press Enter.



## Find Combo Box

A little-known feature enables you to run commands from the Find Combo Box (Ctrl+D) on the standard toolbar. Just type a greater-than sign (>), then type any command, and press Enter.



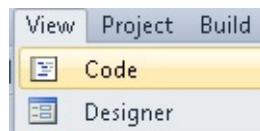
For most language settings, you can bypass the typing of the greater-than sign by using Ctrl+Forward Slash (/), which takes you to the Find Combo Box and

automatically inserts the sign for you.

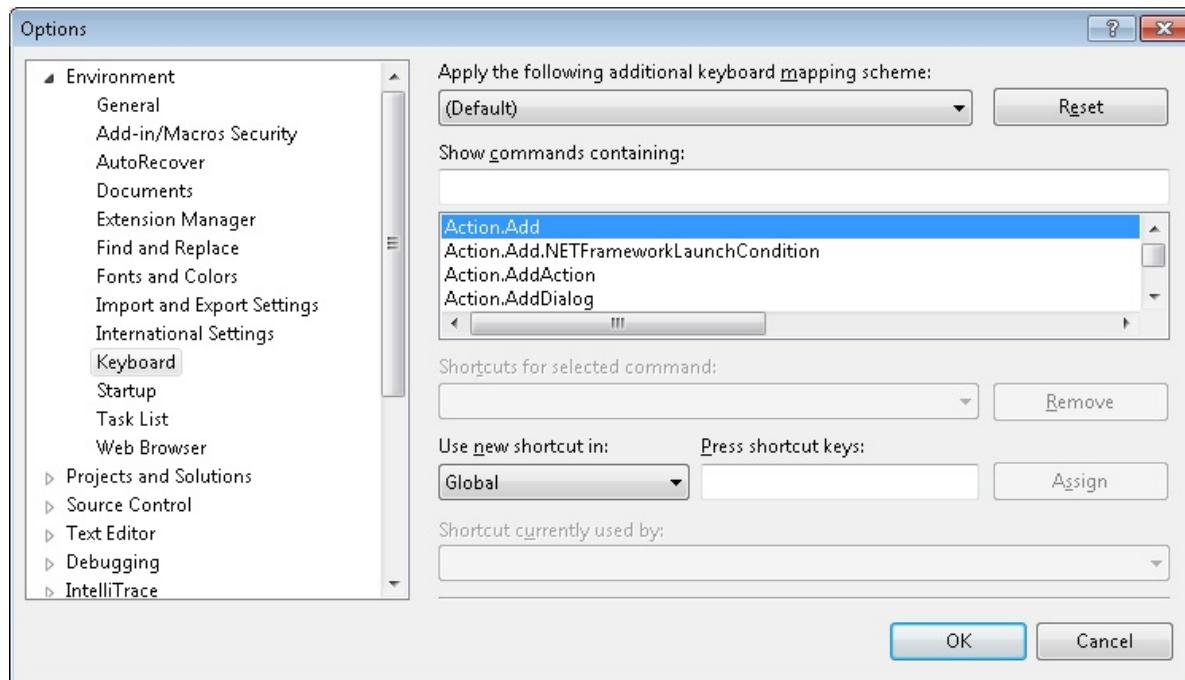
## 03.24 Find Keyboard Shortcuts

WINDOWS	Alt,T, O
MENU	Tools   Options   Environment   Keyboard
COMMAND	Tools.CustomizeKeyboard
VERSIONS	2005, 2008, 2010
CODE	vstipTool0061

Ever just want to see the keyboard shortcuts available in Visual Studio? Let's say, for example, that you want to see if View | Code has any keyboard shortcuts. A quick look at the menu doesn't reveal anything.

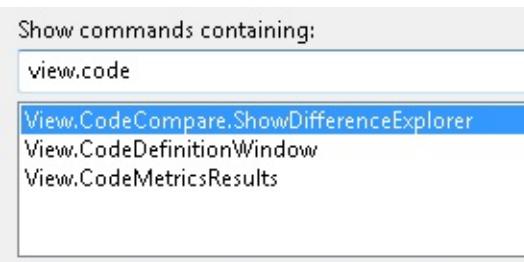


But that's not the end of the story. If we go to Tools | Options | Environment | Keyboard, we get the following dialog box.

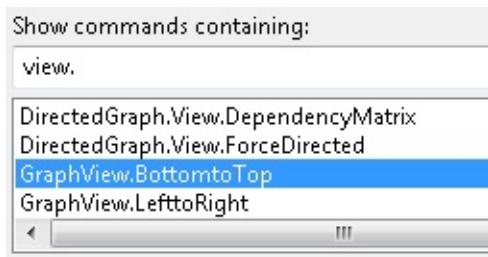


Pretty much everything you do in Visual Studio has a command that runs to execute that action. In our case, we know that View | Code is the path to the command we want, so let's start by trying to see whether we have a "View.Code" command.

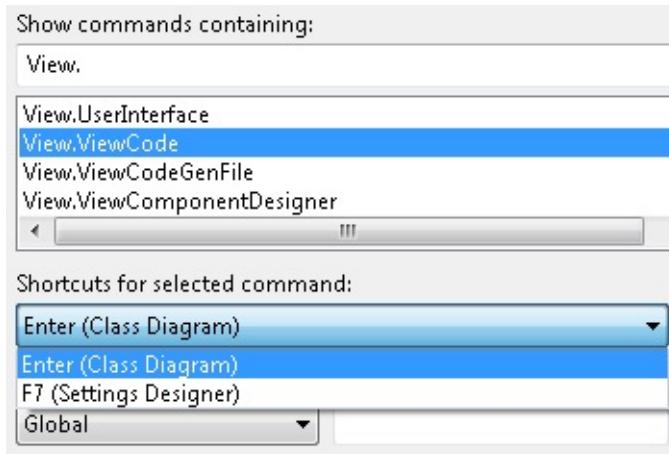
Notice that commands use dot notation between items:



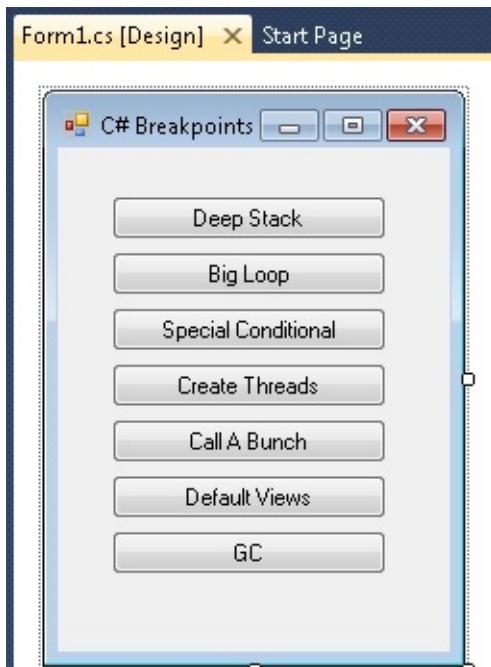
Sadly, what we want isn't there. OK, so either it isn't there or it's called something else. We still know it's off the View menu, so let's type in **View.** and browse to see whether anything pops up.



If we scroll down far enough, we actually find an entry for "View.ViewCode". If we look under Shortcuts For Selected Command, we see a couple of shortcut entries, as shown in the following illustration.



It looks like pressing F7 in the Designer does the trick, so let's try it.



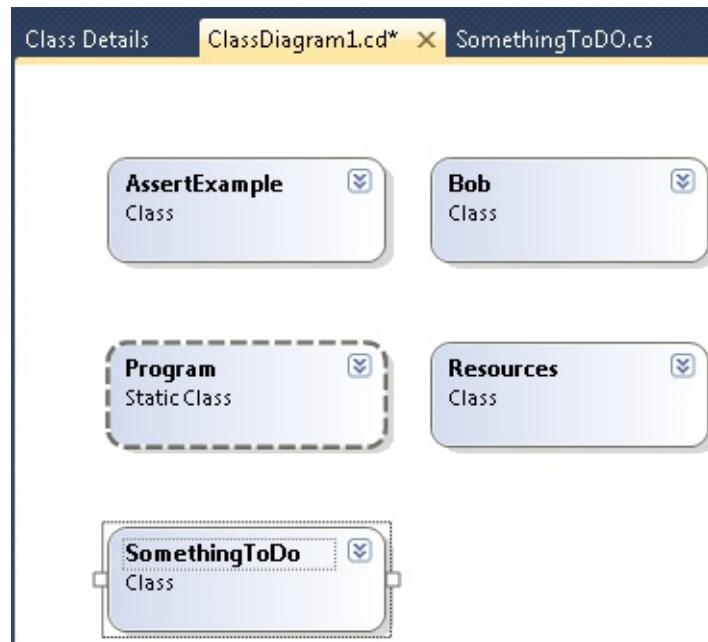
Go to the Designer and press F7.

A screenshot of the Windows Forms Designer in Visual Studio. The title bar shows 'Form1.cs [Design]'. Below it is a code editor window titled 'WindowsFormsApplication1.Form1'. The code in the editor is:

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
```

As we can see in the preceding illustration, we have discovered the shortcut key for viewing code in the Designer.

There is also another entry for the Class Diagram that uses Enter to show us code. Let's open up a Class Diagram and select a class.



Then press Enter.

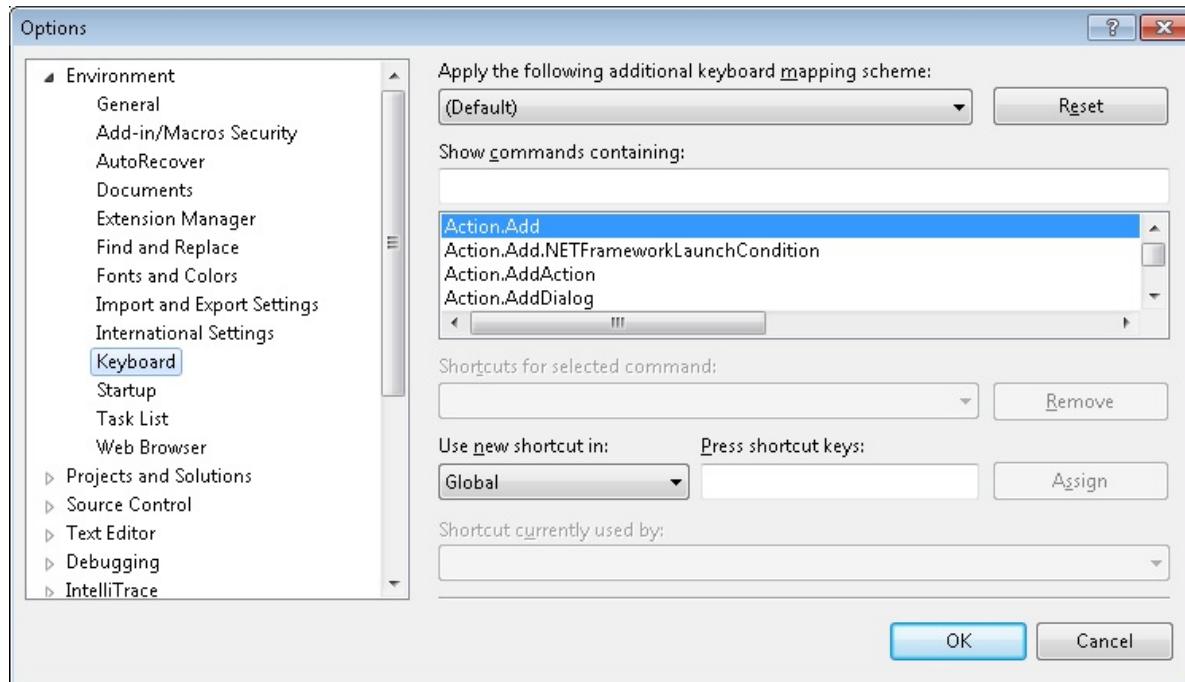
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace WindowsFormsApplication1
7  {
8      class SomethingToDo
9      {
```

It takes us to the code as well. So now you know how to find shortcut keys (if they exist) for a command.

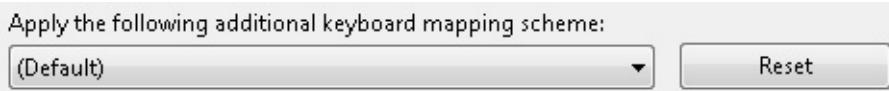
## 03.25 Keyboard Shortcuts: Additional Mapping Schemes

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Keyboard
<b>COMMAND</b>	Tools.CustomizeKeyboard
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0062

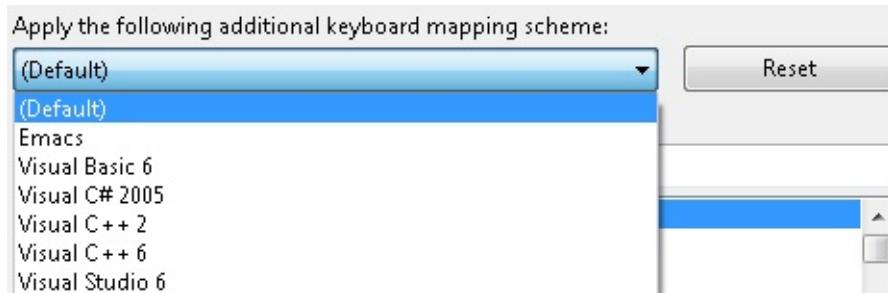
One of the most important places you can go in Visual Studio is the Tools | Options | Environment | Keyboard area, shown in the following illustration.



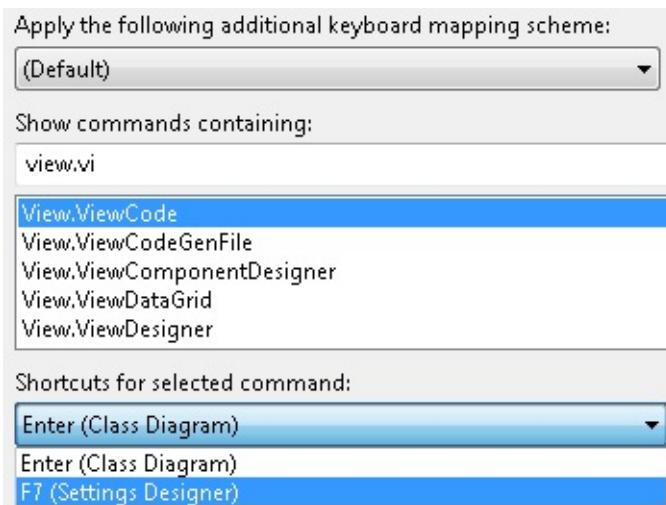
In vstipTool0061, [03.24 Find Keyboard Shortcuts](#), page 122, we looked at how to find shortcut keys for given commands. Now let's focus on the Apply The Following Additional Keyboard Mapping Scheme drop-down list.



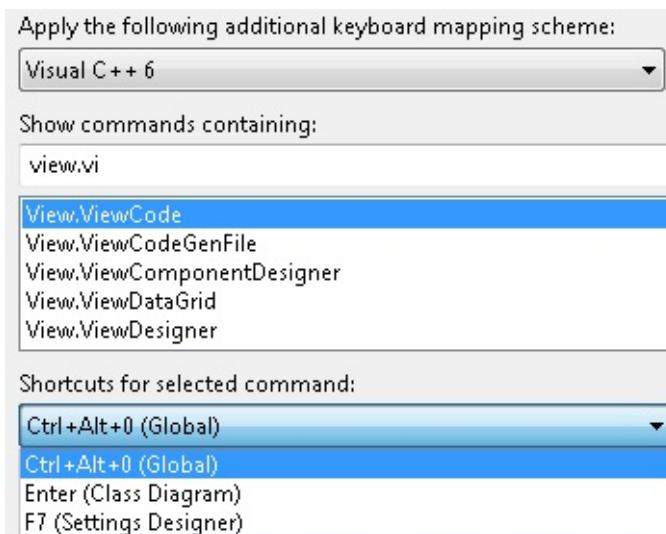
This list allows you to apply additional keyboard shortcuts that were common in certain previous versions.



Let's take an example. If we have the default mapping scheme and want to see the shortcuts for View.ViewCode, we see the options presented in the following illustration.



However, if we add an additional mapping scheme (Visual C++ 6 in this example), this gets the following result:



Notice that we have a new shortcut that wasn't there before. This is how adding additional mapping schemes work. If you don't want these additional keys, just set

Apply The Following Additional Keyboard Mapping Scheme to Default (or General, if it is available in the list).

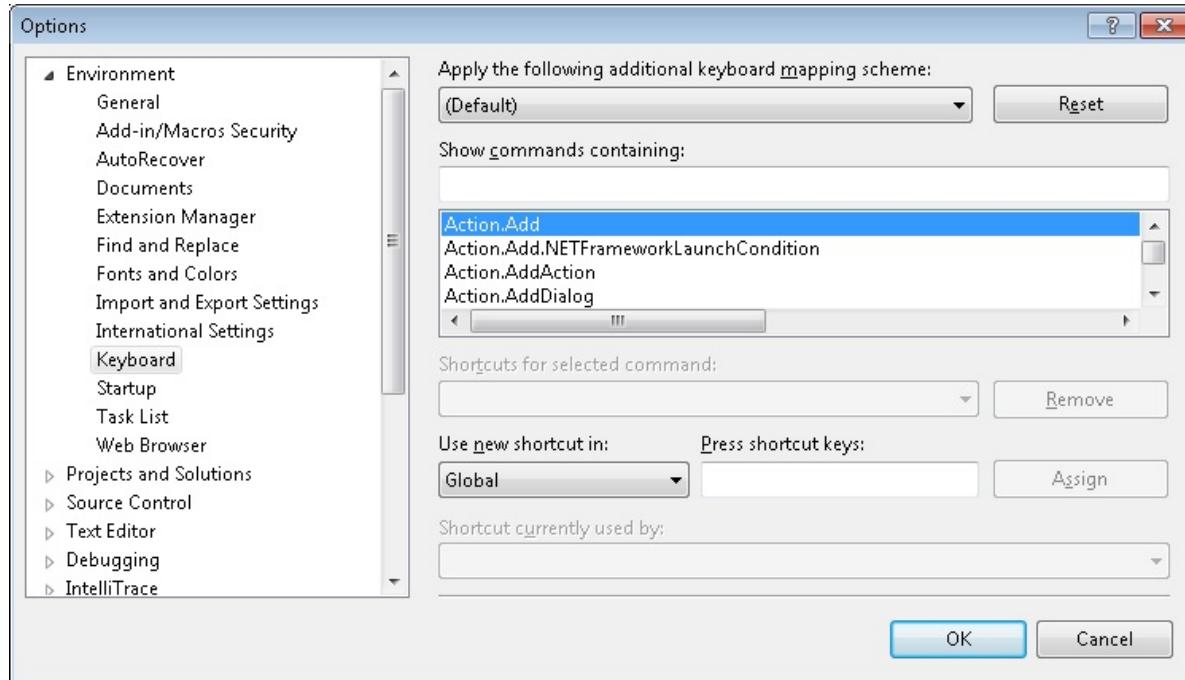
Apply the following additional keyboard mapping scheme:

(Default)	▼
-----------	---

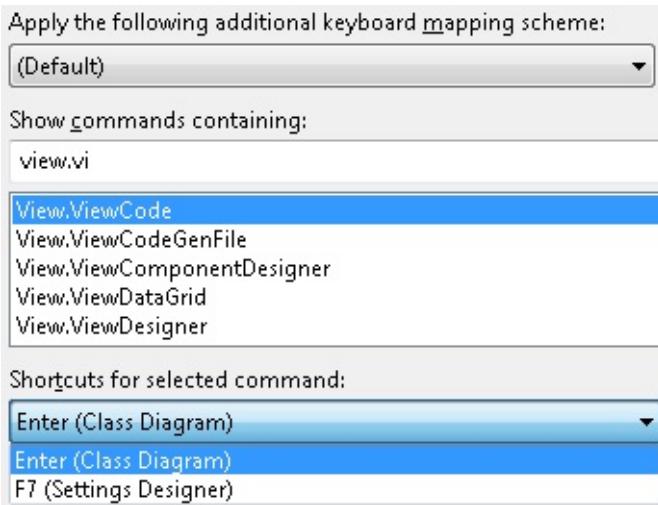
## 03.26 Keyboard Shortcuts: Creating New Shortcuts

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Keyboard
<b>COMMAND</b>	Tools.CustomizeKeyboard
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0063

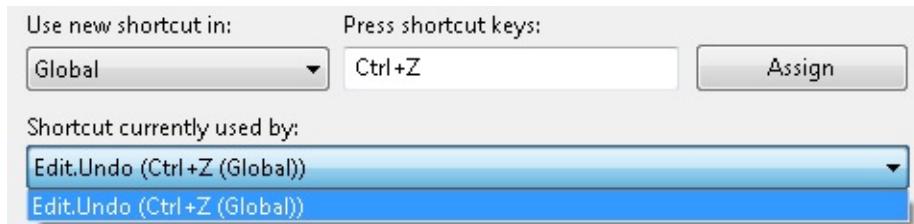
Creating keyboard shortcuts is easy. Let's walk through an example to show you how. First, go to Tools | Options | Environment | Keyboard.



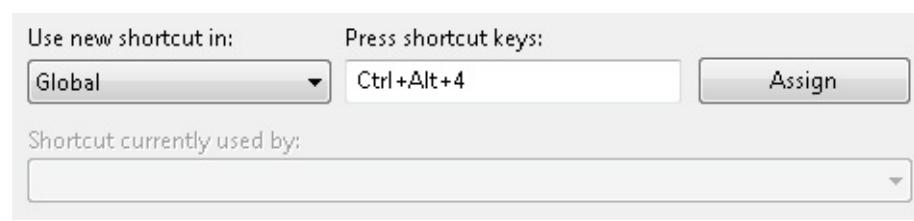
Now let's assume we want to modify the "View.ViewCode" command to include a new shortcut. Notice the existing shortcuts (assuming no additional keyboard mapping schemes).



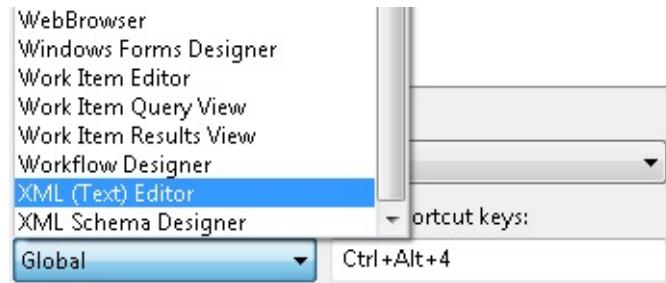
We want to add Ctrl+Z as the shortcut, so click in the Press Shortcut Keys area and press Ctrl+Z.



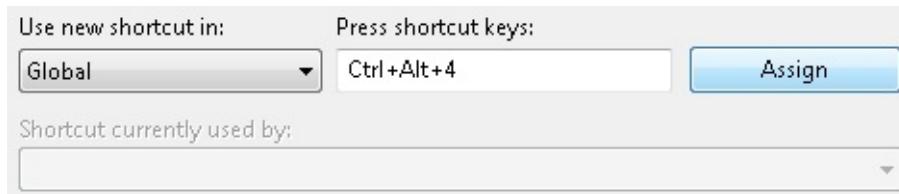
There is a problem. Looking down at the Shortcut Currently Used By area, we see that Ctrl+Z is already mapped to the Edit.Undo command. Because that is an important key combination to us, let's try a new set of shortcut keys. How about Ctrl+Alt+4? Just use backspace to get rid of the current entry, and press Ctrl+Alt+4.



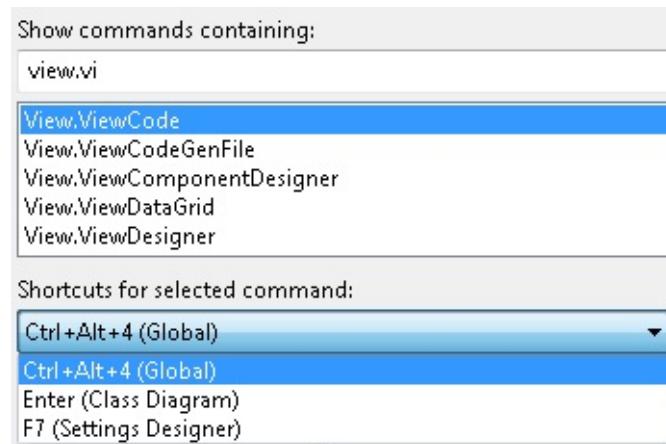
Perfect. It isn't being used by anything currently—but we aren't done yet. We have to decide what scope we want this shortcut to be available in. Notice the drop-down to the left of the new shortcut under Use New Shortcut In? The default scope is global, so you can use it at any time. However, you can narrow the scope down to a specific area.



For example, if we wanted to have this shortcut available only when we're editing XML, we would change Global to XML (Text) Editor. For now, let's keep the Global setting.



After we've decided on the scope and the shortcut keys, all we have to do is click Assign, as shown in the preceding illustration, to make the shortcut available.

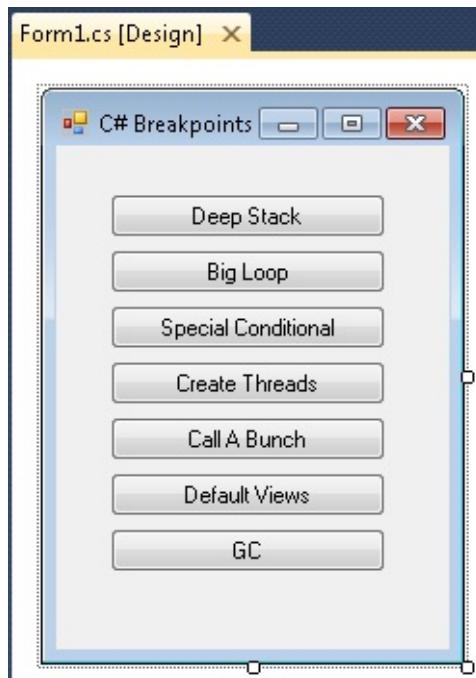


If you mess up here, you can choose the shortcut key and click Remove to start over.

#### WARNING

You can remove *any* shortcut, so be careful to remove only the shortcuts you actually want to eliminate.

Click OK, and let's go test our new shortcut. Go to Design View in any project.



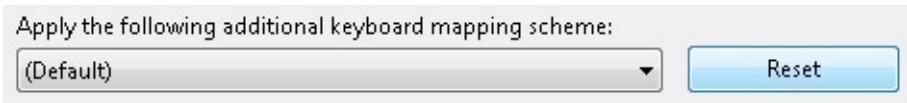
Press Ctrl+Alt+4 to see the following result:

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9
10 using WindowsFormsApplication1;
11 {
12     public partial class Form1 : Form
13     {
14         public Form1()
15         {
16             InitializeComponent();
17         }
18     }
}
```

It should take you to the code. You now know how to map new keyboard shortcuts.

## Reset

If you make a mistake with your shortcut keys, you can always click the Reset button in Tools | Options | Environment | Keyboard.

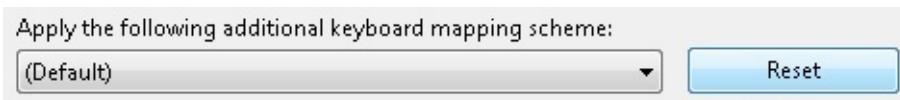


Don't take this option lightly. Refer to vstipTool0064 ([03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#), page 131).

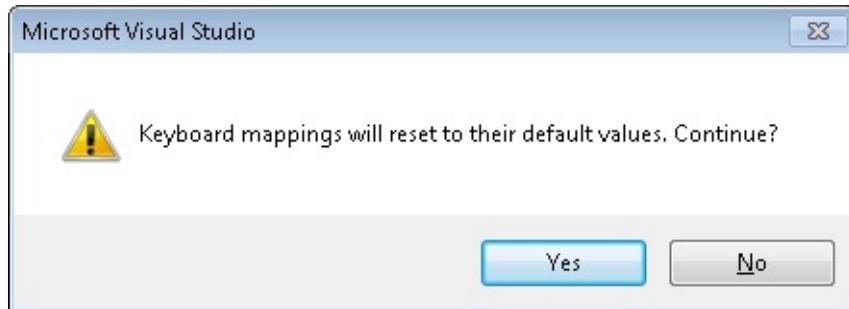
## 03.27 Keyboard Shortcuts: Reset All Your Shortcuts

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Keyboard
<b>COMMAND</b>	Tools.CustomizeKeyboard
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0064

On rare occasions, you might lose track of all the custom shortcuts you have made, or maybe you just want to reset all your shortcuts back to their default settings. You can reset all your keyboard shortcuts by clicking the Reset button in Tools | Options | Environment | Keyboard.



When you click this button, you get the following warning message:

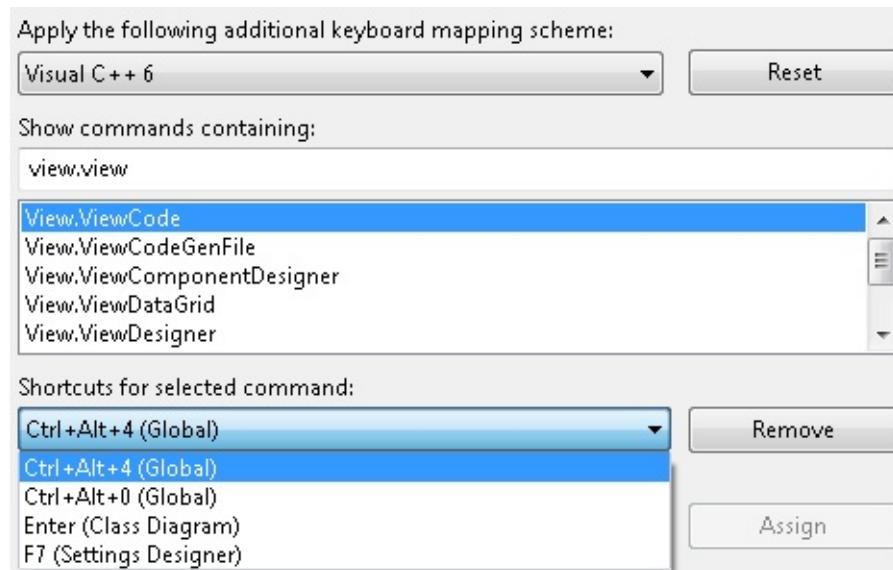


### NOTE

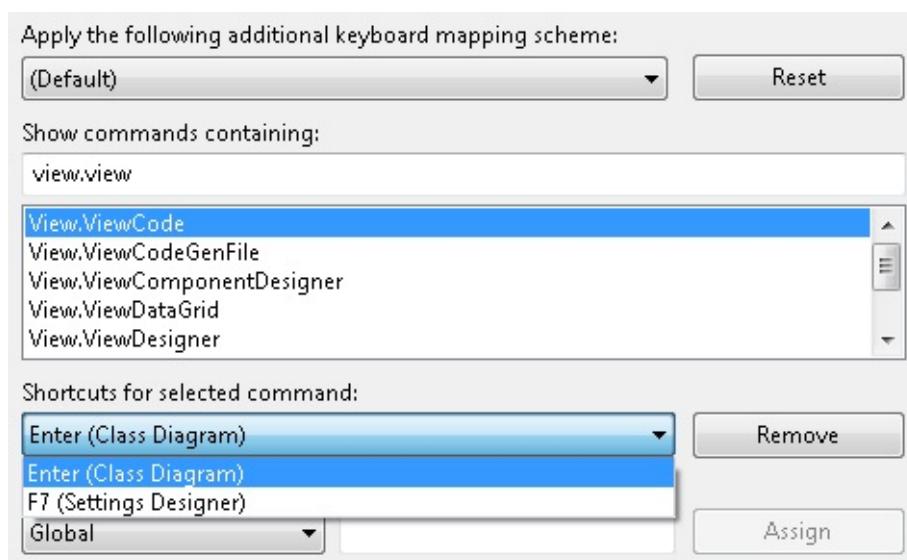
Take this warning seriously! It *resets all keyboard mappings to their default values* if you click Yes, so you should use this only if you are sure of the consequences.

The following before-and-after illustrations show what happens after you click Yes in the preceding Warning message.

Before:



After:



## 03.28 Understanding Commands: Logging Commands

<b>COMMAND</b>	log; Tools.LogCommandWindowOutput
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0071

When using commands, sometimes you want to keep a log of the ones you used. This is especially useful when you are experimenting with commands to see what iterations you went through. The syntax for logging is as follows:

```
log [filename] [/on]/[off] [/overwrite]
```

Or you can use the following:

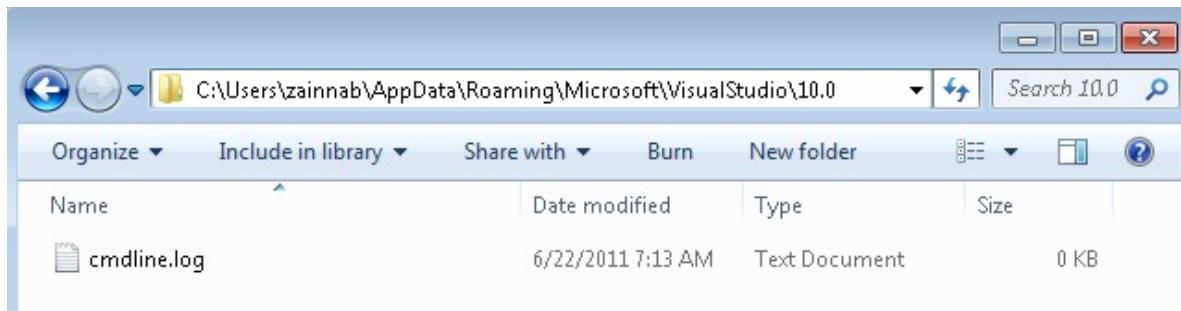
```
Tools.LogCommandWindowOutput [filename] [/on]/[off] [/overwrite]
```

## Arguments

The following sections describe what the preceding arguments do.

### Filename

It's highly recommended that you use a path and file name; otherwise, the default file name is cmdline.log and the log file is stored at C:\Users\<user>\AppData\Roaming\Microsoft\VisualStudio\<version>.



### /on /off

This argument turns logging on or off.

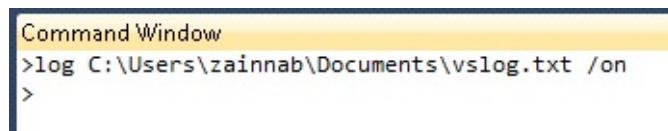
### /overwrite

By default, all logging operations append to your log file. The /overwrite switch changes this behavior and erases any previous commands from the log when a

new log session starts.

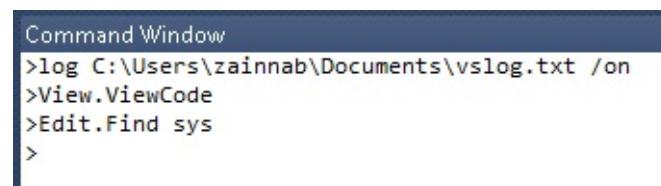
## Example

First, type the “log” command in the Command Window (Ctrl+Alt+A) and specify a file name by typing **log C:\Users\<user>\Documents\vslog.txt /on**, as shown in the following illustration.



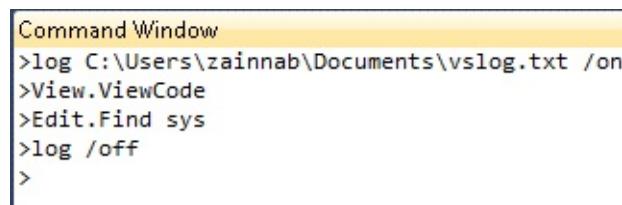
```
Command Window
>log C:\Users\zainnab\Documents\vslog.txt /on
>
```

Now type in a couple of commands. The specific commands don’t matter here, so feel free to substitute your own commands instead of using mine if you like. I used the commands View.ViewCode to get a code window, and I used Edit.Find sys to find some text.



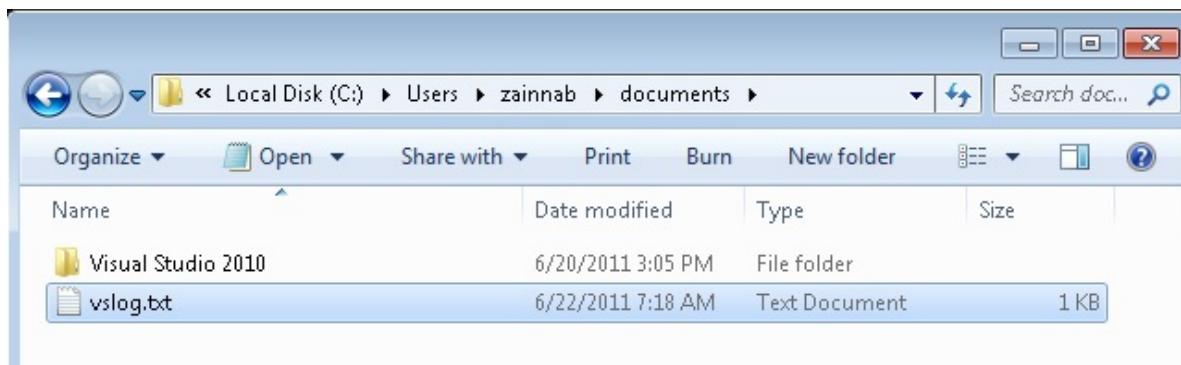
```
Command Window
>log C:\Users\zainnab\Documents\vslog.txt /on
>View.ViewCode
>Edit.Find sys
>
```

At this point, you’re done typing commands. Turn logging off by using the **log /off** command.

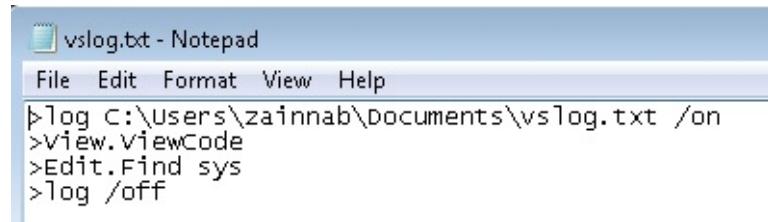


```
Command Window
>log C:\Users\zainnab\Documents\vslog.txt /on
>View.ViewCode
>Edit.Find sys
>log /off
>
```

Now browse to the My Documents folder to see the file.



If you open the file, you should see something similar to the following:



A screenshot of a Microsoft Notepad window titled "vslog.txt - Notepad". The window shows a list of commands entered into a command-line interface. The commands are:

```
>log C:\Users\zainnab\Documents\vslog.txt /on
>View.ViewCode
>Edit.Find sys
>log /off
```

As you can see, the file contains a running log of the commands you entered, including the last command to turn logging off.

## 03.29 Export Your Window Layouts

<b>WINDOWS</b>	Alt,T, I
<b>MENU</b>	Tools   Import and Export Settings
<b>COMMAND</b>	Tools.ImportandExportSettings
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0040

After you get your tool and document windows just the way you want them inside Visual Studio, you want to be able to get those settings back if anything goes wrong.

You can export just your window layouts by going to Tools | Import And Export Settings and choosing Export Selected Environment Settings, and then click Next.



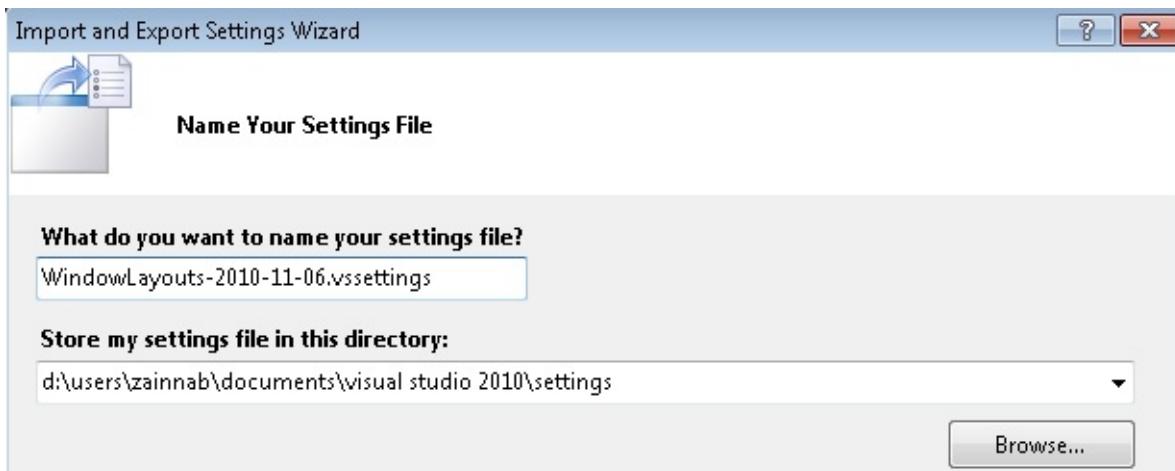
Now clear the All Settings check box.



Go to General Settings, select Window Layouts, and then click Next.



Give the .vssettings file a name and a location to be saved into, click Finish, and then click Close.



You now can import these settings (see vtipEnv0022, [01.08 Importing or Changing Your Environment Settings](#), in Appendix B [<http://go.microsoft.com/fwlink/?LinkId=223758>]) whenever you want to get your preferred layouts.

## 03.30 Stop the Toolbox from Auto-Populating from the Solution

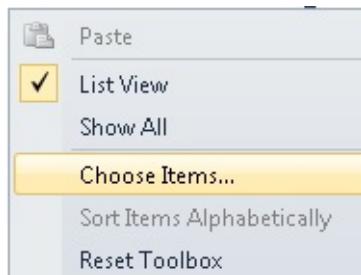
<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Windows Forms Designer
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0090

If you find that the Toolbox is taking a long time scanning a solution with a lot of projects in it, you can keep it from doing this.

Just go to Tools | Options | Windows Forms Designer, and set AutoToolboxPopulate to False.



To display custom items when AutoToolboxPopulate is set to False, you can select Choose Items from the Toolbox context menu and add the items manually to the Toolbox.

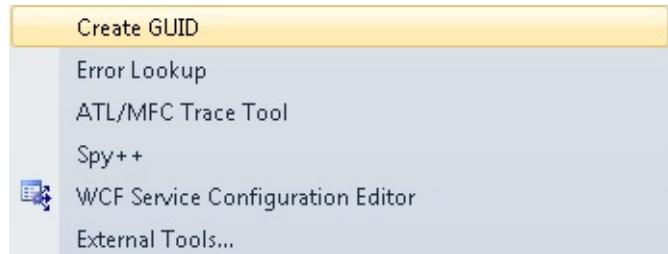


People often set this to False accidentally. If you find your controls are not automatically showing up in the Toolbox, setting this to True might solve the problem.

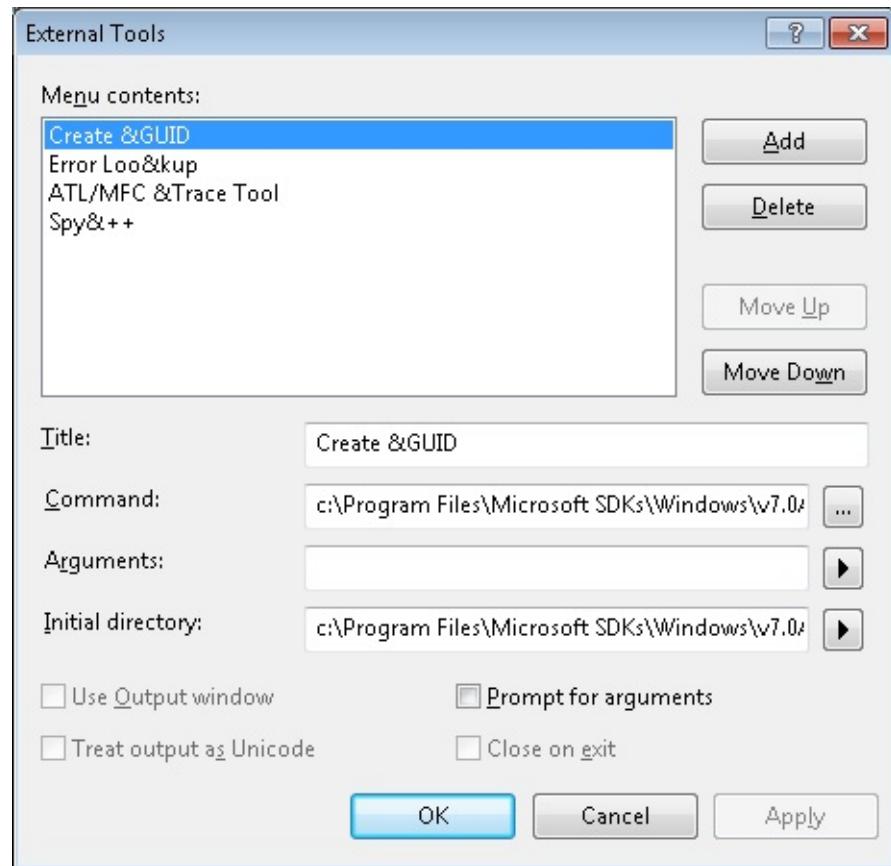
## 03.31 Using External Tools

<b>WINDOWS</b>	Alt,T, E
<b>MENU</b>	Tools   External Tools; Tools   [external tool of choice]
<b>COMMAND</b>	Tools.ExternalTools
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0059

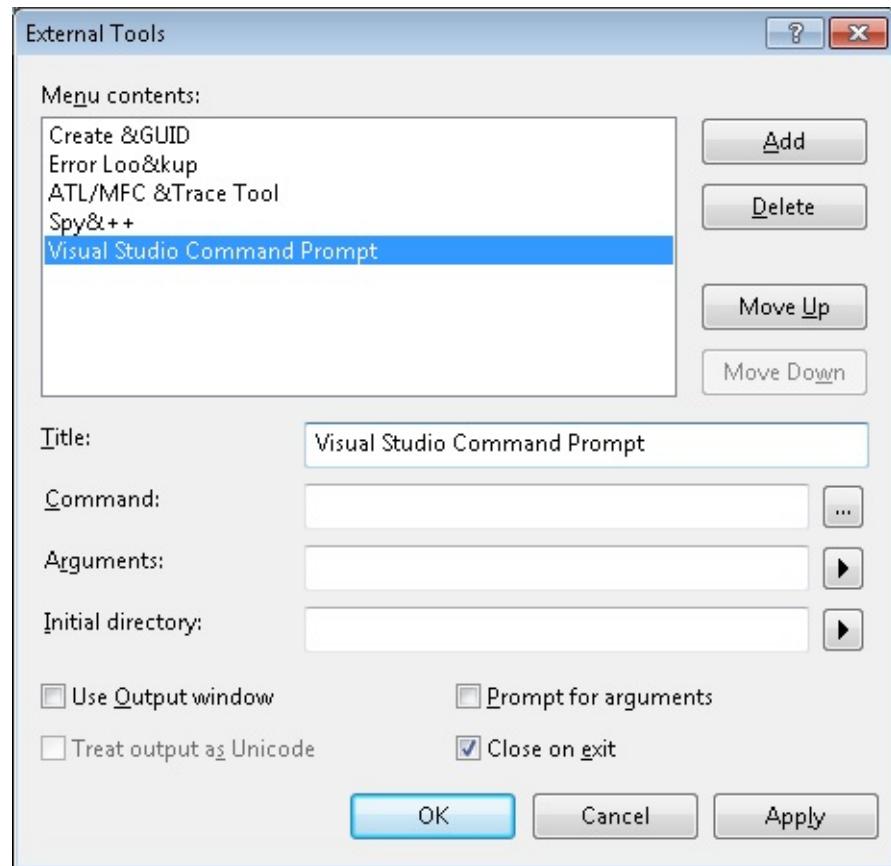
You can run external tools by going to the Tools menu and running your external tool of choice, as shown in the following illustration.



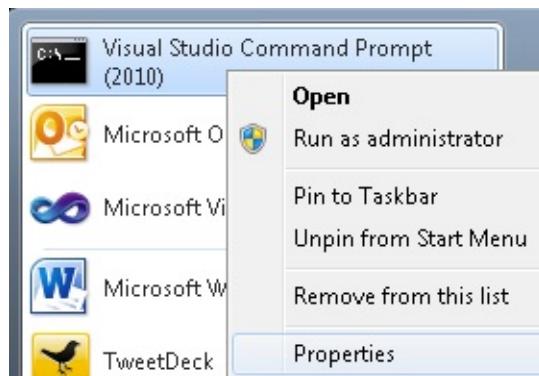
If you want to add additional external tools, you can go to Tools | External Tools.



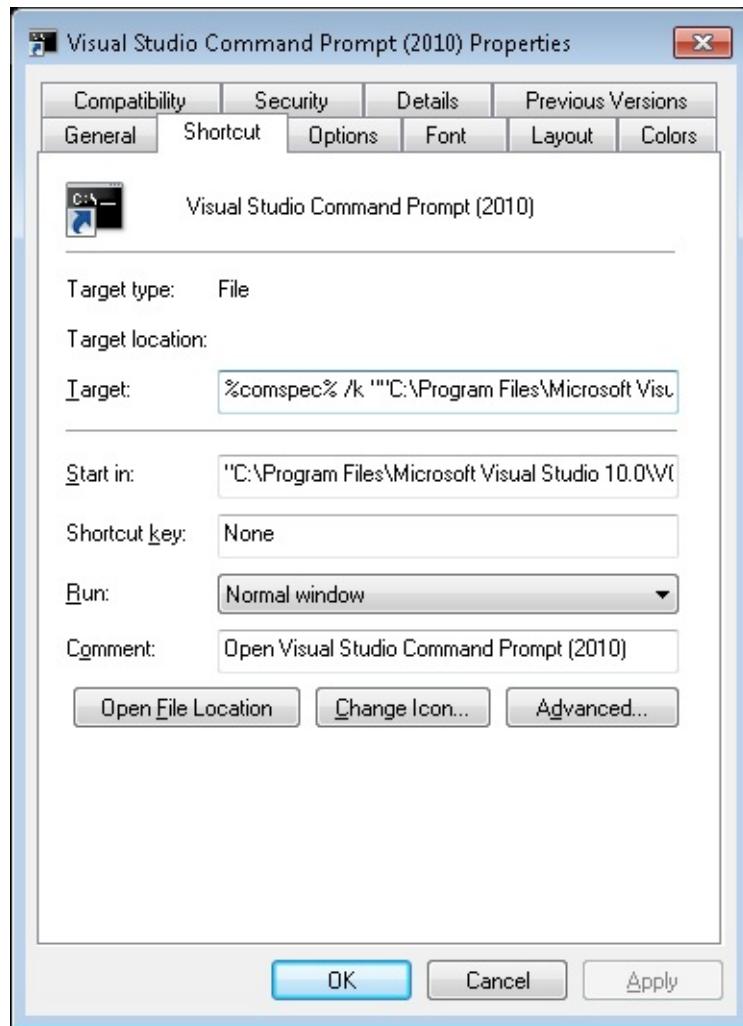
As an example, let's add the Visual Studio command prompt to our external tools list. First, click Add and put in a Title of **Visual Studio Command Prompt**.



In this case, the Visual Studio command prompt has an icon that we can use to get the information we need. Let's go to its properties.



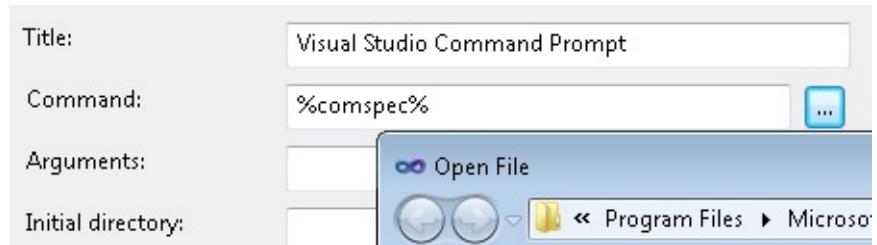
After you click Properties, you see the following dialog box.



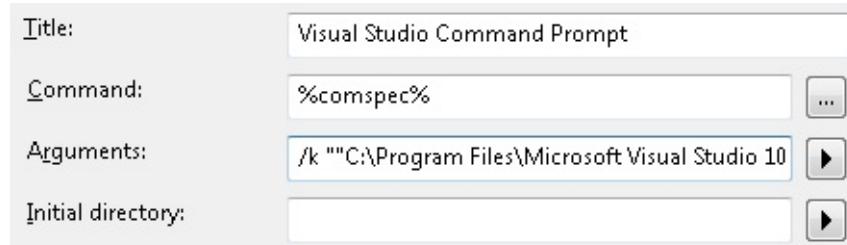
As you can see, the Target field points to %comspec%, with some arguments after it. The variable %comspec% points to the command prompt on the current Windows system and can be used on just about any version of Windows. Let's put that into the Command field for our purposes.

Title:	Visual Studio Command Prompt
Command:	%comspec%
Arguments:	
Initial directory:	

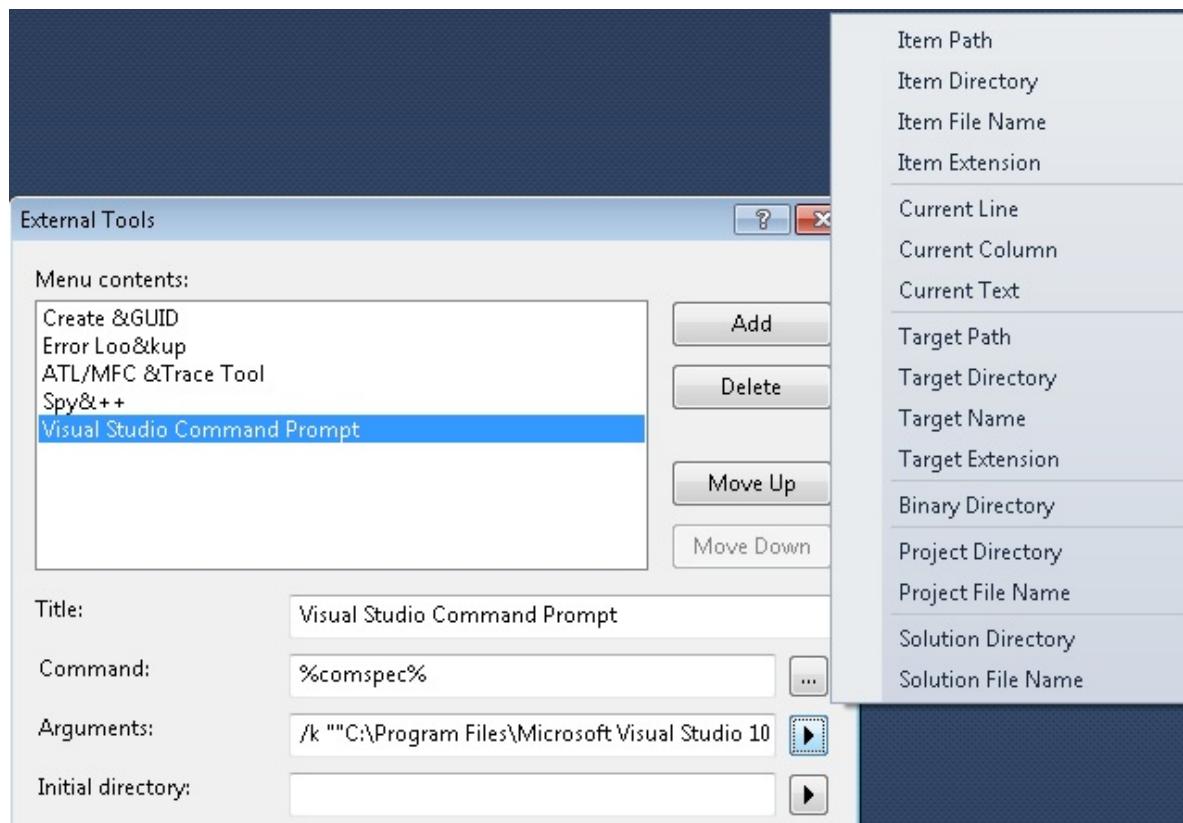
If you weren't using a system variable like this, you could type in the path to the command or use the ellipsis and browse.



For the Arguments field, again, let's just take from the Visual Studio command prompt properties and enter the path shown in the following illustration.

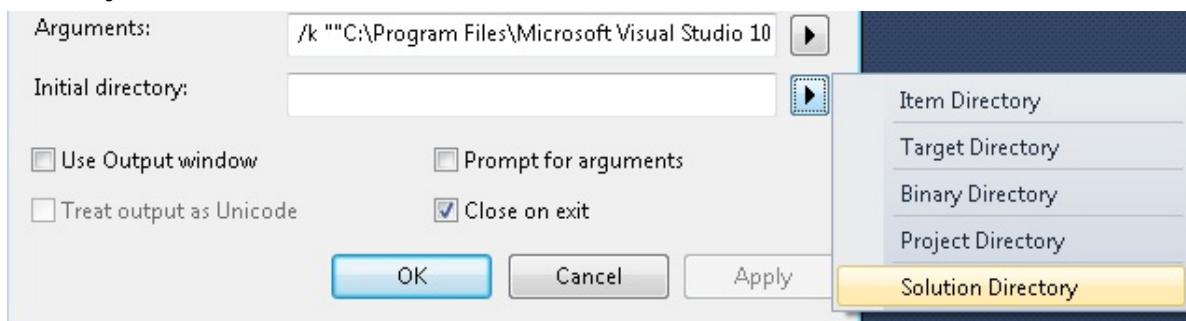


If you want to add items specific to Visual Studio, you can click the arrow to the right and get a large list of variables you can insert as arguments.



The Initial Directory lets you define where you want to start up. I don't have a preference here, so I will leave it blank, but I could type in a path or use one of the variables available to me by clicking on the arrow to the right of the Initial

Directory field.



Finally, you can choose from several options at the bottom.



## Use Output Window

Selecting this option runs the command and puts any output into the Output window. This is useful for commands that just return some data you want to look at. For our command prompt, this wouldn't be a good choice because we want to type in commands.

## Treat Output As Unicode

If your tool returns Unicode output, you would select this check box.

## Prompt For Arguments

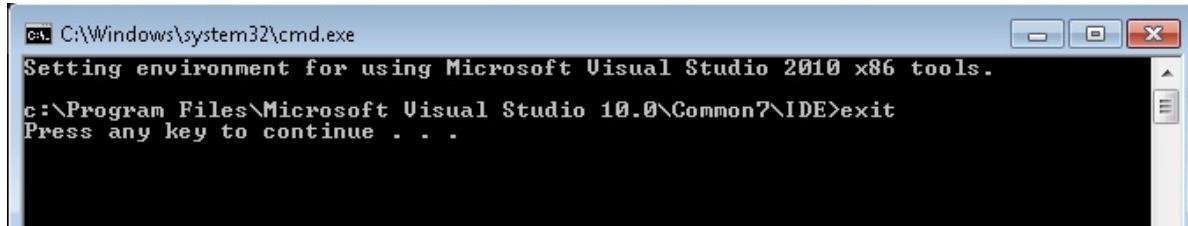
This option shows a dialog box that enables you to modify the arguments before the command is run or to put in completely new arguments.



## Close On Exit

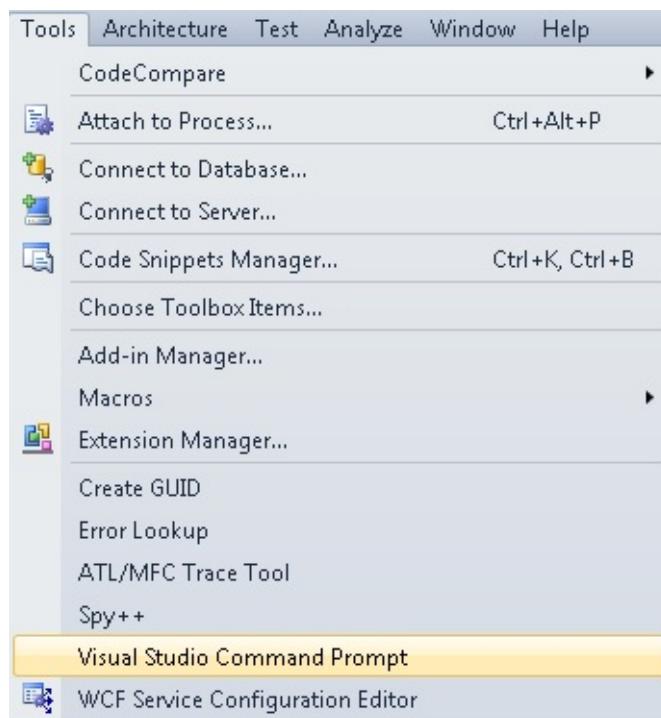
Selecting this option determines whether the window should close when the tool closes. In the case of our command prompt, with this option checked when we

type **exit** and press Enter, the window closes. However, if this option is *not* checked and we do the same thing, we get the result shown in the following illustration.



```
C:\Windows\system32\cmd.exe
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
c:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE>exit
Press any key to continue . . .
```

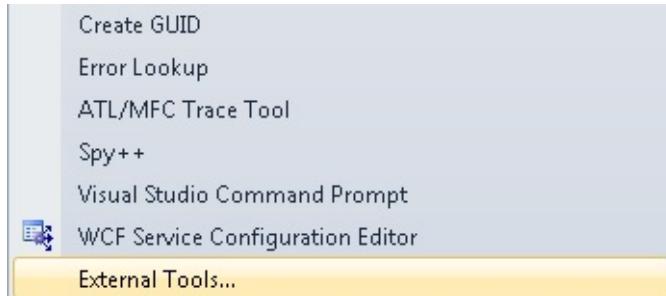
When you are all done, just click OK, and you now have your command showing up in the external tools list off the Tools menu.



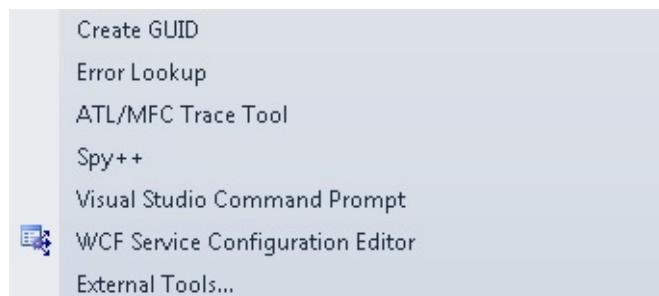
## 03.32 Create Keyboard Accelerators for External Tools

<b>WINDOWS</b>	Alt,T, E
<b>MENU</b>	Tools   External Tools
<b>COMMAND</b>	Tools.ExternalTools
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0093

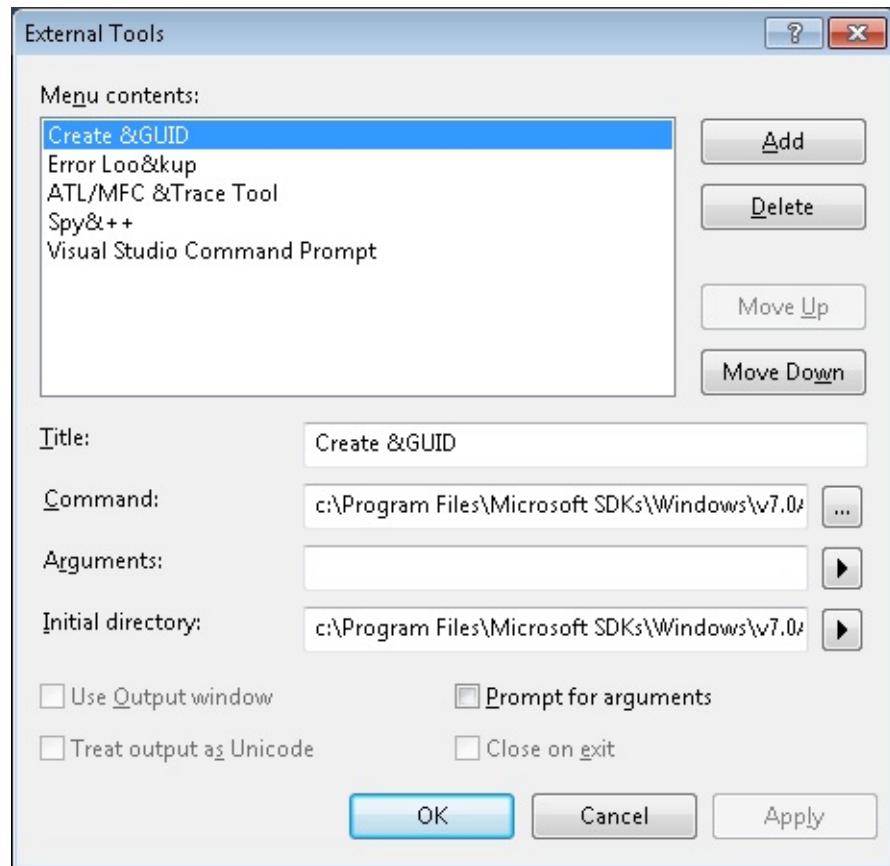
In vstipTool0059 ([03.31 Using External Tools](#), page 136), I showed you how to add external tools to the Visual Studio Tools menu).



After you create an entry, you probably want to have an accelerator key assigned so that you can use your keyboard to activate the new tool. First, you can see what these keys look like by going to the Tools menu, using your keyboard (Alt+T).



Now you can press any letter that is underlined to access that item. For example, if the letter “G” if pressed, it opens up the Create GUID tool. You can see how this is created by going to Tools | External Tools on the menu bar.

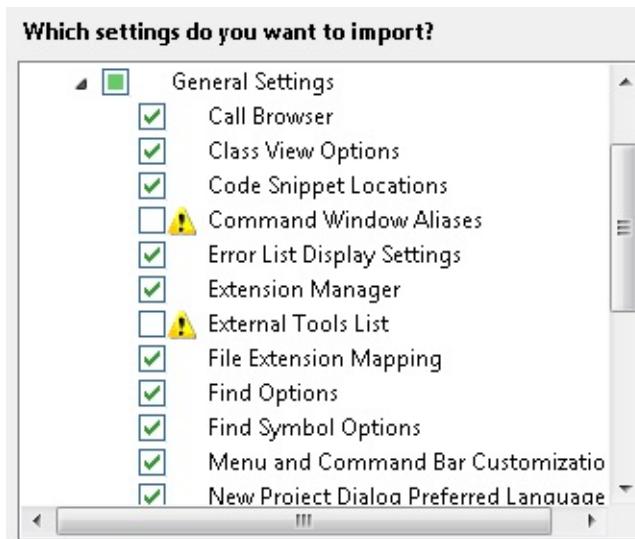


Notice the ampersand (&) before the “G” in “GUID”? Anytime you put an ampersand before any character in your title, the next character after it becomes the accelerator key. As you can see, all these entries have accelerator keys assigned to them. Now you can create special keys for your external tools access.

## 03.33 Exporting Your Command Window Aliases and External Tools List

<b>WINDOWS</b>	Alt,T, I
<b>MENU</b>	Tools   Import and Export Settings
<b>COMMAND</b>	Tools.ImportandExportSettings
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0043

As you probably know, you can export just about any setting you want. Usually, you export several settings together. However, when it comes time to import settings, the following two settings do not get imported by default: Command Window Aliases and External Tools List:



Sometimes, when importing, users just click through the wizard and forget this is the case. For this reason, it might be a good idea to make sure that you have these two settings exported separately as an extra copy. Then you can just import these settings as needed.

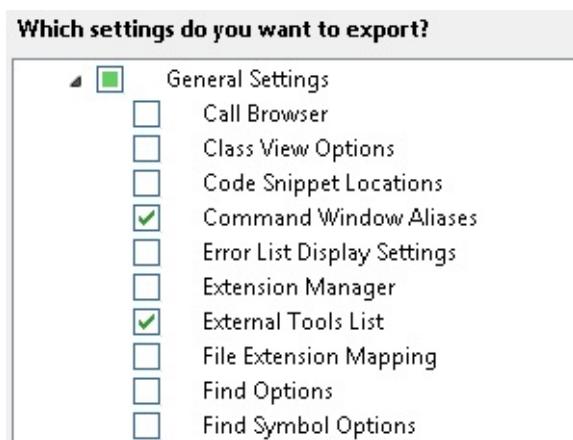
To do this, just go to Tools | Import And Export Settings and select Export Selected Environment Settings:

What do you want to do?

**Export selected environment settings**

Settings will be saved out to a file so they can later be imported at any time on any machine.

Click Next, clear the All Settings check box, and then select the Command Window Aliases and External Tools List check boxes:

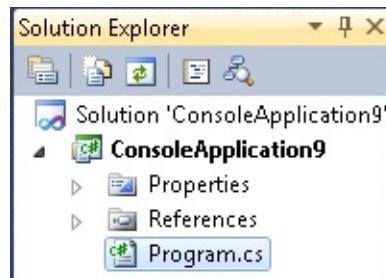


From here, just complete the wizard normally and make sure to give your exported file a logical name, such as Command Aliases and External Tools List.

## 03.34 Creating and Using a Macro

<b>DEFAULT</b>	Ctrl+Shift+R (record/stop recording); Ctrl+Shift+P (run)
<b>VISUAL BASIC 6</b>	[no shortcut] (record/stop recording); [no shortcut] (run)
<b>VISUAL C# 2005</b>	Ctrl+Shift+R (record/stop recording); Ctrl+Shift+P (run)
<b>VISUAL C++ 2</b>	Ctrl+Shift+R (record/stop recording); Ctrl+Shift+P (run)
<b>VISUAL C++ 6</b>	Ctrl+Shift+R (record/stop recording); Ctrl+Shift+P (run)
<b>VISUAL STUDIO 6</b>	[no shortcut] (record/stop recording)
<b>WINDOWS</b>	Alt,T, M, C (record / stop recording); Alt,T, M, R (run)
<b>MENU</b>	Tools   Macros   Record Temporary Macro; Tools   Macros   Run Temporary Macro; Tools   Macros   Save Temporary Macro
<b>COMMAND</b>	Tools.RecordTemporaryMacro; Tools.RunTemporaryMacro; Tools.SaveTemporaryMacro
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0055

You can record macros to do just about anything in Visual Studio. In this example, we create a macro that adds a new class to our project. First, create a new project. For this example, create a Console Application:



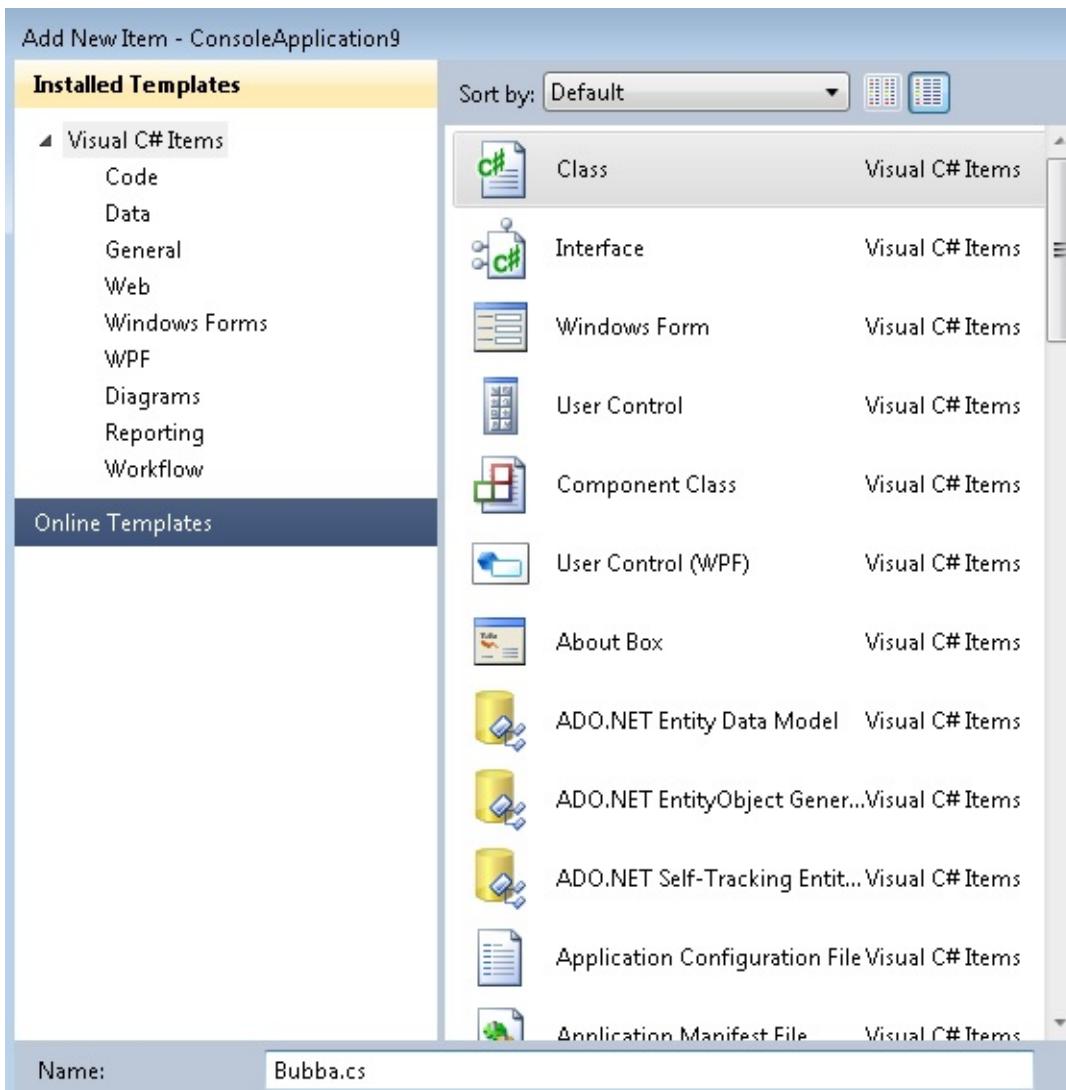
Now we are going to add a class to the project and give the class a name. When we do this, we will record the actions into a temporary macro by pressing Ctrl+Shift+R. You should see the following status in the lower-left corner:



Add a new class (Ctrl+Shift+A) called **Bubba.cs**:

**WARNING**

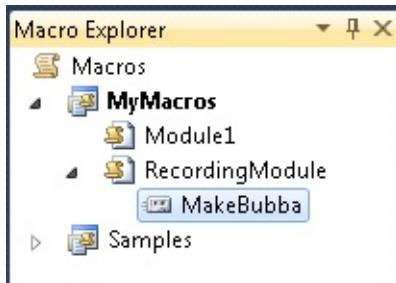
You can run into numerous little “gotchas” when creating macros. One that took me a little while to figure out was leaving off the “.cs” at the end of the file name. For some reason, Visual Studio doesn’t like that at all. Keep an eye out for little things like that as you use this feature.



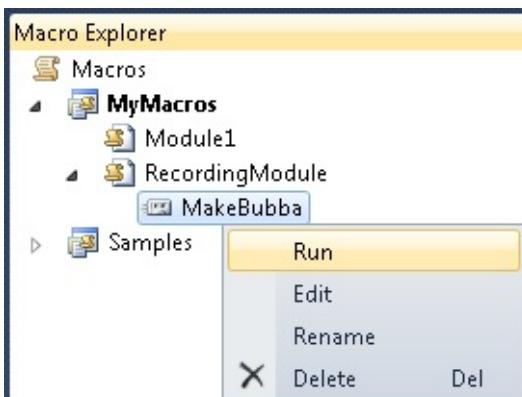
Add a comment to the class:

```
namespace ConsoleApplication1
{
    class Bubba
    {
        // this is the bubba class
    }
}
```

Stop recording (Ctrl+Shift+R), and then go to Tools | Macros | Save Temporary Macro. Name the macro **MakeBubba** in the Macro Explorer:



Now we can test out our new macro by creating a new project and going to the Macro Explorer (Alt+F8). Right-click the MakeBubba macro, and choose Run:



It should make the new class and put your comment in:

```
namespace ConsoleApplication1
{
    class Bubba
    {
        // this is the bubba class
    }
}
```

After you have created your macro, you might want to see the code behind it. You can go to the Macro Explorer (Alt+F8), right-click any macro, and then choose Edit to see the code. Here is what my code (cleaned up a bit) looks like for the macro we just made:

```
Sub MakeBubba()
    DTE.ItemOperations.AddNewItem _
        ("Visual C# Items\Code\Class", "Bubba.cs")

    DTE.ActiveDocument.Selection.LineDown(False, 8)
    DTE.ActiveDocument.Selection.CharRight(False, 5)
    DTE.ActiveDocument.Selection.NewLine()

    DTE.ActiveDocument.Selection.Text = "/* bubba class"
End Sub
```

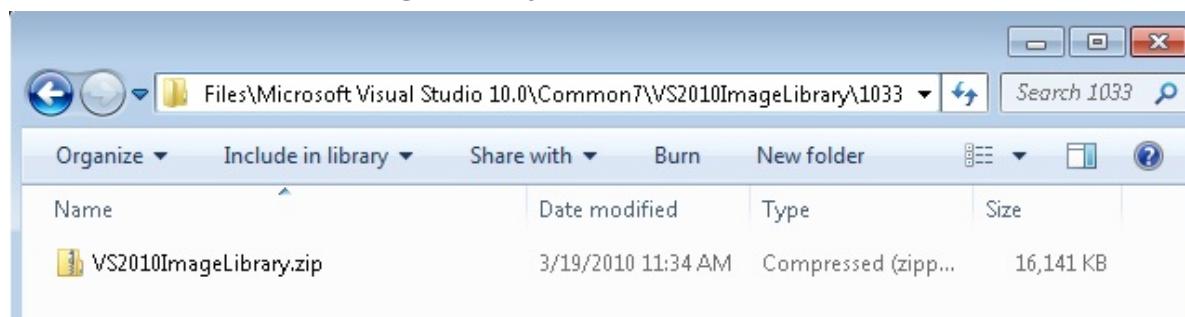
Notice that it is fairly easy to read and understand, which makes it easy to edit as well. Now that you have a working macro, you should visit vstipTool0066 ([AX.37 Create a Shortcut Key for a Macro](#), in Appendix B [<http://go.microsoft.com/fwlink/?LinkId=223758>]) and create a shortcut key for it.

## 03.35 Visual Studio Image Library

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0049

As a developer, you are always looking for images that can be used in your applications. Visual Studio comes with a set of images to help you out. In fact, it comes with over 2000 output files, images ready for immediate use, as well as a variety of source files that you can use to create your own images if needed. These images come from Microsoft Windows, the Office system, Microsoft Visual Studio, and other Microsoft software.

You can find them in a .zip file located at “C:\Program Files\Microsoft Visual Studio <version>\Common7\VS<version>ImageLibrary\1033.” For example, I found my images at “C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\VS2010ImageLibrary\1033”:



## Types of Files

The Visual Studio Image Library folders contain source files and output files.

### Source files

Source files contain building blocks intended for use with an image editor to generate new or to customize existing icons.



## Output files

Typically, output files are composite images made up of a base concept with 1–3 adorners, ready for immediate use by developers.



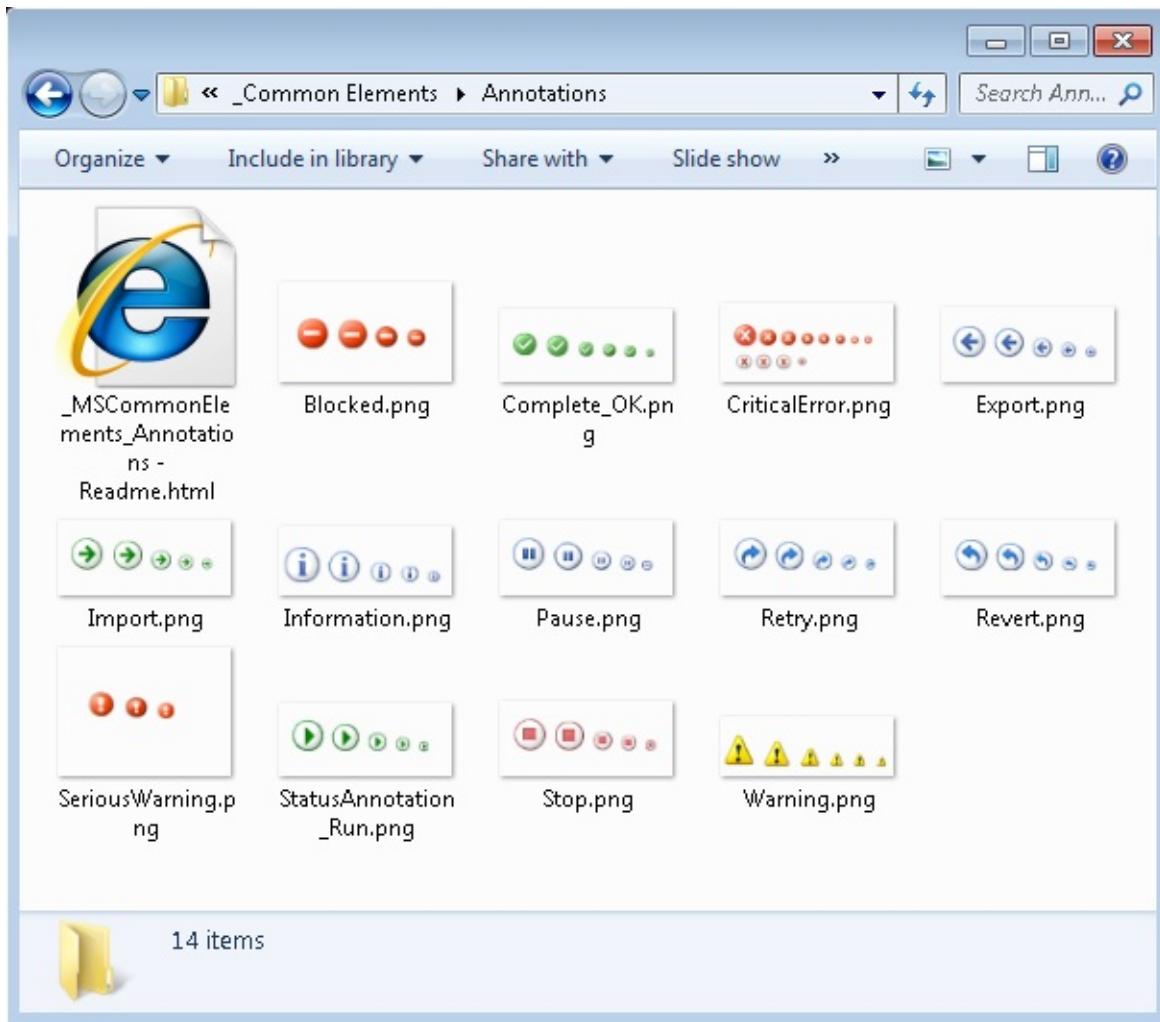
## Image Library Contents

The following illustration provides an idea of what is included in the Visual Studio Image Library and what each group of images is used for:

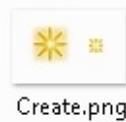
Name	Date modified	Type
_Common Elements	6/22/2011 9:51 AM	File folder
Actions	6/22/2011 9:51 AM	File folder
Animations	6/22/2011 9:52 AM	File folder
Annotations_Buttons	6/22/2011 9:52 AM	File folder
Objects	6/22/2011 9:51 AM	File folder

### Common Elements

Have you ever tried to edit an existing image that is made up of several overlapping elements? Then you know how hard it is to make simple changes in an output file. In this section, each source file is made up of various sizes of each element on a transparent background. When used with an image editor that utilizes layers, such as Adobe Photoshop or Paint.NET, you can choose the size of an element that fits best with what you're trying to do and then make adjustments by layering the pieces, moving them around without editing the bits that are in the lower layers.



When the final image is composed just the way you like it, you can then flatten and save the file in a usable format such as .ico, .bmp, or .png. A typical use of source files would be to take an existing image and add an adorner from the \_Common Elements source files, such as adding a “new” star to the upper left to indicate a command that creates a new item of that base type:



## Actions

The Actions folder set contains output file images that represent verbs. Most commands are verbs, so if you are building a toolbar or ribbon, you would find most of those images in the Actions folder. In the Visual Studio Image Library, the Actions images are separated by format (24-bit, 32-bit, .ico, and .png), size

(16x16, 24x24, 32x32), and style (Office/Visual Studio, DataTools, Windows Vista, and Windows XP).



Use the format that works best within your code; 24-bit .bmp files use a fuchsia color that you can map to the background color of your user interface so that it appears to have a transparent background; 32-bit .bmp files contain a transparent background, although in File Explorer it appears black, and .ico and .png files have transparent backgrounds as well. Also, each style is illustrated differently, so generally, you should choose one style and stay consistent, not mixing different styles in the same user interface.

## Animations

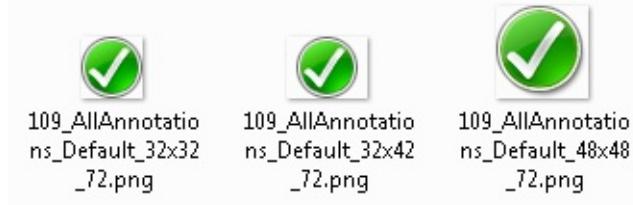
This folder contains a few of the common animations that you see in Windows and that are used in dialog boxes or other user interface elements to indicate that a process is underway. You have .avi and .gif formats available for most of the animations; which format you use will be determined by which format is best supported by the technology you're using for your user interface.



## Annotations\_Buttons

The Annotations\_Buttons folder contains images for notifications, simple actions such as expand/collapse, or to describe the state of an object or process—for example, running, paused, or offline. This group of files is also separated out by format (24-bit .bmp, .ico, and .png) as well as style. Notice that the Windows Vista .ico annotations contain the full range of sizes that can be viewed in the File

Explorer and appear correctly in accordance with the size required by your user interface.

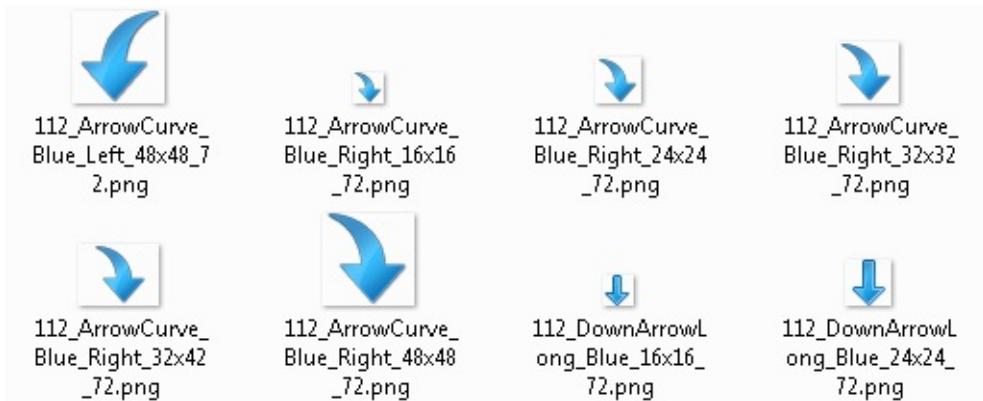


## Objects

Objects are output image files representing nouns. Because they represent objects, the most common usage for these files is in tree views, list views, or containers such as toolboxes. When used in this way, these images enable the user to scan a list of elements and to identify types of objects without needing to read the name of the item. Sometimes objects can also represent commands, such as a command to create a new object of that type (New File) or to launch a user interface element related to that object (for example, a stopwatch image used to indicate Start Timer).

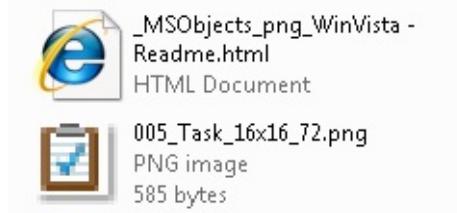


In the png\_format Windows Vista folder, you can find a wide variety of sizes and colors of various flags, arrows, +/- signs, and so forth, which can be used to indicate a variety of meanings. Generally, Object images are used as base elements when creating a new compound icon. Base elements augmented by annotations or other actions/object images can indicate the state or type of the base image and form a visual language when used with variations of similar icons:



## Using the Images

The images are meant to be used for their original intent. So, for example, you can't use the Paste image for something other than a paste operation. When you are using or creating these images, it is important that you make sure to use the images in a manner that is consistent with the description of the respective image in the readme document found in its respective folder:



# Chapter 4. Working with Documents

*“He turned off the word processor, realizing just a second after he’d flicked the switch that he’d forgotten to save the document. Well, that was all right. Maybe it had even been the critic in his subconscious, telling him the document wasn’t worth saving.”*

— Stephen King “Secret Window, Secret Garden” in “Four Past Midnight”

Documents serve as the cornerstone of your activities in Visual Studio. Writing code, debugging code, creating interfaces, or just about anything else you do is done with documents. Yet we still seem to take our documents for granted. This chapter focuses on working with documents in the File Tab Channel as well as ways to navigate better. Several advanced topics, such as creating custom file extensions and working with previous versions are covered as well.

## 04.01 Insert Documents to the Right of Existing Tabs

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Documents
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEnv0001

By default, Visual Studio 2010 opens up new tabs to the left of existing ones, as shown in the following illustration.



You now have an option to put newly opened documents in the file channel to the right of existing tabs.

Just go to Tools | Options | Environment | Documents, and select the Insert Documents To The Right Of Existing Tabs option.

You should see new tabs show up to the right of existing tabs.

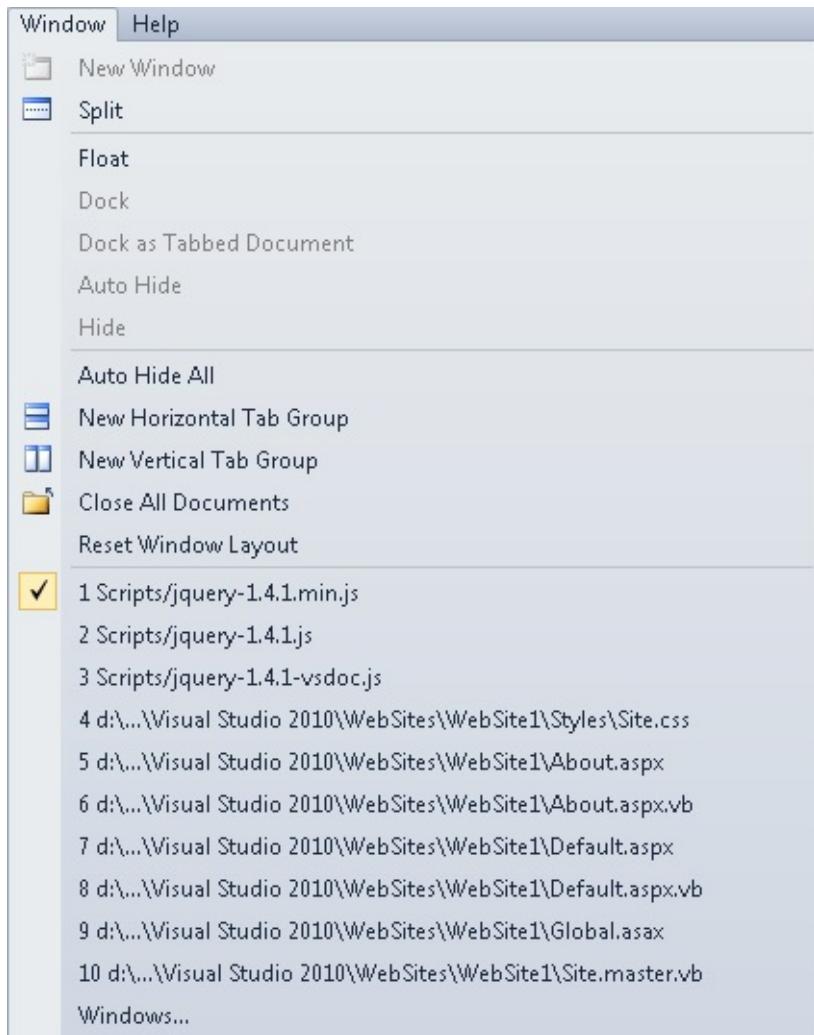


## 04.02 Recent Files

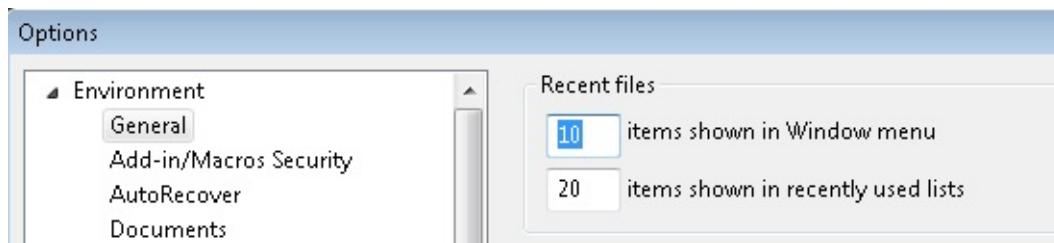
<b>WINDOWS</b>	Alt,W, [1,2,3, etc] (windows); Alt,F, F, [1,2,3, etc] (files); Alt,F, J, [1,2,3, etc] (projects and solutions)
<b>MENU</b>	Tools   Options   General   Recent files; Window   [1,2,3, etc]; File   Recent Files   [1,2,3, etc]; File   Recent Projects and Solutions   [1,2,3, etc]
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0013

By default, the Window menu shows the 10 most recent files you had open, as shown in the illustration to the right.

Likewise, the Recent Files and Recent Projects And Solutions items on the File menu show only the last 20 entries.



You can easily modify these numbers (up or down) by going to Tools | Options | General | Recent files.

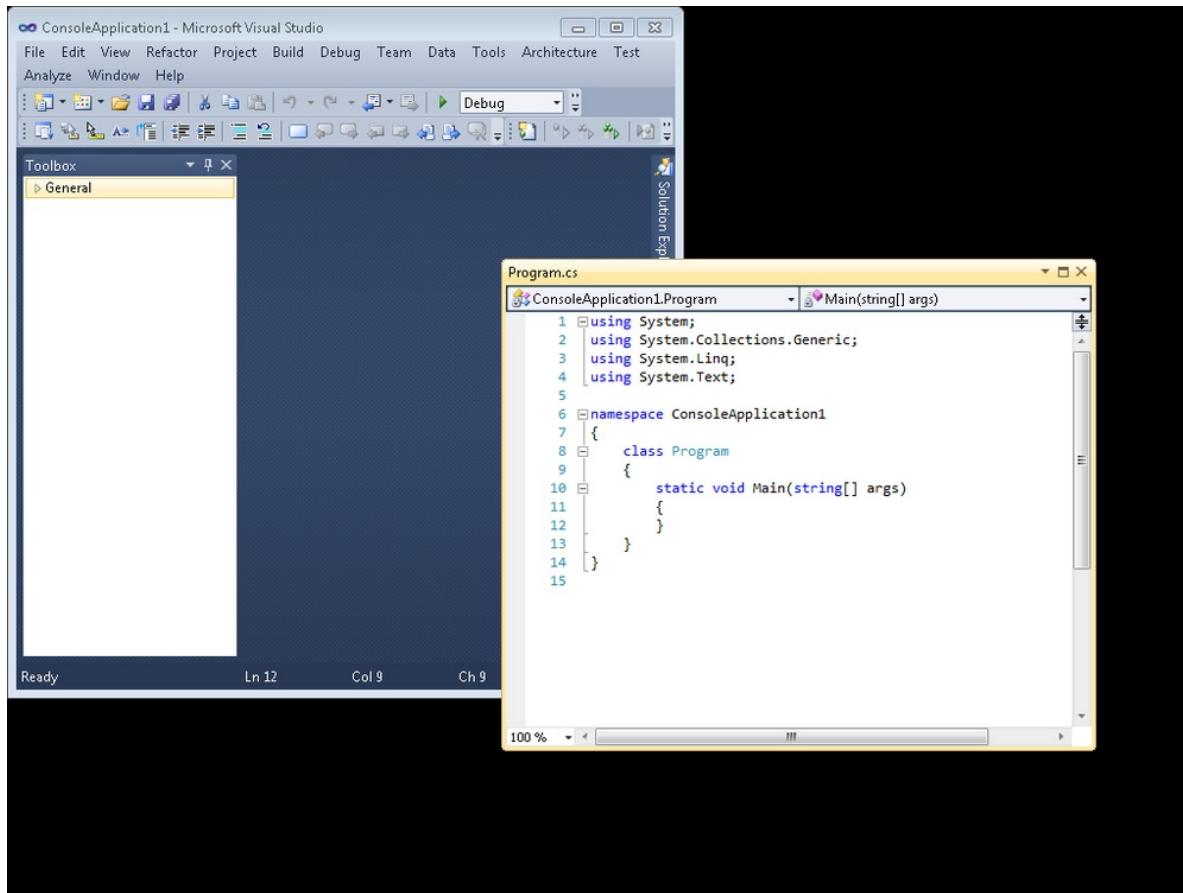


For each of these options, the minimum value is 1 and the maximum value is 24. Experiment with numbers that suit your taste.

## 04.03 Working with Documents on Multiple Monitors

<b>WINDOWS</b>	Alt,W, F (float); Alt,W,T (dock)
<b>MENU</b>	Window   Float; Window   Dock as Tabbed Document
<b>COMMAND</b>	Window.Float; Window.DockasTabbedDocument
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0004

This is one we have all been wanting for a long time: detachable document windows. You can now detach document windows and put them on another monitor! You have a couple of ways to do this.



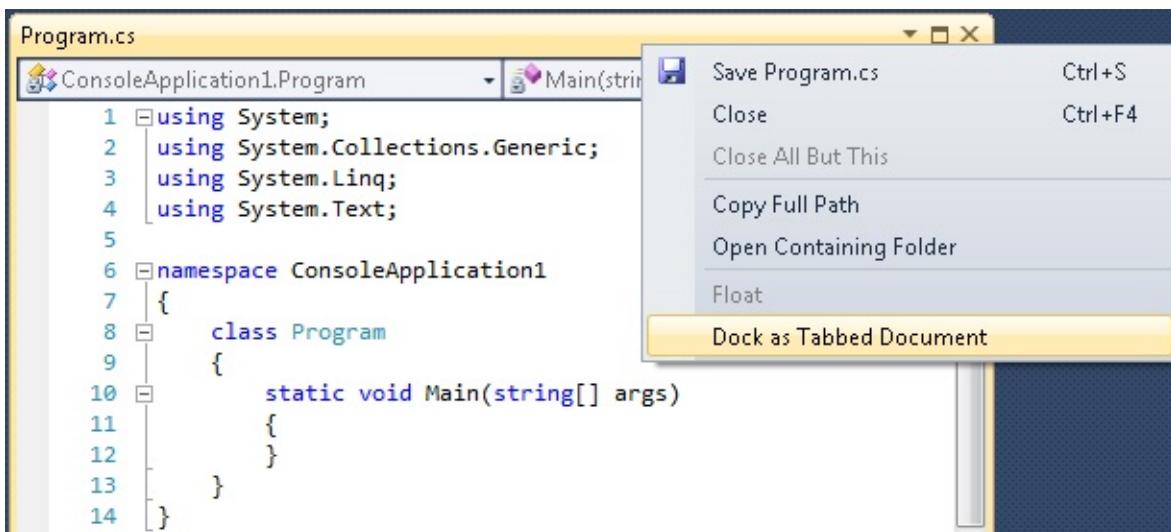
The first way is to simply click and drag the tab for the document window out of the IDE.



The second way is to go to Window | Float on the menu bar, as shown in the following illustration.

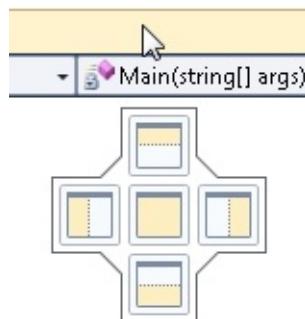


What if you want to put the window back? No worries; just right-click the title bar of the document and choose Dock As Tabbed Document.



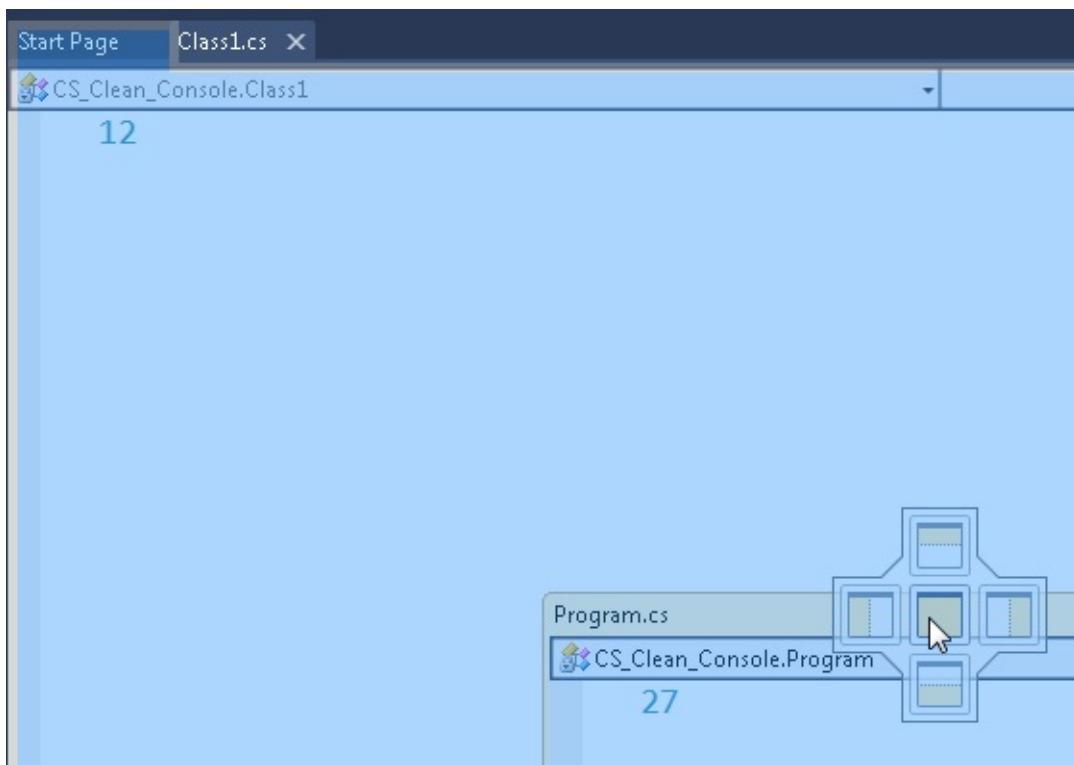
If you want an alternative method, you can go the following route: Click and drag the document window by its title bar into the IDE.

The guide diamond appears in the IDE, as shown in the following illustration.



Hold down the left mouse button, and move your cursor over the middle item in the guide diamond. You should see an outline of where the window will be

docked.



Release the mouse button, and it should dock where you want it to go.

## 04.04 Navigate Open Document Windows

<b>DEFAULT</b>	Ctrl+F6 (next); Ctrl+Shift+F6 (previous)
<b>VISUAL C++ 2</b>	Ctrl+F6; Ctrl+Tab (next) Ctrl+Shift+F6; Ctrl+Shift+Tab (previous)
<b>VISUAL STUDIO 6</b>	Ctrl+F6; Ctrl+Tab (next) Ctrl+Shift+F6; Ctrl+Shift+Tab (previous)
<b>COMMAND</b>	Window.NextDocumentWindow; Window.PreviousDocumentWindow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0013

OK, so you have a lot of files open in the file channel:



And you don't want to use your mouse to switch between tabs. Just press Ctrl+F6 to go forward.

Or press Ctrl+Shift+F6 to go backward.

## 04.05 Close the Current Document Window

<b>DEFAULT</b>	Ctrl+F4
<b>VISUAL BASIC 6</b>	Ctrl+F4
<b>VISUAL C# 2005</b>	Ctrl+F4
<b>VISUAL C++ 2</b>	Ctrl+F4
<b>VISUAL C++ 6</b>	Ctrl+F4
<b>VISUAL STUDIO 6</b>	Ctrl+F4
<b>WINDOWS</b>	Alt,F, C
<b>MENU</b>	File   Close
<b>COMMAND</b>	Window.CloseDocumentWindow; File.Close
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0014

You can close the current document window from the keyboard. Just make sure you are in the document you want to close.



Then press Ctrl+F4. The current document closes, and it prompts you to save changes if you haven't already.

## 04.06 Open a File Location from the File Tab

<b>DEFAULT</b>	Alt+- (minus sign), O (VS2010 Only)
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt+- (minus sign), O (VS2010 Only)
<b>COMMAND</b>	File.OpenContainingFolder; Window.ShowDockMenu
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0014

Do you often find yourself needing to go to your project location in Windows Explorer? Just right-click the file's tab, and choose Open Containing Folder.

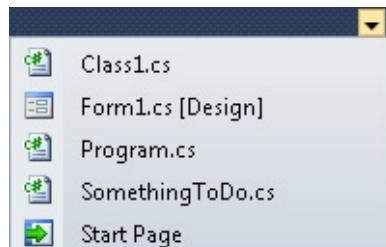


The file location opens in Windows Explorer, and you can manipulate the files from there.

## 04.07 Open the File Menu Drop-Down List from Your Keyboard

<b>DEFAULT</b>	Ctrl+Alt+Down Arrow
<b>VISUAL BASIC 6</b>	Ctrl+Alt+Down Arrow
<b>VISUAL C# 2005</b>	Ctrl+Alt+Down Arrow
<b>VISUAL C++ 2</b>	Ctrl+Alt+Down Arrow
<b>VISUAL C++ 6</b>	Ctrl+Alt+Down Arrow
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+Down Arrow
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Window.ShowEzMDIFileList
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0003

When you have a lot of files open, it is sometimes easier to view them as a list instead of tabs. The File menu drop-down list does that for you. You can click the drop-down button to the far right on the file tab, or you can simply use Ctrl+Alt+Down Arrow to activate it.

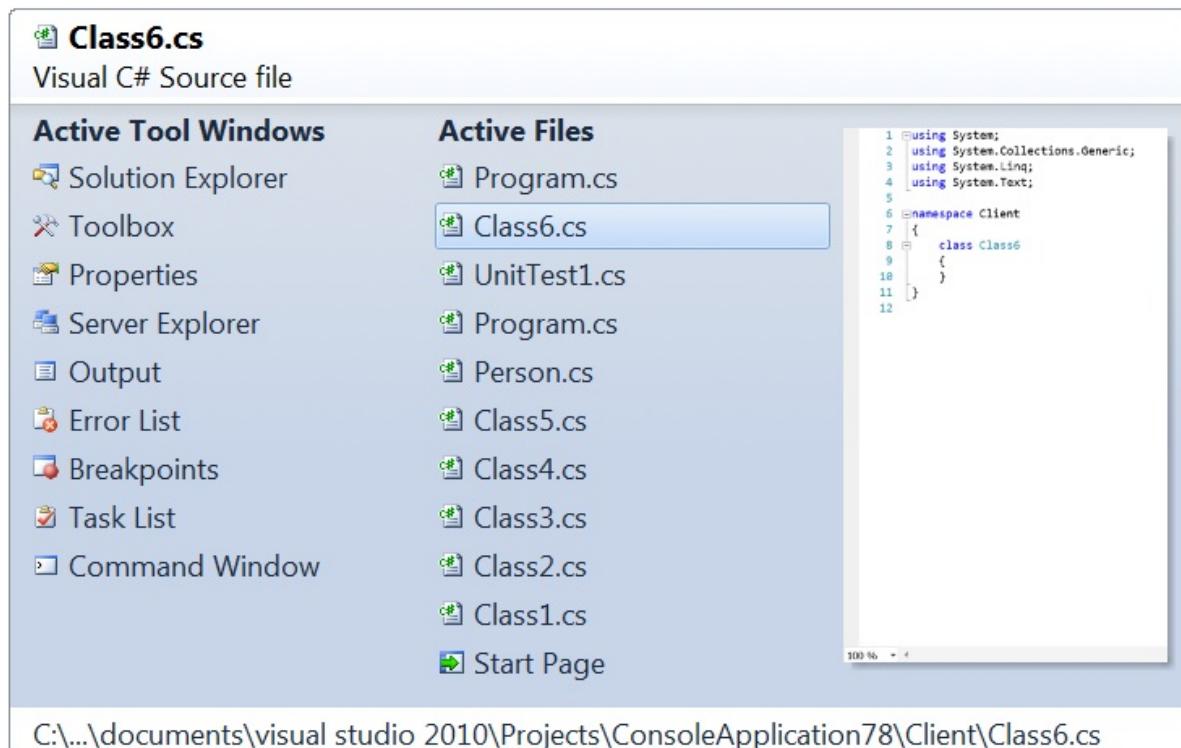


An interesting thing about this list is that it has type-ahead functionality. So, in this example, if you type the letter **S**, it automatically selects **SomethingToDo.cs**. Hitting **S** again results in **Start Page** being selected. If you have a lot of files, you can type more characters to narrow down the selection. For example, typing **ST** jumps straight to **Start Page**.

## 04.08 Using the IDE Navigator

<b>DEFAULT</b>	Ctrl+Tab (forward in Active Files); Ctrl+Shift+Tab (backward in Active Files); Alt+F7 (forward in Active Tool Windows); Alt+Shift+F7 (backward in Active Tool Windows)
<b>VISUAL BASIC 6</b>	[no shortcuts]
<b>VISUAL C# 2005</b>	Ctrl+Tab (forward in Active Files); Ctrl+Shift+Tab (backward in Active Files); Alt+F7 (forward in Active Tool Windows); Alt+Shift+F7 (backward in Active Tool Windows)
<b>VISUAL C++ 2</b>	[no shortcuts]
<b>VISUAL C++ 6</b>	Ctrl+Tab (forward in Active Files); Ctrl+Shift+Tab (backward in Active Files); Alt+F7 (forward in Active Tool Windows); Alt+Shift+F7 (backward in Active Tool Windows)
<b>VISUAL STUDIO 6</b>	Ctrl+Tab (forward in Active Files); Ctrl+Shift+Tab (backward in Active Files); Alt+F7 (forward in Active Tool Windows); Alt+Shift+F7 (backward in Active Tool Windows)
<b>WINDOWS</b>	[no shortcuts]
<b>COMMAND</b>	Window.NextDocumentWindowNav; Window.PreviousDocumentWindowNav; Window.NextToolWindowNav; Window.PreviousToolWindowNav
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0023

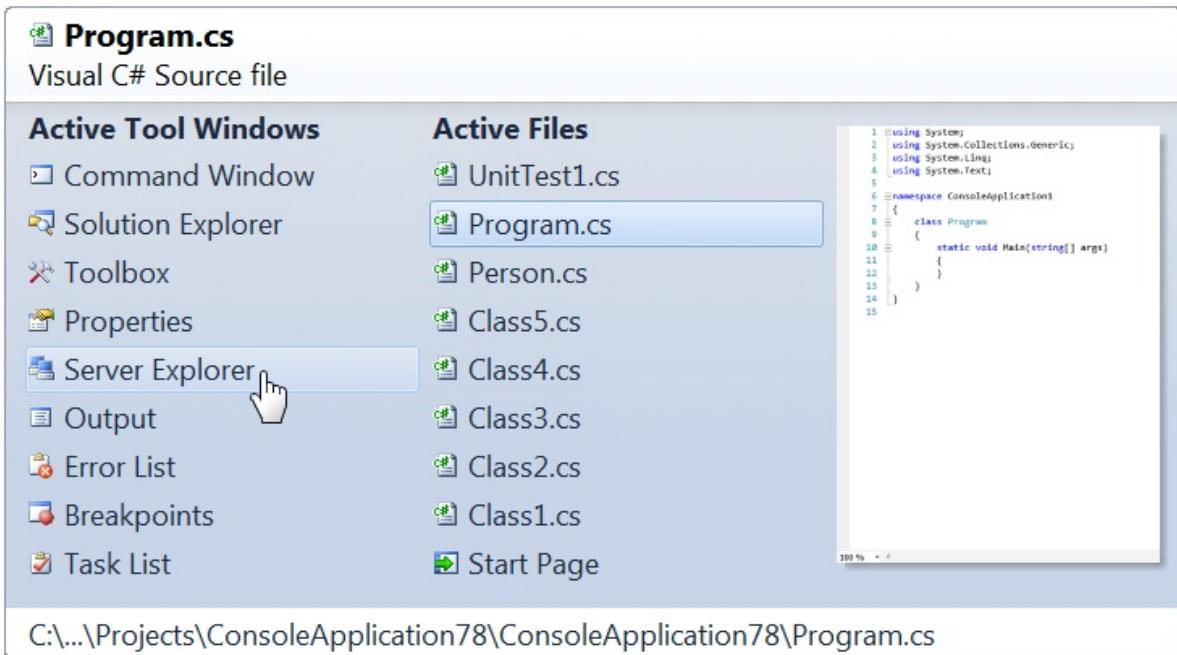
Navigating documents and tool windows in the IDE is a critical part of your development experience. You can easily move among active file and tool windows by pressing Ctrl+Tab.



#### NOTE

The images in this tip show the IDE Navigator with document preview (image to the right of the lists). This feature is off by default in Visual Studio 2010, but can be turned on as shown in vstipTool0113, [AX.43](#) [Thumbnail Previews in the IDE Navigator](#), in Appendix B (<http://go.microsoft.com/fwlink/?LinkId=223758>).

Some interesting things come with using this feature. For example, holding down the Ctrl key keeps the IDE Navigator showing once it is up. Also, you can select any item in this dialog box, while it is showing, by using your mouse or arrow keys.



## Navigator Areas

Let's take a look at the two major areas in the navigator: Active Files and Active Tool Windows.

### Active files

To navigate active files, press **Ctrl+Tab** to go forward and **Ctrl+Shift+Tab** to go backward through the list. The currently selected file is highlighted, and its name is displayed at the top of the dialog box. Also, notice that the full file path is shown at the bottom of the IDE navigator.

### Active tool windows

This part of the dialog box shows all your tool windows that are currently open. To get to this area, you can use **Alt+F7** or **Alt+Shift+F7**. The interesting part is that this list changes depending on when you use it. The following illustration shows what mine looks like while I am writing code.

**Program.cs**  
Visual C# Source file

**Active Tool Windows**

- Command Window
- Solution Explorer
- Toolbox
- Properties
- Server Explorer
- Output
- Error List
- Breakpoints
- Task List

**Active Files**

- UnitTest1.cs
- Program.cs**
- Person.cs
- Class5.cs
- Class4.cs
- Class3.cs
- Class2.cs
- Class1.cs
- Start Page

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 
6 namespace ConsoleApplication1
7 {
8     class Program
9     {
10         static void Main(string[] args)
11         {
12         }
13     }
14 }
```

C:\...\Projects\ConsoleApplication78\ConsoleApplication78\Program.cs

And here's what it looks like when I'm debugging:

**UnitTest1.cs [ReadOnly]**  
Visual C# Source file

**Active Tool Windows**

- Output
- Command Window
- Toolbox
- IntelliTrace
- Breakpoints
- Immediate Window
- Autos
- Locals
- Watch 1
- Call Stack

**Active Files**

- Program.cs
- UnitTest1.cs [ReadOnly]**
- Program.cs [ReadOnly]
- Person.cs [ReadOnly]
- Class5.cs
- Class4.cs
- Class3.cs
- Class2.cs
- Class1.cs
- Start Page
- Threads
- Parallel Tasks
- Parallel Stacks

```
72
```

C:\...\visual studio 2010\Projects\ConsoleApplication78\TestProject1\UnitTest1.cs

## 04.09 Multiple Views of the Same Document

WINDOWS	Alt,W, N
MENU	Window   New Window
COMMAND	Window.NewWindow
VERSIONS	2005, 2008, 2010
CODE	vstipEnv0016

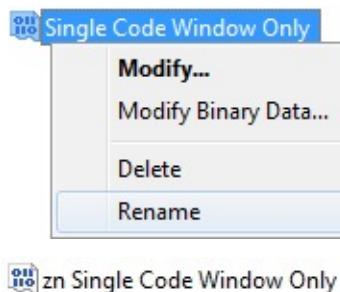
Sometimes you might want to look at a particularly large document in several different areas at the same time. For example, you might want to look at the same document on multiple monitors. This tip shows you how to make this happen.

### Special Note for VB Users in Visual Studio 2010

This feature is turned off by default in VB. A lot of history and reasoning is behind this, but the long and short of it is that this was fixed for 2010 but time ran out and it wasn't tested. So you can turn this on for VB, but you do so at your own risk. Special thanks to my friend Dustin Campbell for supplying the history and the fix.

To fix this, go to

"HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\VisualStudio\10.0\Language Services\Basic\" and rename the Single Code Window Only registry key to something like **[your initials here] Single Code Window Only**. The following illustration shows what I did:



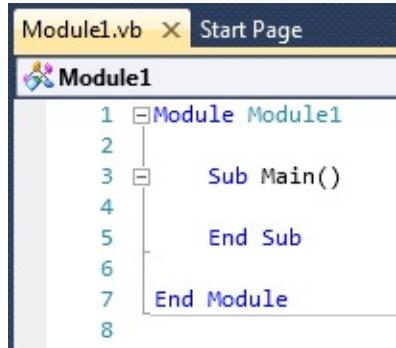
Now restart Visual Studio, and you are good to go for the rest of this tip.

### Multiple Views

I came across this while I was checking my email one day and noticed a thread started by the legendary Deborah Kurata concerning the Window | New Window

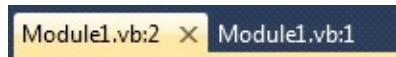
menu item. The following example describes how it works.

Open a document window.



```
Module1.vb X Start Page
Module1
1 Module Module1
2 Sub Main()
3 End Sub
4 End Module
```

Now go to Window | New Window on the menu bar to open a duplicate window of the current document.



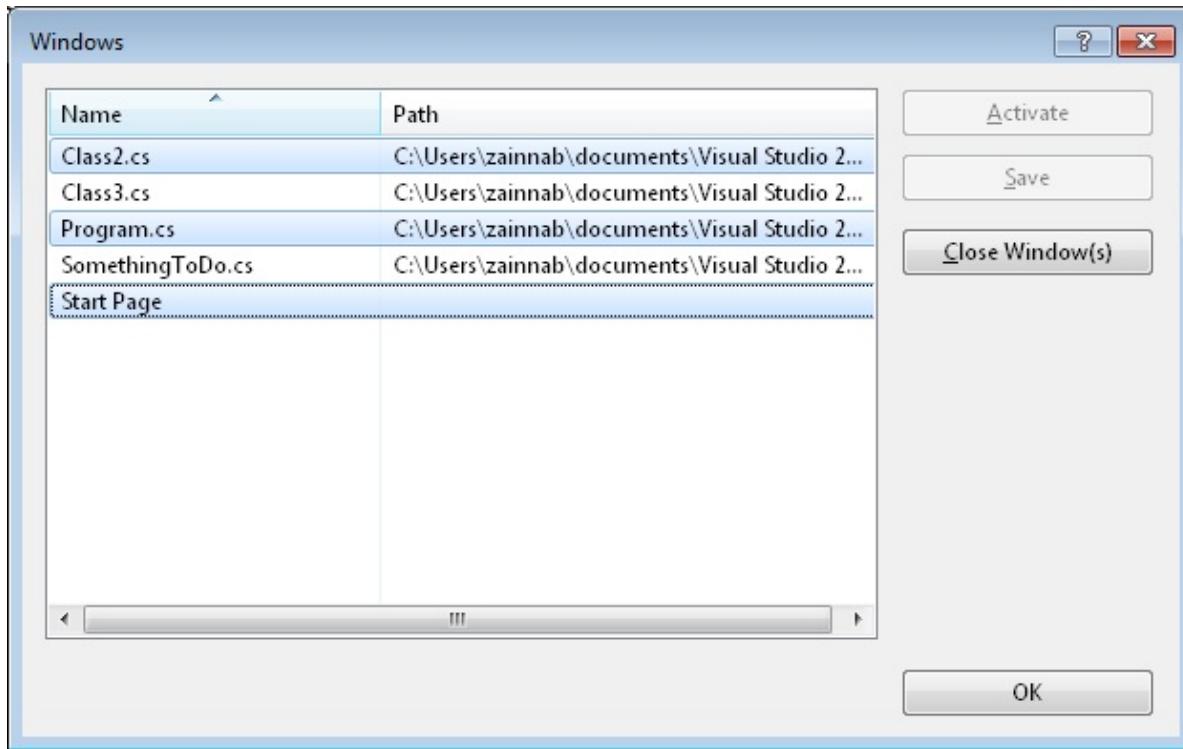
Notice that “:1” is added to the existing document tab text and that “:2” is appended to the name on the new document tab. You can apparently do this *forever* (or at least up to 150, which is as high as I have tested this feature).



## 04.10 Closing Just the Selected Files You Want

<b>WINDOWS</b>	Alt,W, W
<b>MENU</b>	Window   Windows
<b>COMMAND</b>	Window.Windows
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0010

What do you do when you have a lot of files open and want to close only a few of them? Just go to Window | Windows on the menu bar, as shown in the following illustration.

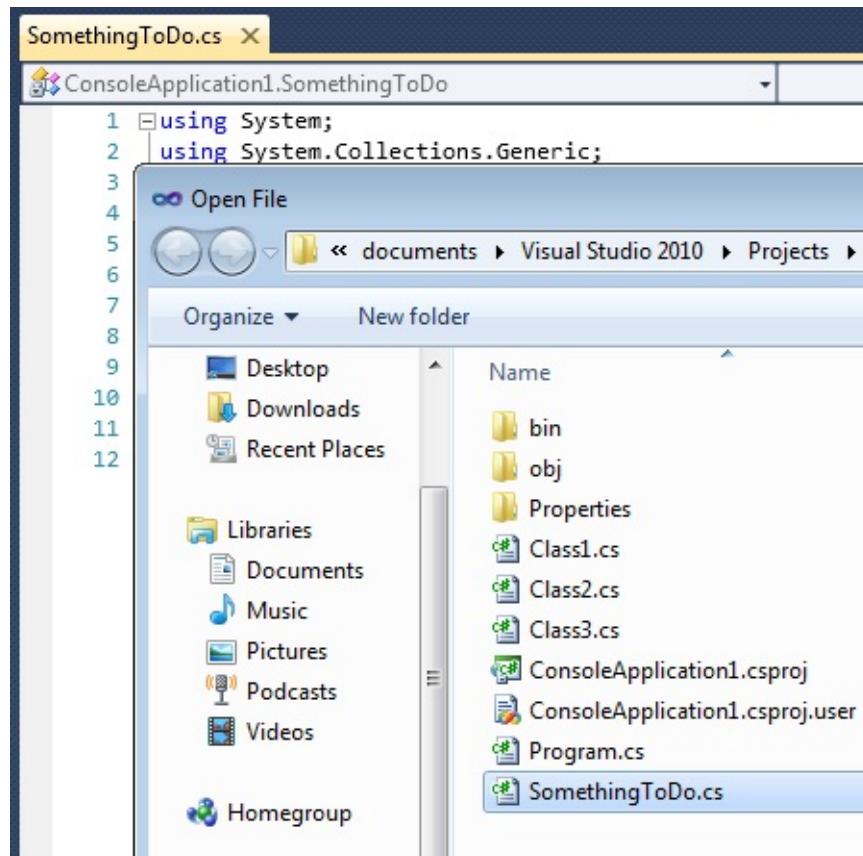


Select the files you want to close (Ctrl+Left-click), and then click Close Window(s). It closes the windows you selected and leaves the rest open.

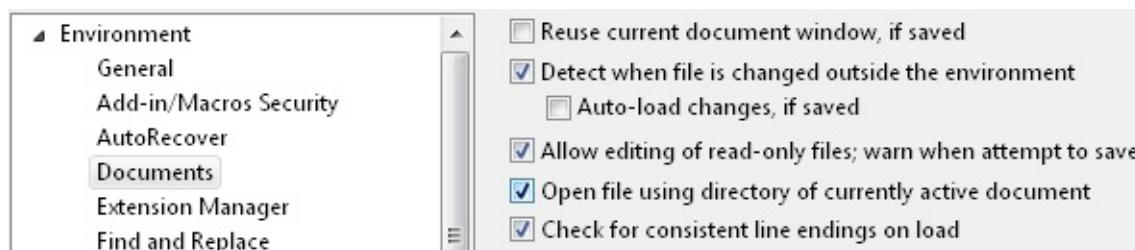
## 04.11 Understanding the File Open Location

<b>MENU</b>	Tools   Options   Environment   Documents
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0035

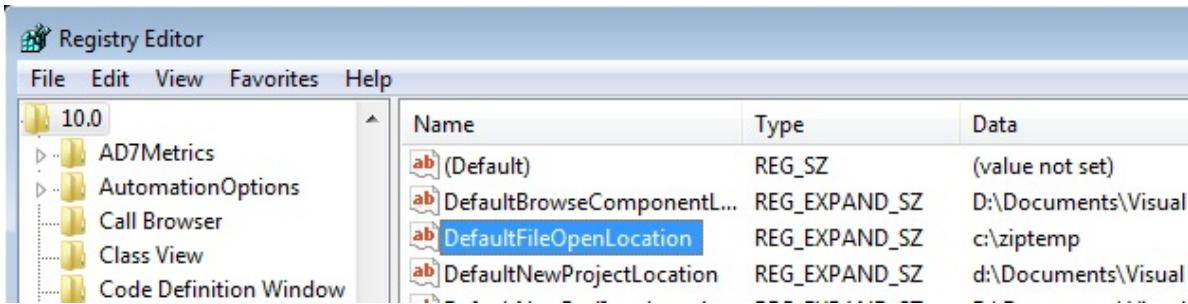
Have you ever noticed that when you go to open a file in Visual Studio (Ctrl+O) it automatically uses the directory of the current active document?



This is controlled by the Open File Using Directory Of Currently Active Document option. You can find this at Tools | Options | Environment | Documents.



You can turn this feature off by clearing its check box, and Visual Studio then uses the DefaultFileOpenLocation from HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\<version> in the registry instead.



Be aware that the DefaultFileOpenLocation changes every time you successfully open a file in the Open File dialog box. However, the update is not written to the registry until you close Visual Studio.

## 04.12 Show Previous Versions

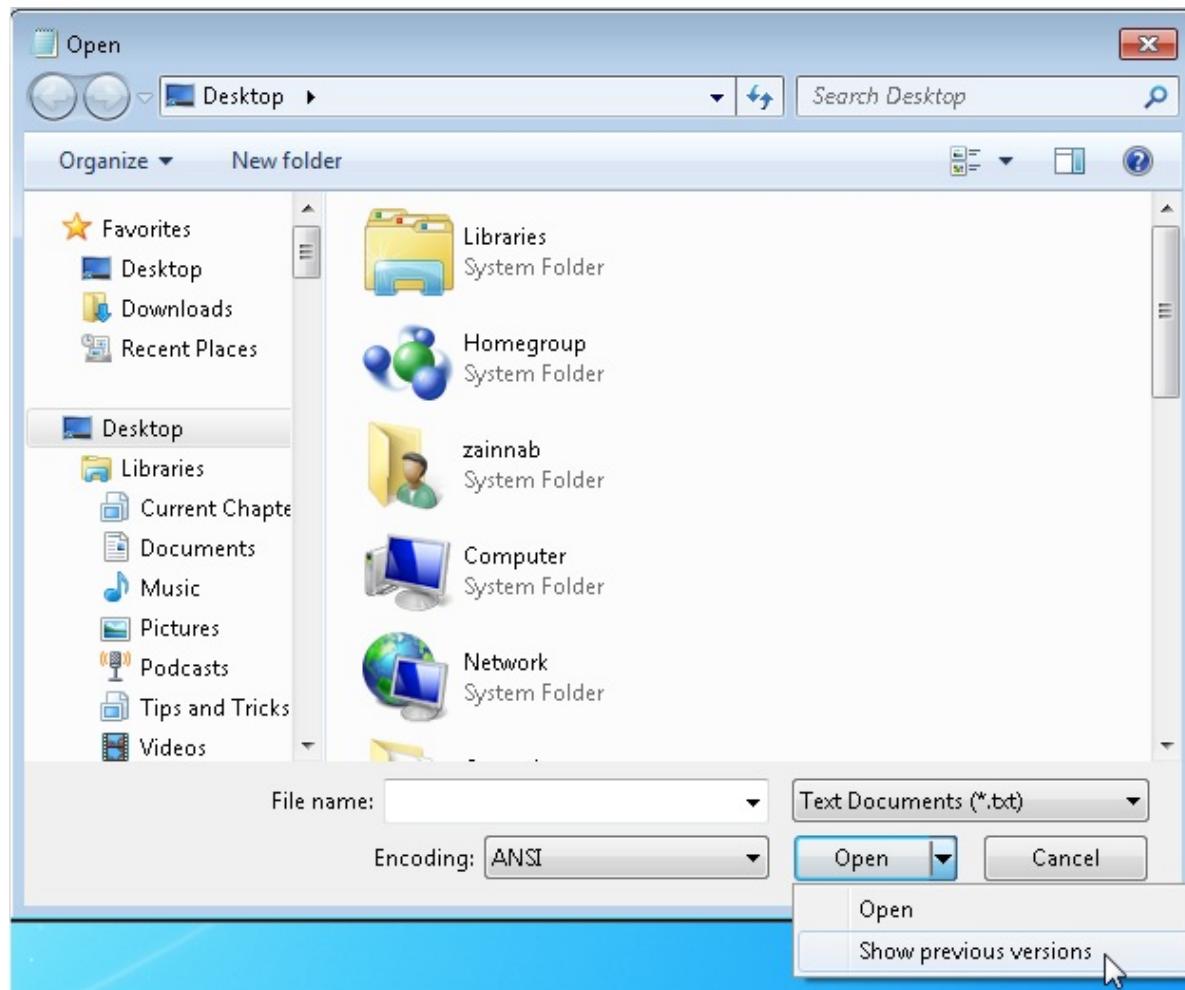
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEnv0036

### NOTE

For more information about previous versions including how to activate it if you don't currently have it turned on, go to <http://windows.microsoft.com/en-US/windows-vista/Previous-versions-of-files-frequently-asked-questions>.

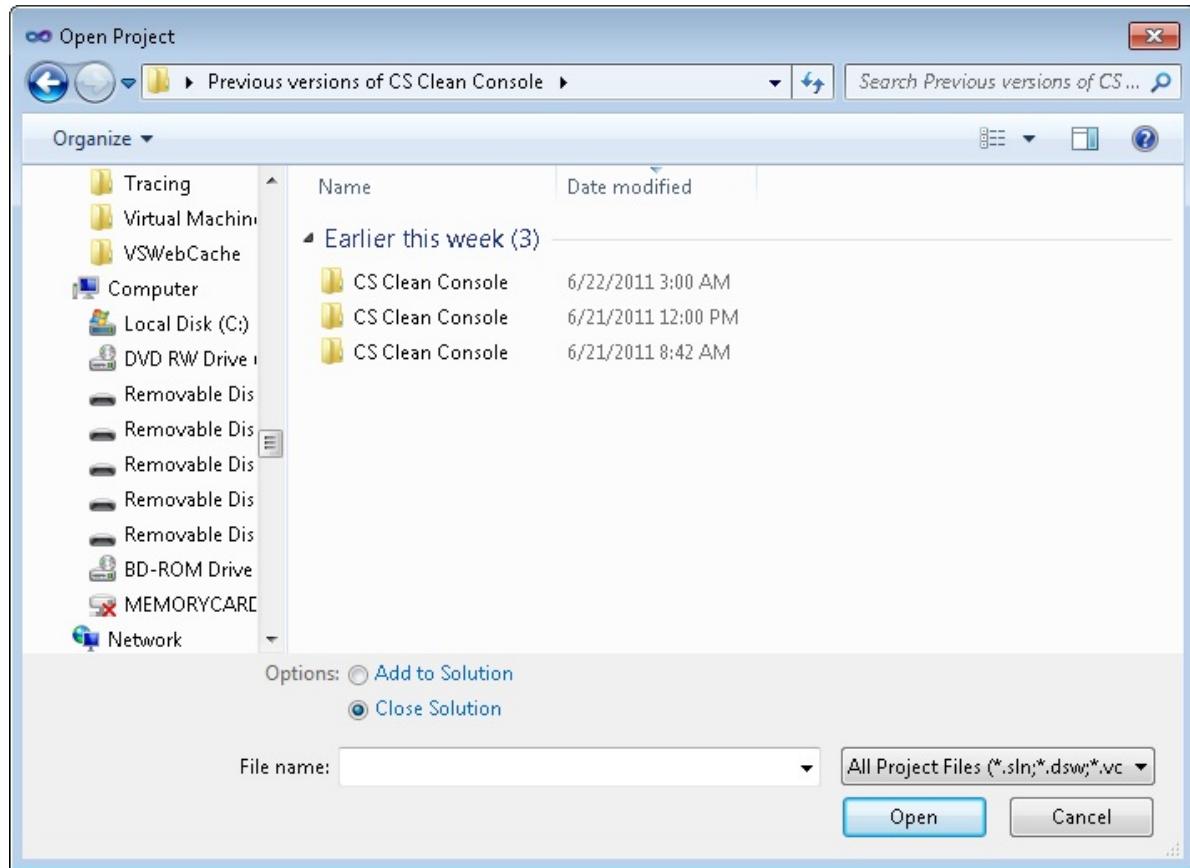
Ever want to go back in time when you save a change to your code that you didn't want saved? If you use source control, you are usually OK, but if you don't, this tip is for you.

If you run a Windows Vista or later operating system (excluding Home Editions), you have an option you might not have noticed before called Show Previous Versions. It shows up in various applications, such as Notepad, as shown in the following illustration.



You can also see this option in the Open Project dialog box as well.

When you click Show Previous Versions, you can see prior versions of the current directory you are in, as shown in the following illustration.

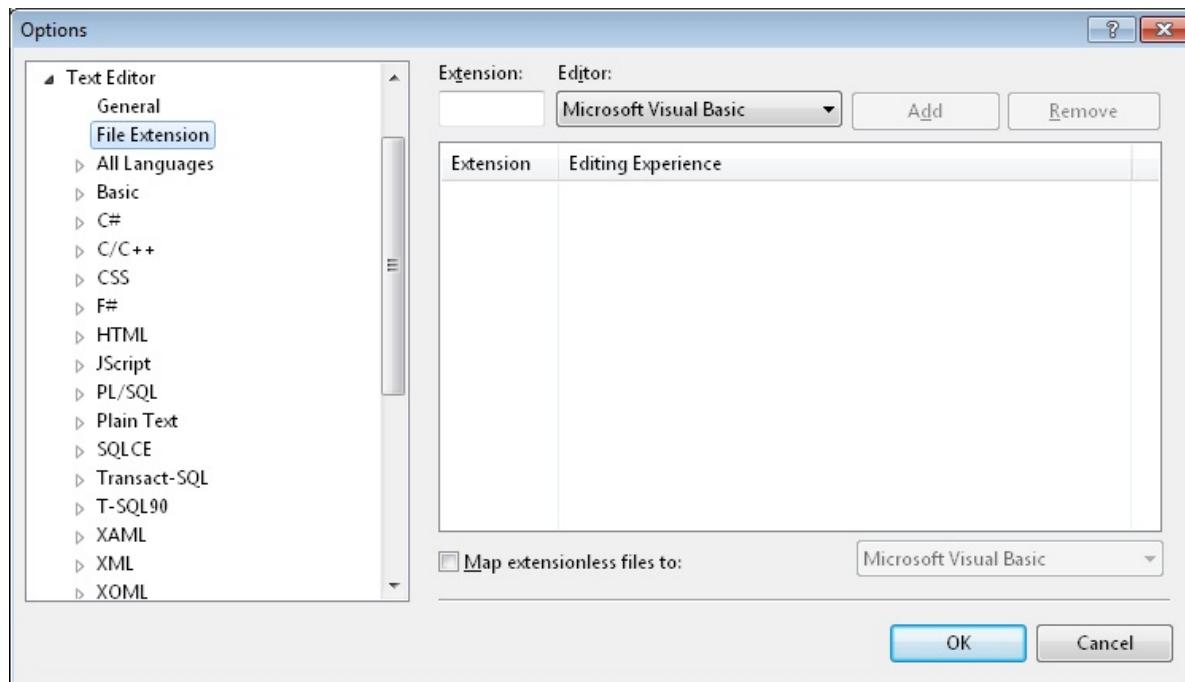


Now you can open previous versions of solutions, projects, files, and so forth, and do what you like.

## 04.13 Using Custom File Extension Associations

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   File Extension
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0038

If you have a custom file extension that you would like to associate with an editing experience, just go to Tools | Options | Text Editor | File Extension to see the options shown in the following illustration.



Simply type in your extension and the editing experience you want to have when it is opened. In the following example, I have a .cool extension that is associated with the Script Editor:



When I click Add, as shown in the preceding illustration, the following information is added to the overall list:

Extension:	Editor:
cool	Script Editor
<input type="button" value="Apply"/> <input type="button" value="Remove"/>	
Extension	Editing Experience
cool	Script Editor
jack	Transact-SQL Editor
stuff	Web Form Editor

Notice that you can select any item in the list, modify the extension and/or editor, and then click Apply to save the changes. Additionally, you can click Remove to take any entry out of the list.

# Chapter 5. Finding Things

*“I am in hopes, then, that we may find the object of our search thus. I imagine that our state, being rightly organized, is a perfectly good state.”*

— Plato “*The Republic*”

It's very frustrating when you are looking for something in your code and can't find it. This can range from a simple method definition to a complex set of classes in the .NET Framework, and everything in between. This chapter explores how to use the various search features in Visual Studio.

The ability to find information and then act on that information in some way is one of the central keys to creating good code. When we search for or replace code with complex criteria, our mastery of the various Find dialog boxes becomes the difference between a few minutes or several hours of work.

## 05.01 Repeat Your Last Search

<b>DEFAULT</b>	F3 (next); Shift+F3 (previous)
<b>VISUAL BASIC 6</b>	F3 (next); Shift+F3 (previous)
<b>VISUAL C# 2005</b>	F3 (next); Shift+F3 (previous)
<b>VISUAL C++ 2</b>	F3 (next); Shift+F3 (previous)
<b>VISUAL C++ 6</b>	F3 (next); Shift+F3 (previous)
<b>VISUAL STUDIO 6</b>	F3 (next); Shift+F3 (previous)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.FindNext; Edit.FindPrevious
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0004

After you perform a Find operation, you can quickly repeat that find. The following steps show you how.

Verify that your last Find shows up in the Find combo box on the Standard toolbar (usually located toward the upper right of your screen).



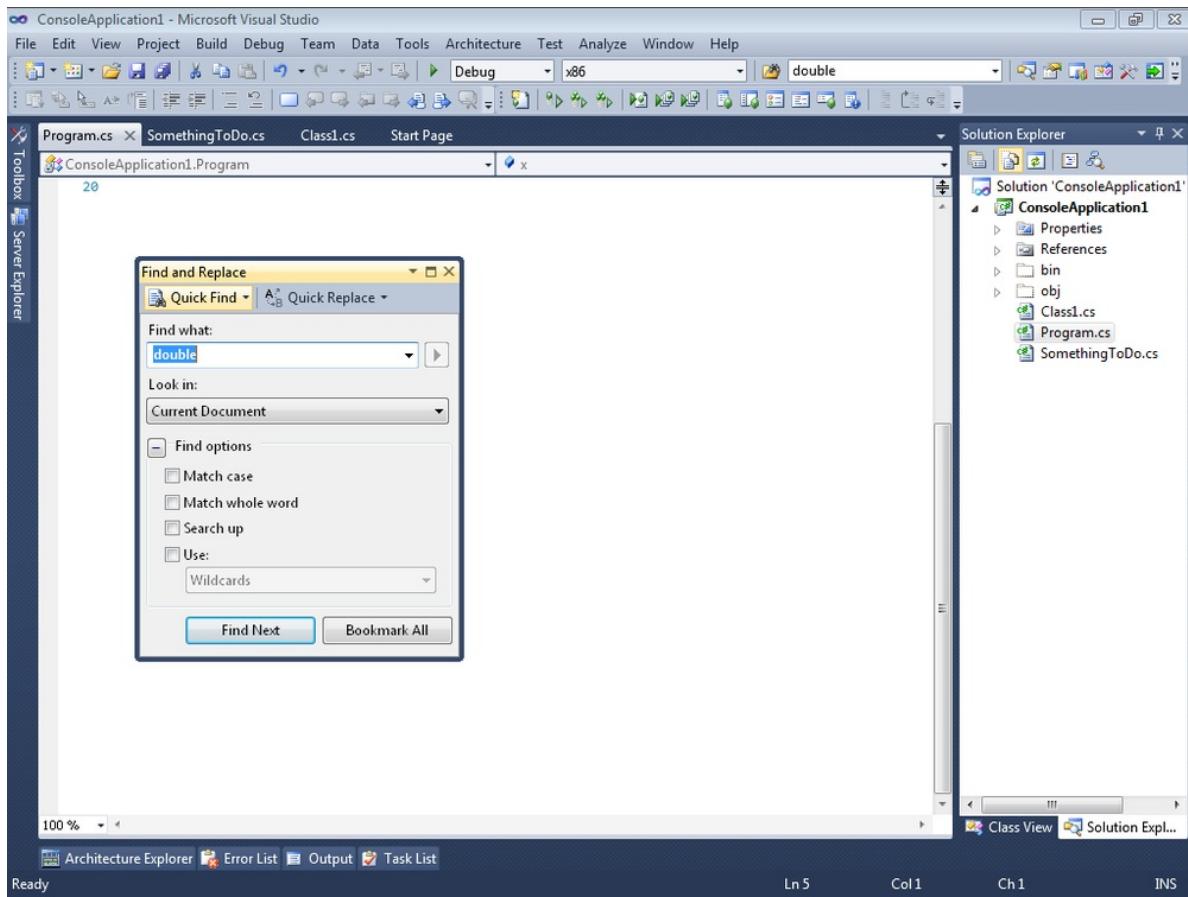
Press F3 (next) or Shift+F3 (previous) to move through the results.

Continue pressing F3 (next) or Shift+F3 (previous) as needed to find what you are looking for.

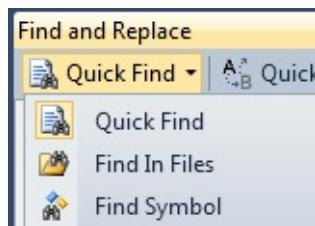
## 05.02 Using Quick Find

<b>DEFAULT</b>	Ctrl+F
<b>VISUAL BASIC 6</b>	Ctrl+F
<b>VISUAL C# 2005</b>	Ctrl+F
<b>VISUAL C++ 2</b>	Alt+F3
<b>VISUAL C++ 6</b>	Ctrl+F
<b>VISUAL STUDIO 6</b>	Ctrl+F
<b>WINDOWS</b>	Alt, E, F, F
<b>MENU</b>	Edit   Find and Replace   Quick Find
<b>COMMAND</b>	Edit.Find
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0007

There's more to Quick Find than meets the eye. The first thing to understand is that this is a tool window, so it can be moved and docked like any other tool window. Press Ctrl+F to bring up Quick Find.



The Quick Find drop-down menu lets you choose what type of find you want to do:

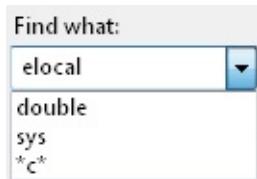


## Find What

For this discussion, we want to focus only on Quick Find, but each of these items comes with its own set of options. The Find What area is used to determine what you want to find:



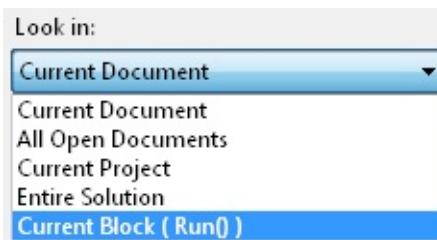
You can type what to look for in the drop-down combo box, or you can choose from the list of previous searches:



Don't worry about the arrow button to the right of the drop-down combo box (not shown here)—we will get to that later.

## Look In

Next is the Look In area. It's used to determine the scope of your search:

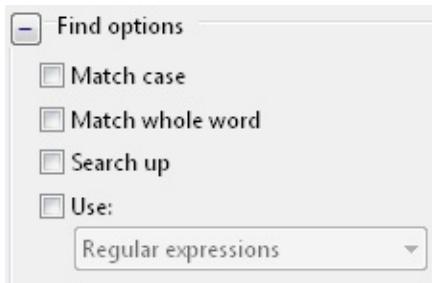


Most of the options are pretty self-explanatory, but we have a couple of key things to know, as follows:

1. The Current Project and Entire Solution actions search files whether they are open or closed.
2. Current Block is a little misleading because it doesn't search the current block but the entire method you are currently in.

## Find Options

The Find Options area is where the fun really happens:



Following is a run-down of these options:

### Match case

This option makes your search case-specific. Searching for elocal would show “elocal” but not “eLocal” or any other variant.

## Match whole word

By default, your search is a “contains” operation and therefore finds a result anywhere the word exists. For example, searching for “elocal” finds “elocal” and “elocalstuff”, and so on. This option restricts the search to only the word by itself. So, in this example, it finds “elocal” but *not* “elocalstuff.”

## Search up

Ordinarily, the search starts from the current cursor location and searches down in the current document. You can use this option to search up from the current cursor location instead.

## Use

This option is a *lot* more interesting and requires a bit of explanation. When you select this, you get to choose between Regular Expressions and Wildcards:



When you use this option, it automatically enables the Expression Builder button to the right of the Find What combo box:



## Regular expressions

Whether or not you are familiar with regular expressions, Visual Studio has its own syntax, so be aware of the differences. The following illustration shows what options the Expression Builder button provides you when Regular Expressions is selected.

```
. Any single character
* Zero or more
+ One or more
^ Beginning of line
$ End of line
< Beginning of word
> End of word
\n Line break
[ ] Any one character in the set
[^ ] Any one character not in the set
| Or
\ Escape special character
{ } Tag expression
:i C/C++ identifier
:q Quoted string
:b Space or Tab
:z Integer
Complete character list
```

#### NOTE

To see the details of the regular expression syntax available in Visual Studio, see the topic “Regular Expressions (Visual Studio)” at <http://msdn.microsoft.com/en-us/library/2k3te2cs.aspx>.

## Wildcards

These aren't as advanced as regular expressions but are more familiar to people. They allow you to use special characters to represent one or more letters. More information about wildcard searches can be found at <http://msdn.microsoft.com/en-us/library/afy96z92.aspx>. The following illustration shows what options the Expression Builder button provides when Wildcards is selected.

```
* Zero or more of any character
? Any single character
# Any single digit
[ ] Any one character in the set
[!] Any one character not in the set
\ Escape special character
Complete character list
```

## Buttons

Finally, we have the two buttons at the bottom of the Quick Find tool window:



### Find Next

Keeps going to the next instance of the search term you are looking for until it reaches your original starting point.

### Bookmark All

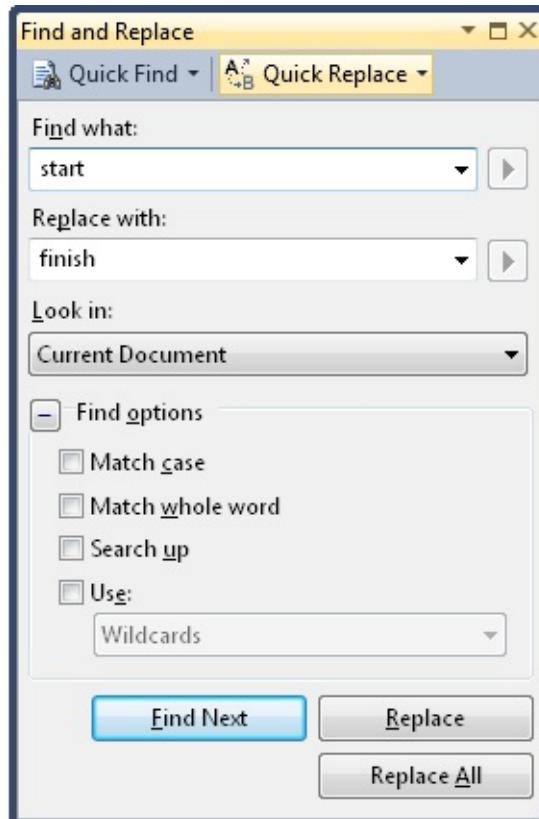
Automatically places a bookmark at every location where the search term is found. Use this with caution because it can definitely cause a large number of bookmarks to be created.

## 05.03 Using a Simple Quick Replace

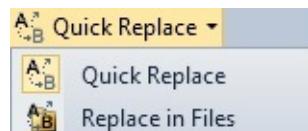
<b>DEFAULT</b>	Ctrl+H
<b>VISUAL BASIC 6</b>	Ctrl+H
<b>VISUAL C# 2005</b>	Ctrl+H
<b>VISUAL C++ 2</b>	Ctrl+H
<b>VISUAL C++ 6</b>	Ctrl+H
<b>VISUAL STUDIO 6</b>	Ctrl+H
<b>WINDOWS</b>	Alt, E, F, R
<b>MENU</b>	Edit   Find and Replace   Quick Replace
<b>COMMAND</b>	Edit.Replace
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0008

Previously, we looked at Quick Find, and now we will look at Quick Replace. They are almost exactly the same except for the replace operation itself. For that reason, I will not repeat all the options here but refer you back to vtipFind0007 ([05.02 Using Quick Find](#), page 172), for most of the details.

Press Ctrl+H to bring up Quick Replace:



This is a tool window, so it can be docked like any other tool window, pretty much anywhere you want. Notice that the Quick Replace drop-down menu lets you choose what type of replace you want to do:



For this discussion, let's focus only on Quick Replace. The Find What area is used to determine what you want to find:



The Replace With area functions exactly the same way, but it takes the text that you want to be used to replace the Find What text with:



The Look In and Find Options function the same way as using Quick Find and are explained in vtipFind0007 ([05.02 Using Quick Find](#), page 172).

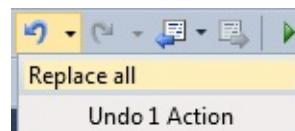
Finally, we have the three buttons at the bottom of the Quick Replace tool window:



- **Find Next**—Selects the next instance of the search term you are looking for until it reaches your original starting point.
- **Replace**—Replaces the currently selected item from Find What by using the text in Replace With.
- **Replace All**—Replaces all instances of Find What by using the text in Replace With. This produces a dialog box that shows how many replacements were made:



Make sure to pay attention to this value because it might be higher (or lower) than expected and might require further investigation. If you make a mistake, you can always undo a Replace All:



## 05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match

<b>WINDOWS</b>	Alt, T, O
<b>MENU</b>	Tools   Options   Environment   Find and Replace
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0010

When using the Quick Find or the Quick Replace tool window, you have an option to make the window disappear after the first match. This can be useful when you want to use your shortcut keys after the first match is found.

Just go to Tools | Options | Environment | Find And Replace, and then select the Hide Find And Replace Window After A Match Is Located For Quick Find Or Quick Replace check box.

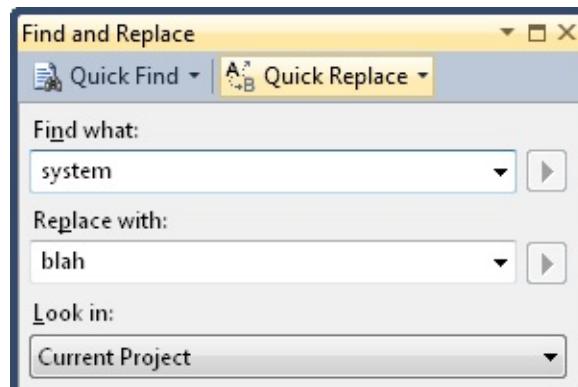
## 05.05 Undo Quick Replace and Replace in Files

<b>DEFAULT</b>	Ctrl+Z; Alt+Backspace
<b>VISUAL BASIC 6</b>	Ctrl+Z; Alt+Backspace
<b>VISUAL C# 2005</b>	Ctrl+Z; Alt+Backspace
<b>VISUAL C++ 2</b>	Ctrl+Z; Alt+Backspace
<b>VISUAL C++ 6</b>	Ctrl+Z; Alt+Backspace
<b>VISUAL STUDIO 6</b>	Ctrl+Z; Alt+Backspace
<b>WINDOWS</b>	Alt,E, U
<b>MENU</b>	Edit   Undo
<b>COMMAND</b>	Edit.Undo
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0020

When using Find And Replace, people often wonder under what conditions you can undo the changes. Because performing a Find Next and Replace operation is very straightforward and easy to undo, let's focus on how to undo the **Replace All** operations. Following is a summary of the conditions that allow undo to happen.

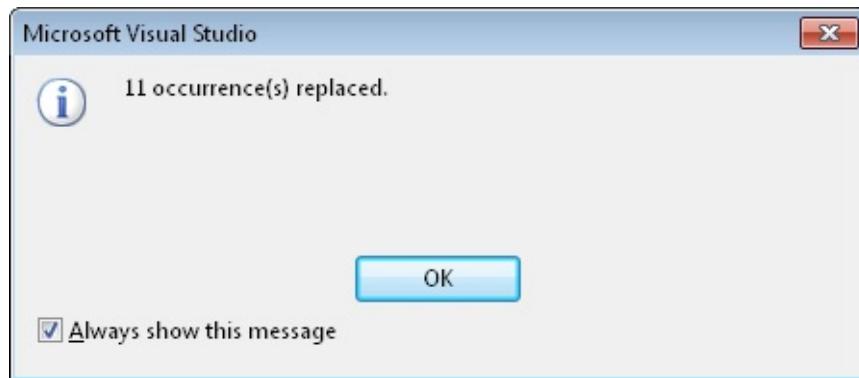
### Quick Replace (Ctrl+H)

Let's assume we're choosing the Quick Replace option with the Look In option set to Current Project or Entire Solution:

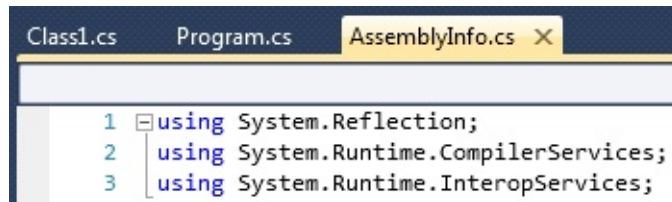


A Replace All operation, by default, opens documents and marks them by putting

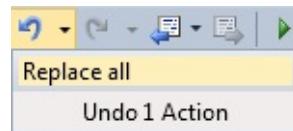
an asterisk in the file name tab so that you can undo the changes:



When we undo (Ctrl+Z), it reverses *all* the changes made. In this case, it undoes all eleven changes:

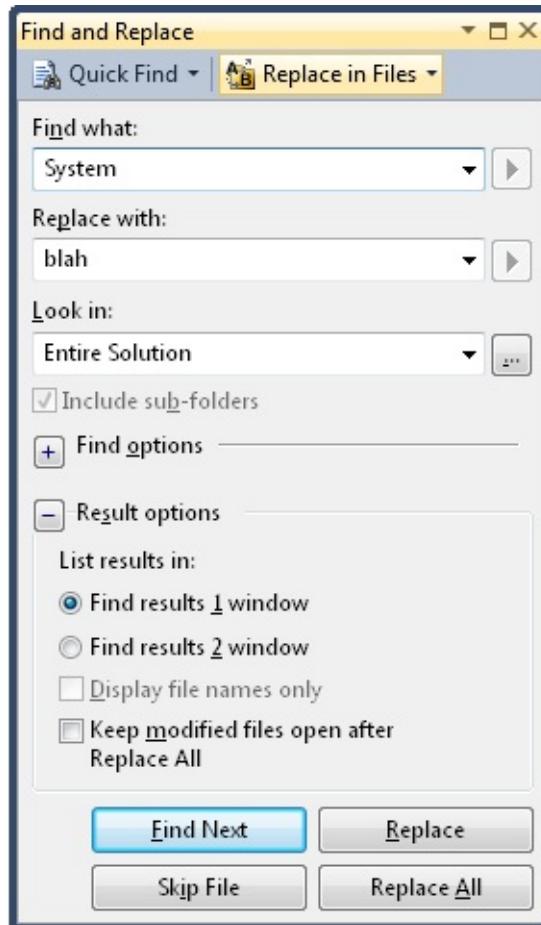


From the toolbar, the following entry appears in the undo stack:

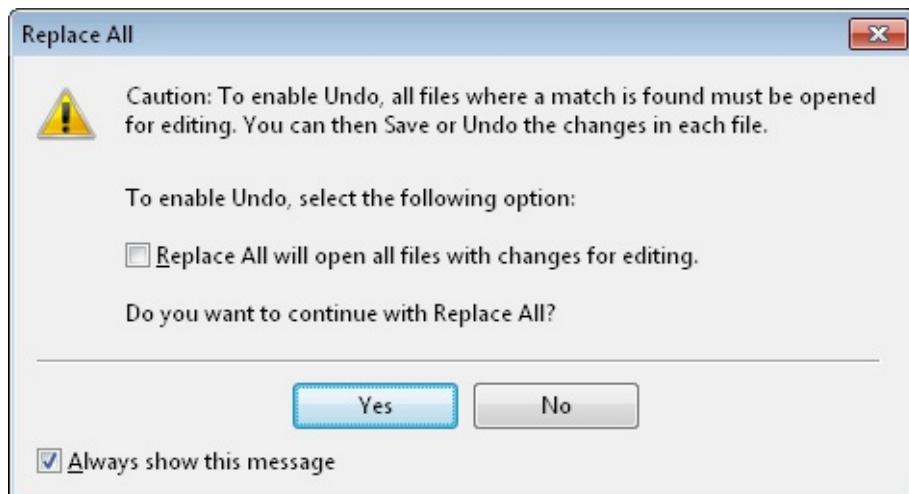


## Replace in Files (Ctrl+Shift+H)

If we perform a Replace in Files operation with Look In set to Current Project, Entire Solution, or Visual C++ Include Directories, we see the following dialog box:



As shown in the preceding illustration, notice the option called Keep Modified Files Open After Replace All. If we do not select this option and click Replace All, we get the following message:

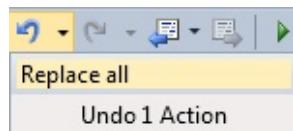


Checking the Replace All Will Open All Files With Changes For Editing makes it possible to undo *all* the changes made:

Find Results 1 < Class1.cs\* Program.cs\* AssemblyInfo.cs\*

Replace all "System", "blah", Subfolders, Find Results 1, "Entire Solution"  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Class1.cs()  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Class1.cs()  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Class1.cs()  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Class1.cs()  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Program.cs  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Program.cs  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Program.cs  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Program.cs  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Properties  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Properties  
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication9\ConsoleApplication9\Properties  
Total replaced: 11 Matching files: 3 Total files searched: 3

The entry in the undo stack on the toolbar looks like this:



So, as you can see, it is quite easy to undo the changes made by Quick Replace and Replace in Files.

## 05.06 Using the Find Combo Box Keyboard Shortcuts

<b>DEFAULT</b>	Ctrl+D (find); Ctrl+/ (run command); Ctrl+G (go to line); Ctrl+Shift+G (go to file); F9 (set breakpoint)
<b>VISUAL BASIC 6</b>	[no shortcut] (find); [no shortcut] (run command); [no shortcut] (go to line); Ctrl+Shift+G (go to file); F9 (set breakpoint)
<b>VISUAL C# 2005</b>	Ctrl+/ (find); [no shortcut] (run command); Ctrl+G (go to line); Ctrl+Shift+G (go to file); F9 (set breakpoint)
<b>VISUAL C++ 2</b>	Ctrl+F (find); ALT+A (find); Ctrl+D (find); Ctrl+/ (run command); Ctrl+G (go to line); Ctrl+Shift+G (go to file); F9 (set breakpoint); Ctrl+Shift+F9 (set breakpoint)
<b>VISUAL C++ 6</b>	Ctrl+D (find); Ctrl+/ (run command); Ctrl+G (go to line); Ctrl+Shift+G (go to file); F9 (set breakpoint)
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+F (find); Ctrl+/ (run command); Ctrl+G (go to line); Ctrl+Shift+G (go to file); F9 (set breakpoint)
<b>COMMAND</b>	Edit.GoToFindCombo; Tools.GoToCommandLine; Edit.GoTo; Edit.OpenFile; Debug.ToggleBreakpoint
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0019

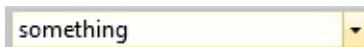
We have looked at the Find Combo box in a variety of places. Recall that this toolbar item is located on the standard toolbar by default:



I thought it would be a good idea to consolidate the keyboard shortcuts of this wonderful tool into one place.

### Find (Ctrl+D)

First and foremost, the Find / Command box is used to find strings. Just press Ctrl+D to go to the box and type in your search term:



When you press Enter, it searches for your string by using the settings from Quick Find (Ctrl+F) as it searches. Pressing Enter again finds the next instance, and so on.

## Run Command (Ctrl+ /)

The Find / Command box is also used to run commands. For more information about commands, see [vstipTool0067](#) ([03.19 Understanding Commands: Simple Commands](#), page 110). Just press Ctrl+/ to go to the box, and type in your command:



When you press Enter, it runs the command you typed.

## Go To Line (Ctrl+G)

When in the Find / Command box, you can type any line number:



Then press Ctrl+G, and you are taken to the line number that you entered.

## Go To File (Ctrl+Shift+G)

Type in any file name that is in your solution or in the INCLUDE path:



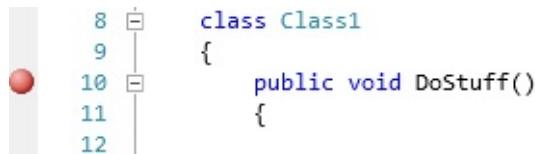
Use Ctrl+Shift+G to go to the file. If the file isn't already open, this command opens the file first.

## Set a Breakpoint (F9)

Enter any function name:



Then press F9, and Visual Studio sets a breakpoint on the function:



A couple of things to note:

- Pressing F9 again does *not* turn off the breakpoint—that is, it's not a toggle.

- This feature works only with open documents.

## 05.07 Using Incremental Search

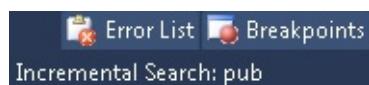
<b>DEFAULT</b>	Ctrl+I
<b>VISUAL BASIC 6</b>	Alt+I
<b>VISUAL C# 2005</b>	Ctrl+I
<b>VISUAL C++ 2</b>	Ctrl+I
<b>VISUAL C++ 6</b>	Ctrl+I
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt,E, V, S
<b>MENU</b>	Edit   Advanced   Incremental Search
<b>COMMAND</b>	Edit.IncrementalSearch
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0001

Incremental search is a powerful feature to use when you want to keep your cursor in the editor while searching in the current document. It allows you to keep your hands on the keyboard without having to use the mouse for any dialog boxes.

To conduct an incremental search, press Ctrl+I and start typing the text you are searching for. You'll see the cursor in the editor jump to the first match, highlighting the current search string, and your mouse cursor turns into a pair of binoculars with an arrow pointing in the direction (up or down) you are searching:

```
// i love public tips
private int x = 10;           ↴
public int y = 0;
```

If you look at the status bar, you can see the details of your incremental search:



Press Ctrl+I again to jump to the next occurrence of the search string:

```
// i love public tips
private int x = 10;    ↴
public int y = 0;
```

The following table lists all the options you can leverage while in this mode:

ACTION	SHORTCUT
Move to the next match in the file	Ctrl+I
Reverse the direction of the search	Ctrl+Shift+I
Remove a character from the search string	Backspace
Stop the incremental search	Esc

## 05.08 Search the Currently Selected String Without the Find Window

<b>DEFAULT</b>	Ctrl+F3 (next); Ctrl+Shift+F3 (previous)
<b>VISUAL BASIC 6</b>	Ctrl+F3 (next); Ctrl+Shift+F3 (previous)
<b>VISUAL C# 2005</b>	Ctrl+F3 (next); Ctrl+Shift+F3 (previous)
<b>VISUAL C++ 2</b>	Ctrl+F3 (next); Ctrl+Shift+F3 (previous)
<b>VISUAL C++ 6</b>	Ctrl+F3 (next); Ctrl+Shift+F3 (previous)
<b>VISUAL STUDIO 6</b>	Ctrl+F3 (next); Ctrl+Shift+F3 (previous)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.FindNextSelected; Edit.FindPreviousSelected
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0003

Ever just want to find the next (or previous) instance of a word quickly without using the Quick Find dialog box? Well, it's easy.

Put the cursor in any word you want to look for:

```
public double x = 10;
public double y = 30;
public double z = 50;
public double a = 30;
```

Press Ctrl+F3 (next) or Ctrl+Shift+F3 (previous) to start a find:

```
public double x = 10;
public double| y = 30;
public double z = 50;
public double a = 30;
```

You can also see a message in the status bar showing you the parameters of the search:



**NOTE**

As you can see, the find has certain settings already in place. These settings can be changed in the Find dialog box (Ctrl+F).

After the Find has started, just press F3 (next) or Shift+F3 (previous) to continue searching as you normally would with a Quick Find.

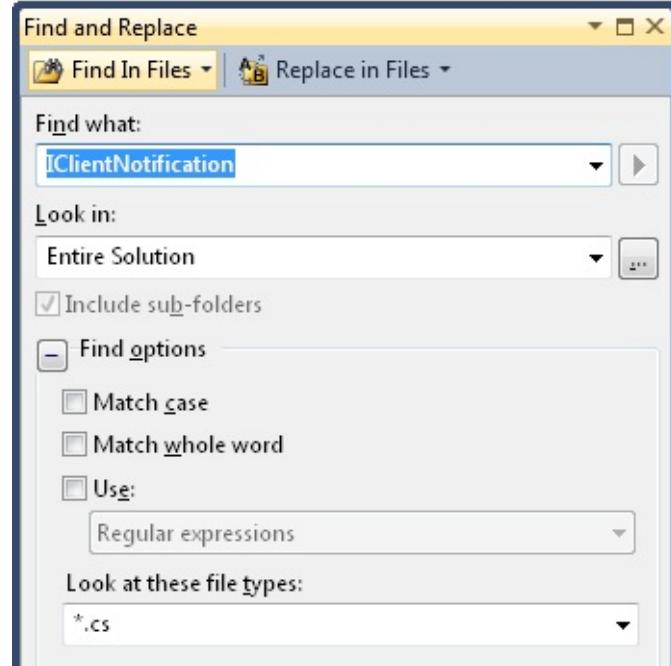
## 05.09 Find In Files: Find Options

<b>DEFAULT</b>	Ctrl+Shift+F
<b>VISUAL BASIC 6</b>	Ctrl+Shift+F
<b>VISUAL C# 2005</b>	Ctrl+Shift+F
<b>VISUAL C++ 2</b>	Ctrl+Shift+F
<b>VISUAL C++ 6</b>	Ctrl+Shift+F
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt,E, F, I
<b>MENU</b>	Edit   Find and Replace   Find in Files
<b>COMMAND</b>	Edit.FindinFiles
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0013

When working in Visual Studio, you often need to search for information in files. Find In Files allows you to quickly locate information you need. Let's begin by looking at the options you can set to find information in files (Ctrl+Shift+F).

### NOTE

If you have already read vstipFind0007 ([05.02 Using Quick Find](#), page 172), you might want to just skim this one because much of the information is repeated here for those who might not have read the prior tip.

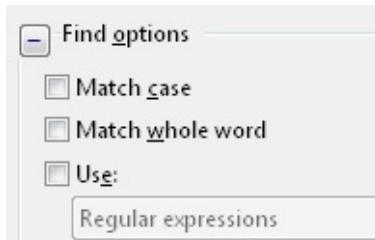


## Find What

The Find What combo box lets you type in text to find or choose from previous text that has been searched:



This area is very closely bound to all but one of the options under Find Options:



Following is an overview of these options:

### Match case

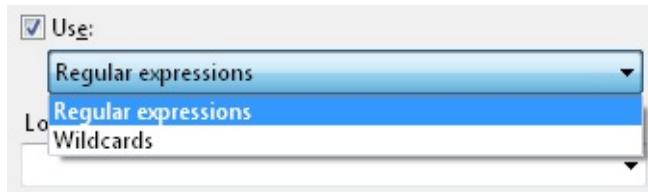
This option makes your search case-specific. Searching for elocal would show “elocal” but not “eLocal” or any other variant.

## Match whole word

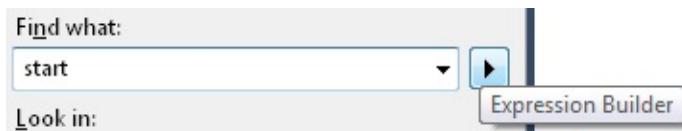
By default, the search is a “contains” operation and finds the word anywhere it exists. For example, searching for “elocal” finds “elocal” and “elocalstuff”, and so on.

## Use

This is a *lot* more interesting and requires a bit of explanation. When you select this, you get to choose between Regular Expressions and Wildcards:



When you use this option, it automatically enables the Expression Builder button to the right of the Find What combo box:



## Regular expressions

Whether or not you are familiar with regular expressions, Visual Studio has its own syntax, so be aware of the differences. The following illustration shows the options provided by the Expression Builder button when you have selected Regular Expressions:

.	Any single character
*	Zero or more
+	One or more
^	Beginning of line
\$	End of line
<	Beginning of word
>	End of word
\n	Line break
[ ]	Any one character in the set
[^ ]	Any one character not in the set
	Or
\	Escape special character
{ }	Tag expression
:i	C/C++ identifier
:q	Quoted string
:b	Space or Tab
:z	Integer
Complete character list	

#### NOTE

To see the details of the regular expression syntax available in Visual Studio, see the topic “Regular Expressions (Visual Studio)” at <http://msdn.microsoft.com/en-us/library/2k3te2cs.aspx>.

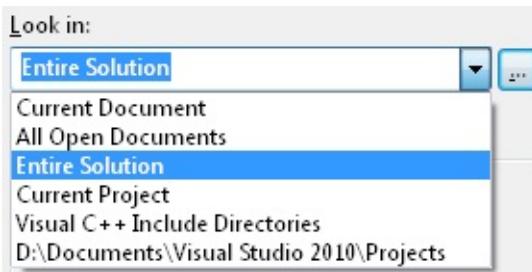
## Wildcards

This option isn’t as advanced as regular expressions but is more familiar to most people. It allows you to use special characters to represent one or more letters. For more information about wildcard searches, see “Wildcards (Visual Studio) at <http://msdn.microsoft.com/en-us/library/afy96z92.aspx>. The following illustration shows the options provided by the Expression Builder button when Wildcards is selected:

*	Zero or more of any character
?	Any single character
#	Any single digit
[ ]	Any one character in the set
[!]	Any one character not in the set
\	Escape special character
Complete character list	

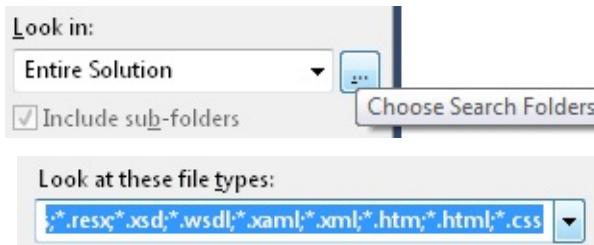
## Look in

The Look in drop-down list lets you specify what areas you want to look in:



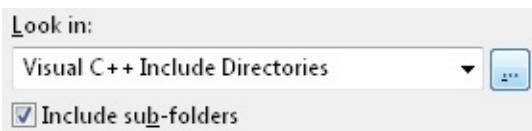
Most of the options are pretty self-explanatory. The Current Project and Entire Solution commands search files whether they are open or closed.

The Choose Search Folders and Look At These File Types options are discussed in vtipFind0005 ([AX.47 Customize the Files to Search with Find In Files](#), in Appendix B [<http://go.microsoft.com/fwlink/?LinkId=223758>]):



## Include sub-folders

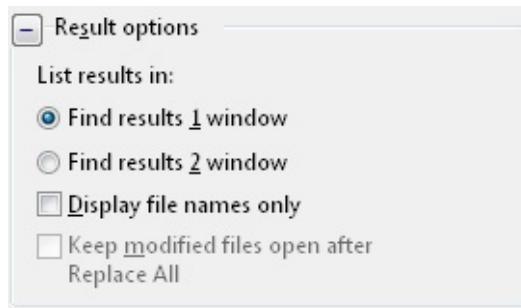
This option searches the current directory and all subdirectories and shows up only if you use the Visual C++ Include Directories or Choose Search Folders options:



## 05.10 Find In Files: Result Options

<b>DEFAULT</b>	Ctrl+Shift+F (find); F8 (next); Shift+F8 (previous);
<b>VISUAL BASIC 6</b>	Ctrl+Shift+F (find);[no shortcut] (next); [no shortcut] (previous);
<b>VISUAL C# 2005</b>	Ctrl+Shift+F (find); F8 (next); Shift+F8 (previous);
<b>VISUAL C++ 2</b>	Ctrl+Shift+F (find); F4 (next); Shift+F4 (previous);
<b>VISUAL C++ 6</b>	Ctrl+Shift+F (find); F8 (next); F4 (next); Shift+F8 (previous); Shift+F4 (previous);
<b>VISUAL STUDIO 6</b>	[no shortcut] (find); F8 (next); F12 (next); Shift+F8 (previous); Shift+F12 (previous);
<b>WINDOWS</b>	Alt,E, F, I (find)
<b>MENU</b>	Edit   Find and Replace   Find in Files
<b>COMMAND</b>	Edit.FindinFiles; Edit.GoToNextLocation; Edit.GoToPrevLocation; Edit.ClearFindResults1; Edit.ClearFindResults2
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0014

When working with Find In Files (Ctrl+Shift+F), you can choose several result options:



## Find Results [1,2] Window

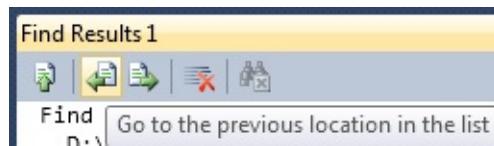
The Find Results windows allow you to view and navigate the results of a find operation. Each time you use Find, the results replace the contents of the previous find. This is why two windows are available—so that you avoid overwriting a

find result you might want to keep.

## Navigation

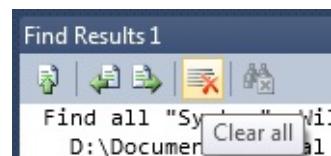
You can use F8 and Shift+F8 to go to the next and previous items in the results list. This operation also shows the line of code where the item was found, which includes opening closed files if needed:

The Find Results windows have toolbar buttons that allow you to go to the next and previous items as well:



## Clear All

You can manually clear the results in a Find Results window by clicking Clear All:



## Display File Names Only

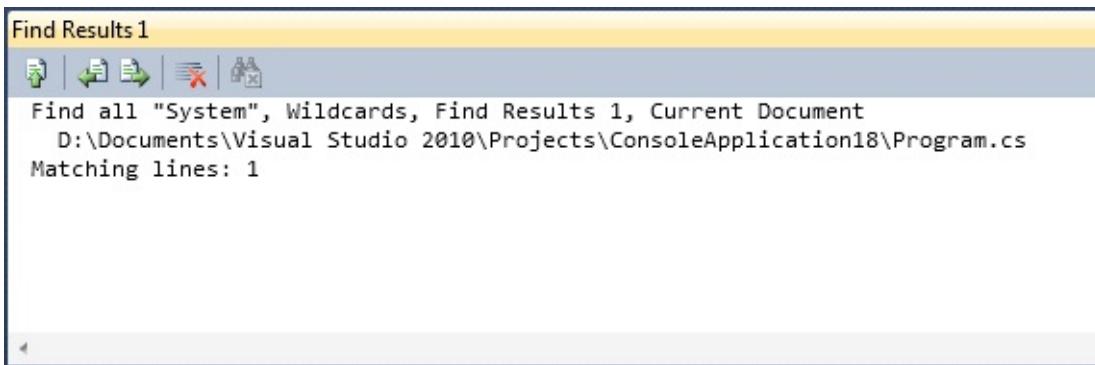
The operation shows only the file names in your results and not the full path and additional information. This means the result set is much smaller.

Before:

A screenshot of the 'Find Results' window. The title bar says 'Find Results 1'. The toolbar has the standard icons. The status bar at the bottom says 'Find all "System"', 'Wildcards', 'Find Results 1', 'Current Document', and 'Matching lines: 4'. The main window lists four lines of code from 'Program.cs':

```
Find all "System", Wildcards, Find Results 1, Current Document
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication18\Program.cs(1):using System;
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication18\Program.cs(2):using System.Collections.Generic;
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication18\Program.cs(3):using System.Linq;
D:\Documents\Visual Studio 2010\Projects\ConsoleApplication18\Program.cs(4):using System.Text;
```

After:



You can do much more with the displayed results than this. See vstipFind0002 ([05.17 Customize Results in Find In Files Searches](#), page 206) for more information.

## Keep Modified Files Open After Replace All

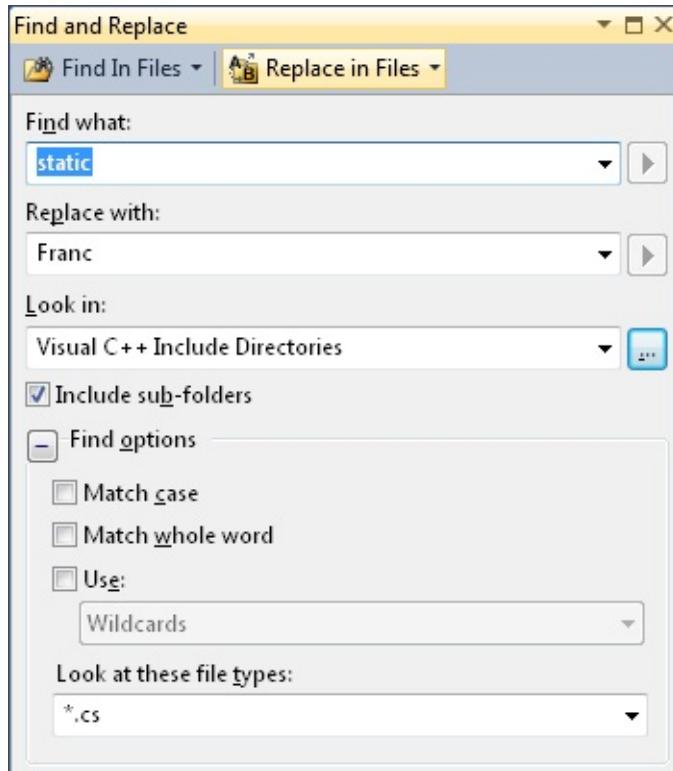
This option doesn't apply to Find In Files and is always disabled when doing a find operation. You can see this option when using the Replace In Files option.

## 05.11 Replace In Files: Basic Options

<b>DEFAULT</b>	Ctrl+Shift+H
<b>VISUAL BASIC 6</b>	Ctrl+Shift+H
<b>VISUAL C# 2005</b>	Ctrl+Shift+H
<b>VISUAL C++ 2</b>	Ctrl+Shift+H
<b>VISUAL C++ 6</b>	Ctrl+Shift+H
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+H
<b>WINDOWS</b>	Alt,E, F, S
<b>MENU</b>	Edit   Find and Replace   Replace in Files
<b>COMMAND</b>	Edit.ReplaceinFiles
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0015

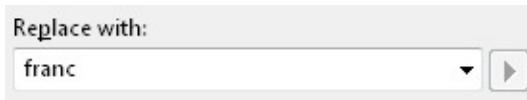
Did you know that you can replace text in files, whether or not they are open, by choosing Replace In Files (Ctrl+Shift+H)? Let's take a look at what can be done.

### Find Options



Fortunately, the majority of find options are the same for Replace In Files as they are for Find In Files (see vstipFind0013, [05.09 Find In Files: Find Options](#), on page 186), so you can leverage those skills again here.

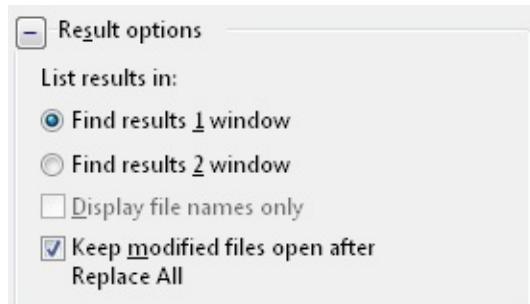
## Replace With



This area is the most interesting piece of the Replace In Files dialog box. It can be as simple as a literal string (replacing “static” with “Franc”, for example) or very, very complex. In simple situations, you just want to use the literal text as a replacement to the Find What text.

## Result Options

Again, these options are exactly like the Find In Files options (vstipTool0014, “Find In Files: Result Options,” page 158) with the exception of the Keep Modified Files Open After Replace All option:



This option makes it possible to keep modified files open after they are changed so that you can review the change or make additional manual changes. This is particularly useful when you are modifying closed files and want to look inside files that were modified.

## Execution

After all the options have been set, we can execute find and replace operations by using the following four buttons:



### Find Next

Used to find the next instance of the Find What search string.

### Replace

Used to replace the current instance of the Find What string with the Replace With string and then find the next instance.

### Replace All

Used to replace all instances of the Find What string with the Replace With string, in all files within the Look In scope.

#### WARNING

The Replace All option can get you into big trouble if you don't pay attention to the scope of Look In.

### Skip File

Available when the Look In list includes multiple files. Choose this button if you do not want to search or modify the current file. The search continues in the next file on the Look In list.

## 05.12 Go To Definition for Cascading Style Sheets

<b>DEFAULT</b>	F12
<b>VISUAL BASIC 6</b>	F12; Shift+F2
<b>VISUAL C# 2005</b>	F12
<b>VISUAL C++ 2</b>	F11; Alt+F1
<b>VISUAL C++ 6</b>	F12
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.GoToDefinition
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipFind0021

For those who are familiar with using Go To Definition in your code, you might not be aware that you can use the same technique to go to your Cascading Style Sheet (CSS) definition class for attributes. Just put your cursor inside the class name:

```
<div class="bold">  
    something cool  
</div>
```

Then press F12 (or right-click and choose Go To Definition). It instantly takes you to the CSS definition and highlights it:

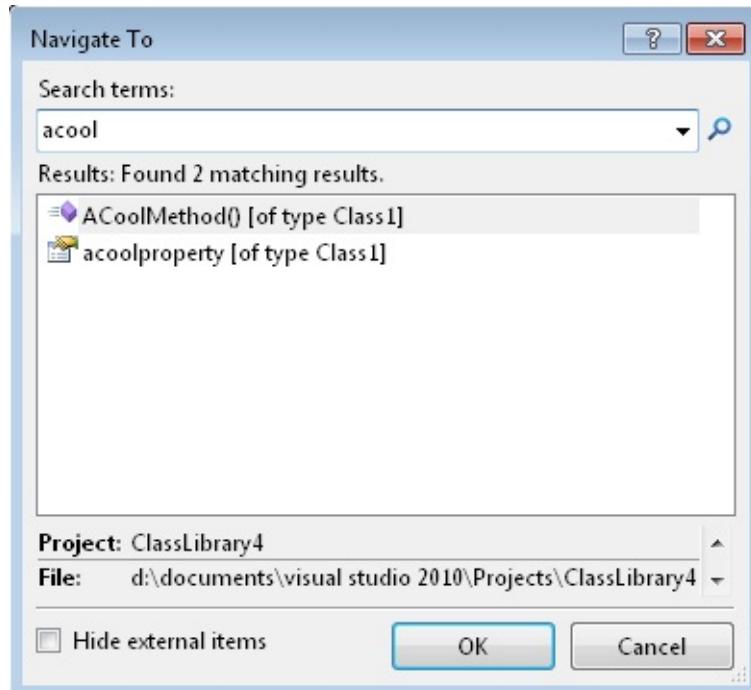
```
.bold  
{  
    font-weight: bold;  
}
```

Now you can review the definition and make changes as you see fit.

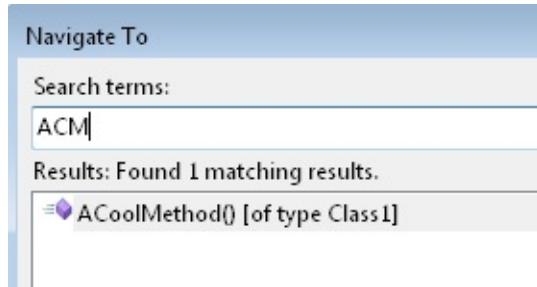
## 05.13 How to Use Navigate To

<b>DEFAULT</b>	Ctrl+, [comma]
<b>VISUAL BASIC 6</b>	Ctrl+, [comma]
<b>VISUAL C# 2005</b>	Ctrl+, [comma]
<b>VISUAL C++ 2</b>	Ctrl+, [comma]
<b>VISUAL C++ 6</b>	Ctrl+, [comma]
<b>VISUAL STUDIO 6</b>	Ctrl+, [comma]
<b>WINDOWS</b>	Alt,E, .[period]
<b>MENU</b>	Edit   Navigate To
<b>COMMAND</b>	Edit.NavigateTo
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0006

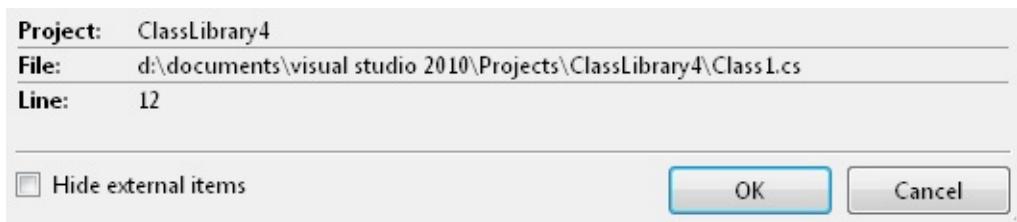
The Navigate To dialog box is very useful for finding symbols. The search is an “includes” operation, so it shows you symbols that contain the letters you type. Just put in what you are looking for:



Notice that the search is *not* case-specific. However, you might notice a surprise in this dialog box. Watch what happens when you put in **ACM**:



It pays attention to Pascal Case. There is also summary information at the bottom of this dialog box:



You're probably wondering about the Hide External Items option as well. When selected, only the local project is examined for symbols, instead of your project *plus* every library you reference.

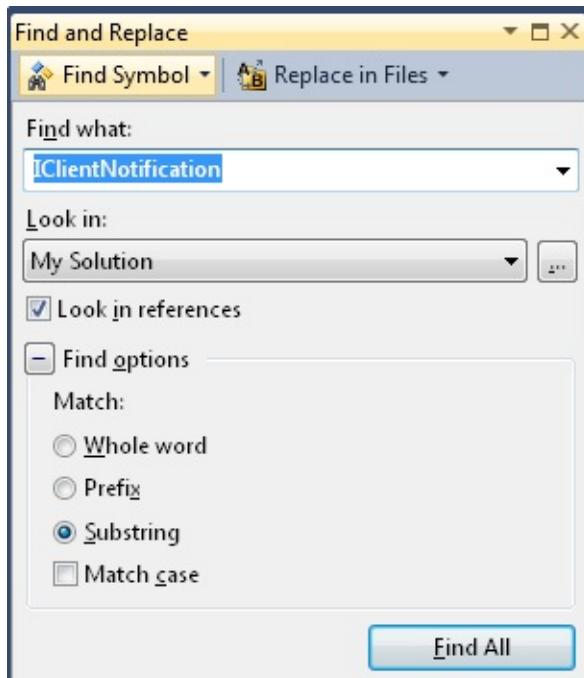
Also, notice that the Navigate To syntax does not support special logic or special characters such as the following:

- Wildcard matching
- Boolean logic operators
- Regular expressions

## 05.14 Understanding Find Symbol

<b>DEFAULT</b>	Alt+F12
<b>VISUAL BASIC 6</b>	Alt+F12
<b>VISUAL C# 2005</b>	Alt+F12
<b>VISUAL C++ 2</b>	Alt+F12; Ctrl+F11; Ctrl+Alt+F11
<b>VISUAL C++ 6</b>	Alt+F12; Ctrl+Shift+Y
<b>VISUAL STUDIO 6</b>	Alt+F12
<b>WINDOWS</b>	Alt,E, F, Y
<b>MENU</b>	Edit   Find and Replace   Find Symbol
<b>COMMAND</b>	Edit.FindSymbol
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0011

You can quickly search for symbols (objects, definitions, and references) by using the Find Symbol dialog box (Alt+F12):



## Find What

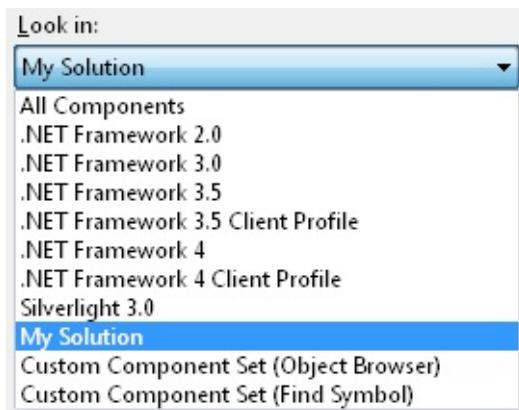
This is pretty straightforward; you just type in the search string you are looking for.

### NOTE

You can also get to Find Symbol by using any shortcut to get to the Find and Replace dialog box and selecting Find Symbol from the drop-down list in the upper-right corner.

## Look In

This indicates where you want to look:



As you can see, the Look In dialog box has a number of search options.

### All Components

Includes the current solution and its referenced components, all of the .NET Framework, and any components that you have added.

### Framework X / Silverlight X

Searches specific versions of the Framework for a symbol.

### My Solution

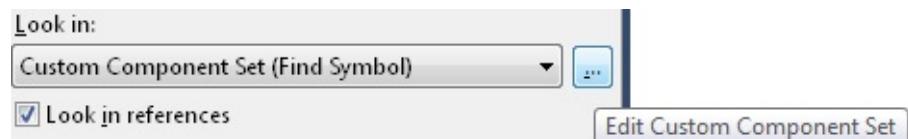
Searches the current open solution.

### Custom Component Set (Object Browser)

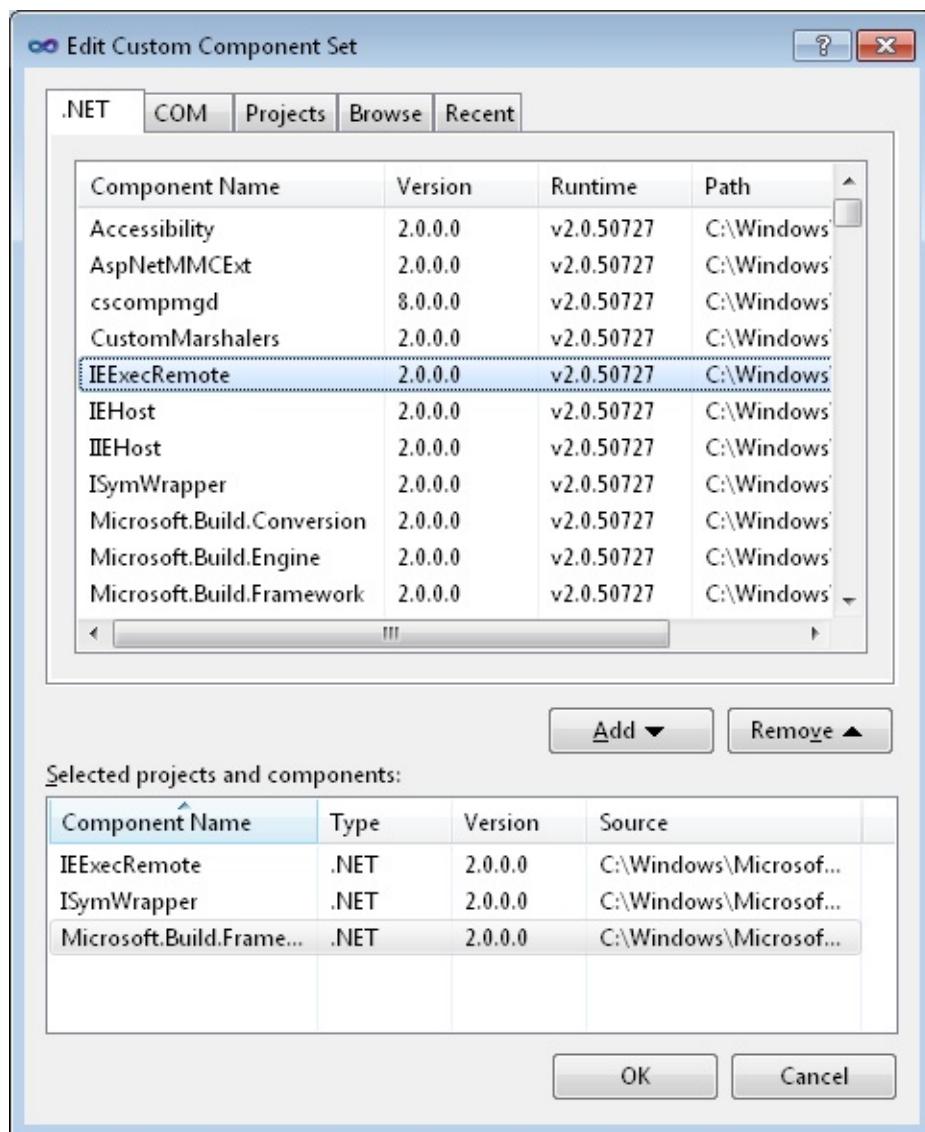
Searches the predefined custom component set in the Object Browser. See vstipTool0078 ([07.40 The Object Browser: Setting the Browsing Scope](#), page 358) for more information.

## Custom Component Set (Find Symbol)

Searches the custom component set defined in this dialog box. You edit the list of components by clicking the ellipsis in the dialog box:



The Edit Custom Component Set dialog box allows you to pick components from a variety of areas to have a specialized search experience when looking for symbols. It allows for a very refined search capability:



## Look In References

Displays references in the projects within the current browsing scope.

## Find Options

### Match

Sets criteria for the search string when finding matches.

#### Whole Word

Finds the complete word only, not partial matches.

#### Prefix

Finds results where the search string is at the beginning of the result.

#### Substring

Finds results where the search string is anywhere in the result.

#### Match Case

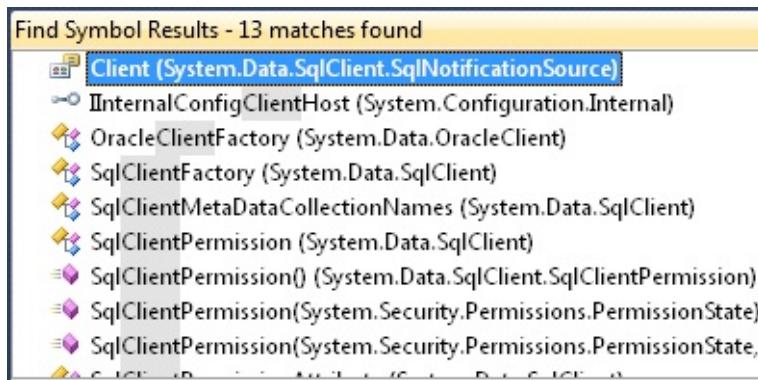
Finds only results that exactly match the case of the search string.

#### Find All

Initiates the search based on the criteria that has been set.

## Search Results

When you run a search, the results look something like this:



The icons to the left of each entry indicate what type of symbol you are looking at in the results. For a complete list of icons and their meanings, see vstipTool0076 ([AX.120 Class View and Object Browser Icons](#), in Appendix B [<http://go.microsoft.com/fwlink/?LinkId=223758>]) for more information.

Additionally, you can use F8 or Shift+F8 to navigate forward or backward through

the results while automatically showing the location of the result as you proceed.

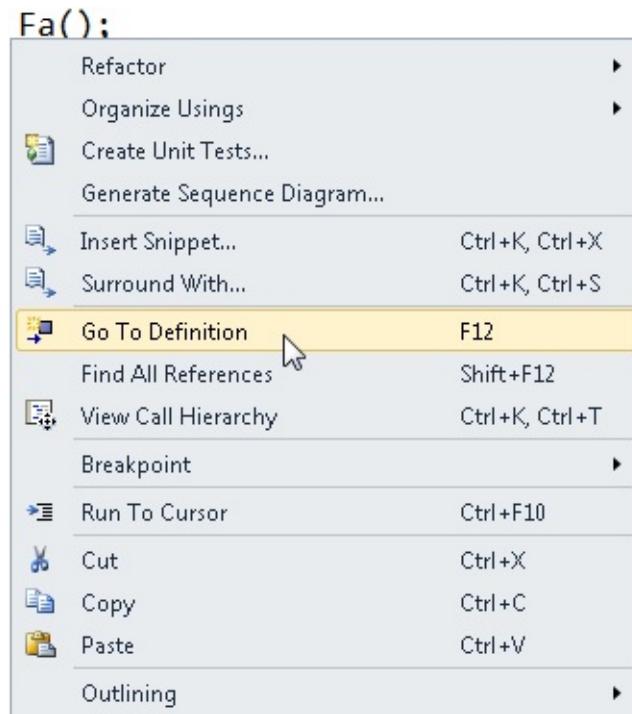
## 05.15 Find Symbol Results Shortcuts

<b>DEFAULT</b>	F12 (go to definition); Ctrl+F12 (go to declaration); Shift+F12 (find all references); Ctrl+C (copy); Ctrl+Insert (copy);
<b>VISUAL BASIC 6</b>	F12 (go to definition); Shift+F2 (go to definition); Ctrl+F12 (go to declaration); Alt+F2 (find all references); Ctrl+C (copy); Ctrl+Insert (copy);
<b>VISUAL C# 2005</b>	F12 (go to definition); Ctrl+F12 (go to declaration); Shift+F12 (find all references); Ctrl+K, Ctrl+R (find all references); Ctrl+K, R (find all references); Ctrl+C (copy); Ctrl+Insert (copy);
<b>VISUAL C++ 2</b>	F11 (go to definition); Alt+F1 (go to definition); [no shortcut] (find all references); Ctrl+C (copy); Ctrl+Insert (copy);
<b>VISUAL C++ 6</b>	F12 (go to definition); Ctrl+F12 (go to declaration); Ctrl+Alt+F12 (go to declaration); [no shortcut] (find all references); Ctrl+C (copy); Ctrl+Insert (copy);
<b>VISUAL STUDIO 6</b>	[no shortcut] (go to definition); Ctrl+F12 (go to declaration); [no shortcut] (find all references); Ctrl+C (copy); Ctrl+Insert (copy);
<b>WINDOWS</b>	Alt+E, C (copy)
<b>MENU</b>	Edit   Copy
<b>COMMAND</b>	Edit.GoToDefinition; Edit.GoToDeclaration; Edit.FindAllReferences;View.BrowseDefinition;Edit.Copy;Edit.ClearAll
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0012

To locate any symbol, you can quickly leverage a series of commands to help you. The nice thing about these commands is that many of them work both in the Find Symbol Results window and in the code editor.

### Go To Definition (F12)

This command takes you to the definition of the symbol in your code, if one is available.



```
public void Fa()
{
    So();
}
```

## Go To Declaration (Ctrl+F12)

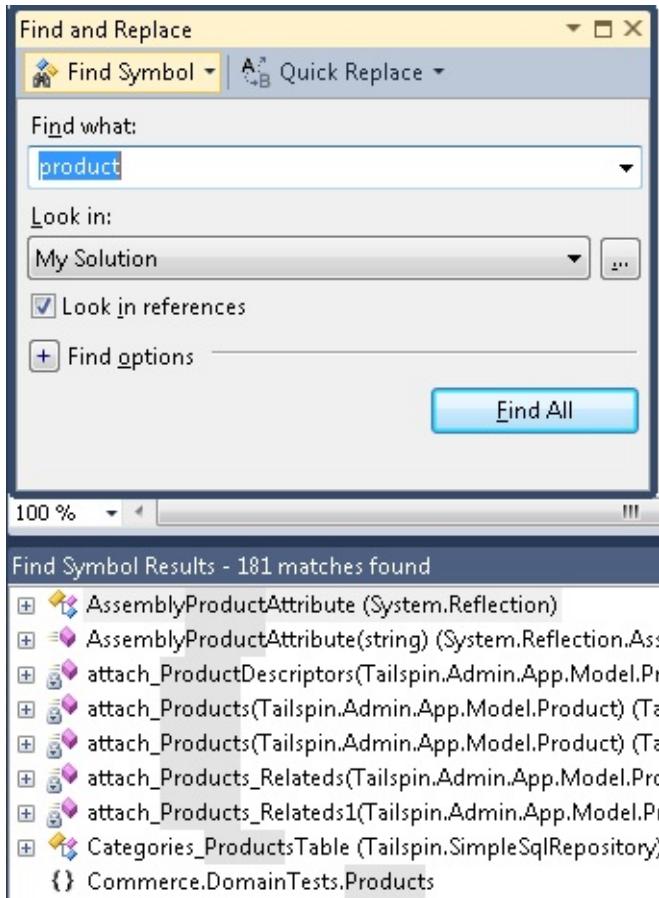
### NOTE

The information in this section applies to C++ only.

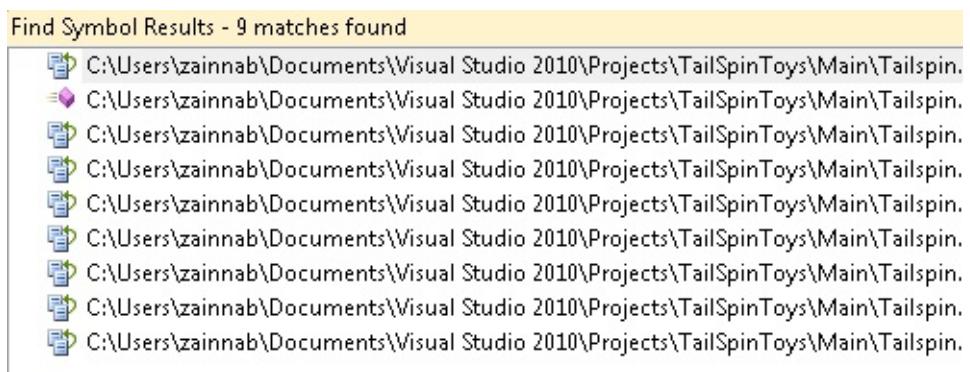
This command takes you to the declaration of the symbol in your code, if one is available.

## Go To Reference (Shift+F12)

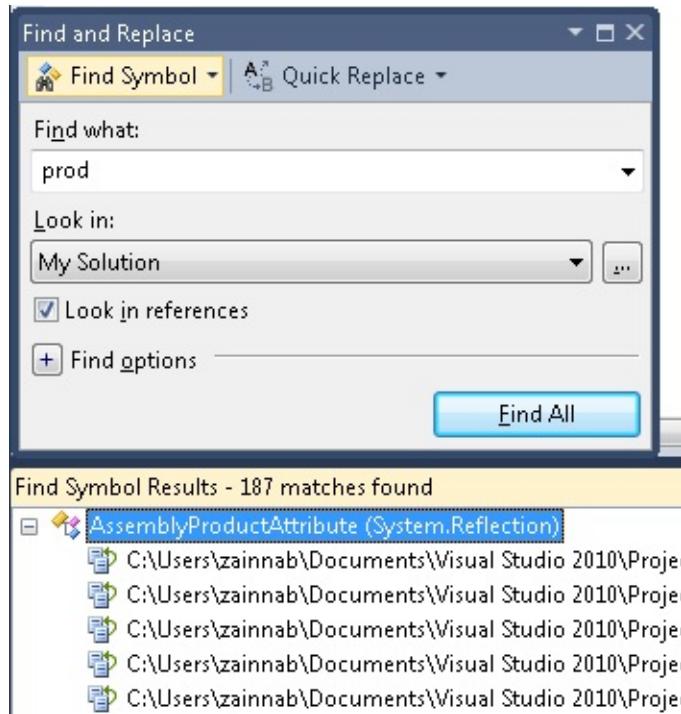
This command works for any symbol:



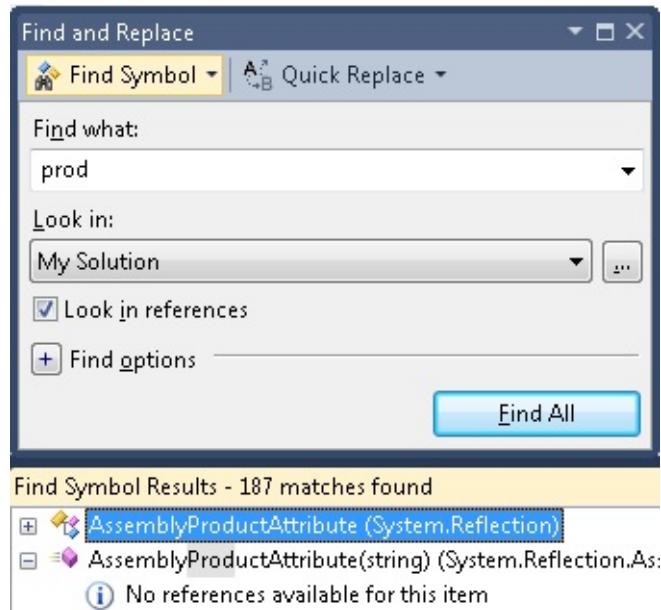
This command finds all references to that symbol:



If you are using the Find Symbol Results window, you can make this much easier by simply expanding a node in the window. It automatically shows all references:

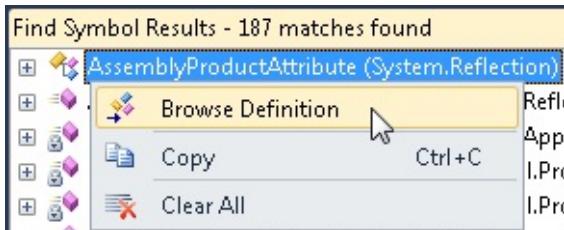


If no references are found, it tells you that also:

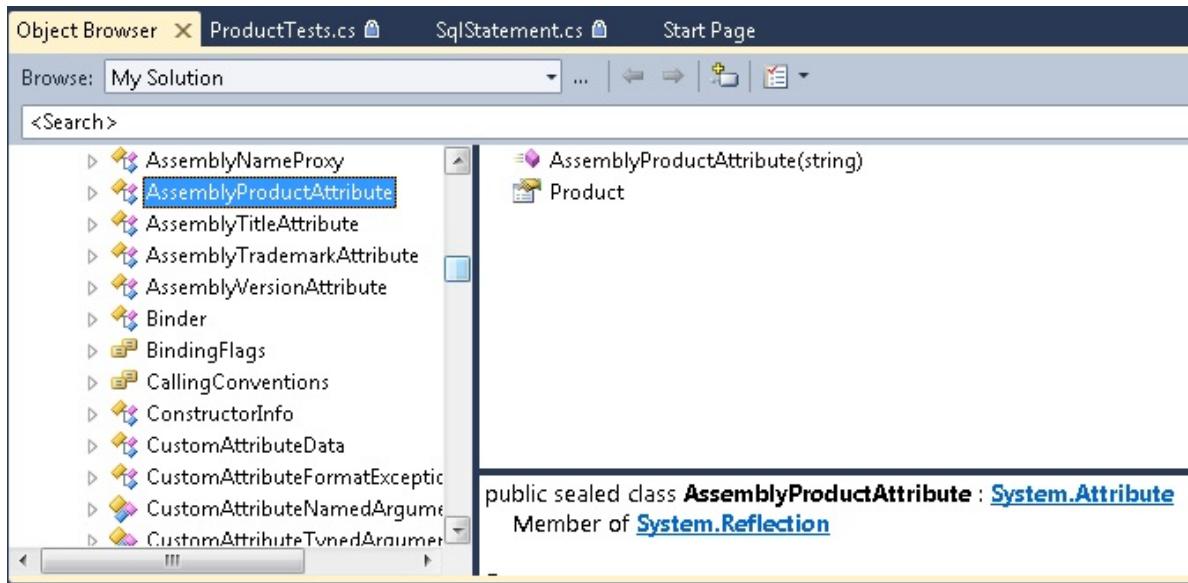


## Browse Definition

For any symbol in your results, simply right-click and choose Browse Definition:



Clicking Browse Definition takes you to the primary node (typically top level) for the symbol in the Object Browser, which can be particularly useful for deeper examination:



## Copy (Ctrl+C)

Allows you to copy the fully qualified name for the selected symbol to the clipboard. You can then paste the code as text into the code editor.

## Clear All

Clears the Find Symbol Results window.

## 05.16 Replace in Files: Tagged Expressions

<b>DEFAULT</b>	Ctrl+Shift+H
<b>VISUAL BASIC 6</b>	Ctrl+Shift+H
<b>VISUAL C# 2005</b>	Ctrl+Shift+H
<b>VISUAL C++ 2</b>	Ctrl+Shift+H
<b>VISUAL C++ 6</b>	Ctrl+Shift+H
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+H
<b>WINDOWS</b>	Alt,E, F, S
<b>MENU</b>	Edit   Find and Replace   Replace in Files
<b>COMMAND</b>	Edit.ReplaceinFiles
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0016

As mentioned in vstipFind0015 ([05.11 Replace In Files: Basic Options](#), page 192), the Replace With area is the most interesting piece of the Replace In Files dialog box. It can be as simple as a literal string replacement:



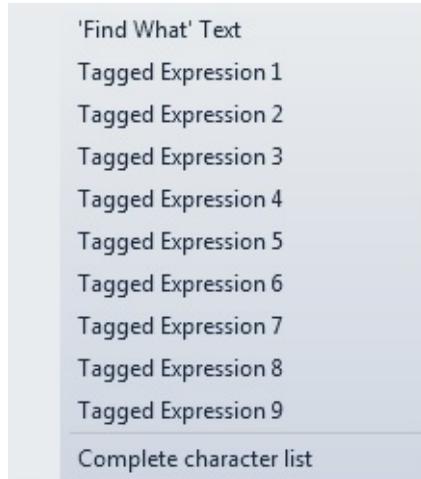
Under normal situations, this is just the literal text you want to use as a replacement for the Find What text. However, suppose you choose to use regular expressions:



This enables the Expression Builder:



These options are *not* like the builder options you get in the Find What area:



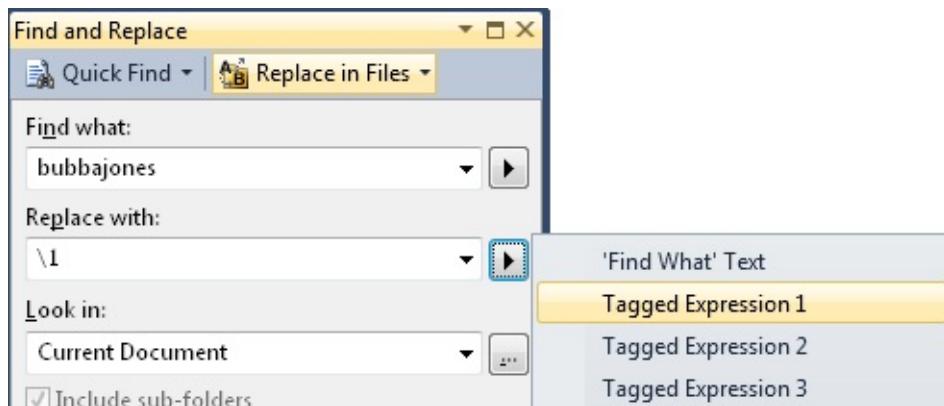
In addition to being able to use any of the regular expression characters, you can refer to the original text and any tagged expressions.

## Example

The best way to show how tagged expressions work is with an example. Let's assume you have the following text:

```
static void Main(string[] args)
{
    // bubbajones
    // bubbajones
    // bubbajones
    // bubbajones
    // bubbajones
    // bubbajones|
}
```

And you use the following options:



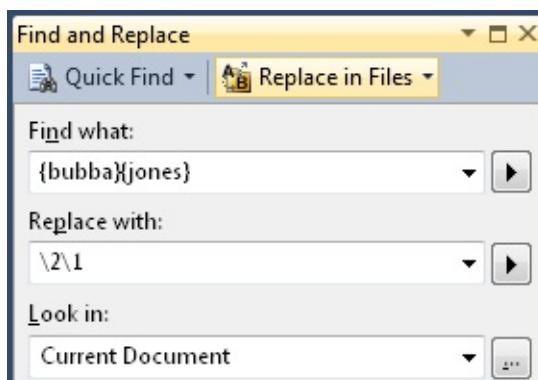
Notice the curly brackets around “jones”? That denotes a tagged expression. Every time you use the brackets, it creates a tagged expression that is numbered

(beginning with 1). So, in this example, we are looking for “bubba{jones}” and replacing it with tagged expression 1 (which is just “jones”). Also, notice the notation used to refer to the tagged expression: \n, where n is the tagged expression we want.

When I do my replacements, this is the result:

```
static void Main(string[] args)
{
    // jones
    // jones
    // jones
    // jones
    // jones
    // jones
}
```

We can take this further. Now we want to turn “bubbajones” into “jonesbubba,” so we can use the following settings:



So now we are looking for “{bubba}{jones},” which creates tagged expression 1 (“bubba”) and tagged expression 2 (“jones”). At this point, it’s simply a matter of replacing with the expressions switched around (“\\2\\1”), and we get the following:

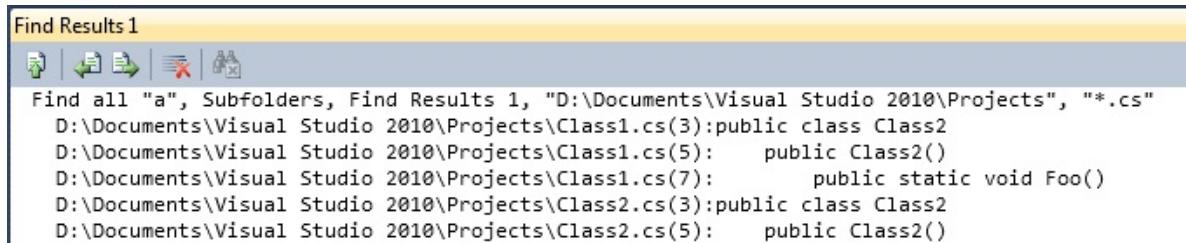
```
static void Main(string[] args)
{
    // bubbajones
    // bubbajones
    // bubbajones
    // bubbajones
    // bubbajones
    // bubbajones
}
```

Of course, this can get much more complex when using regular expressions, so you definitely need to spend some time learning how to fully leverage these features.

## 05.17 Customize Results in Find In Files Searches

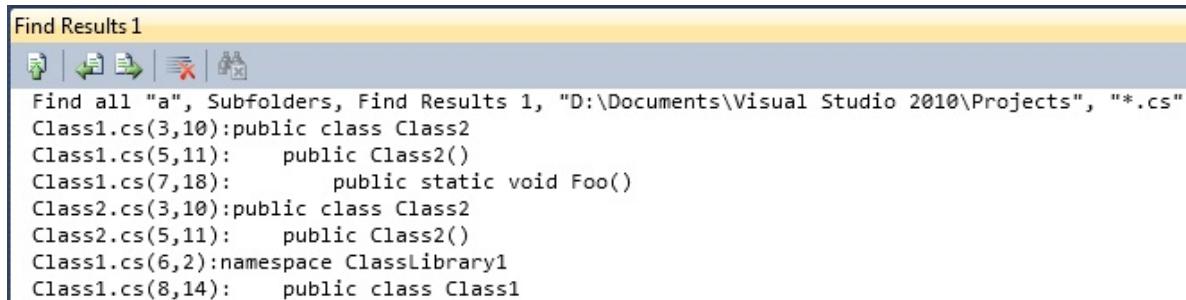
<b>DEFAULT</b>	Ctrl+Shift+F
<b>VISUAL BASIC 6</b>	Ctrl+Shift+F
<b>VISUAL C# 2005</b>	Ctrl+Shift+F
<b>VISUAL C++ 2</b>	Ctrl+Shift+F
<b>VISUAL C++ 6</b>	Ctrl+Shift+F
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt,E, F, I
<b>MENU</b>	Edit   Find and Replace   Find in Files
<b>COMMAND</b>	Edit.FindinFiles
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0002

You can customize your Find In Files results to just about any format you can imagine. For example, let's say you don't want to view the entire file path shown in the Find Results tool window:



```
Find all "a", Subfolders, Find Results 1, "D:\Documents\Visual Studio 2010\Projects", "*.cs"
D:\Documents\Visual Studio 2010\Projects\Class1.cs(3):public class Class2
D:\Documents\Visual Studio 2010\Projects\Class1.cs(5):    public Class2()
D:\Documents\Visual Studio 2010\Projects\Class1.cs(7):        public static void Foo()
D:\Documents\Visual Studio 2010\Projects\Class2.cs(3):public class Class2
D:\Documents\Visual Studio 2010\Projects\Class2.cs(5):    public Class2()
```

Instead, you want this:



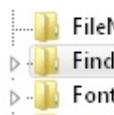
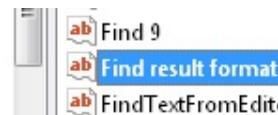
```
Find all "a", Subfolders, Find Results 1, "D:\Documents\Visual Studio 2010\Projects", "*.cs"
Class1.cs(3,10):public class Class2
Class1.cs(5,11):    public Class2()
Class1.cs(7,18):        public static void Foo()
Class2.cs(3,10):public class Class2
Class2.cs(5,11):    public Class2()
Class1.cs(6,2):namespace ClassLibrary1
Class1.cs(8,14):    public class Class1
```

You can easily make this change. Just follow these instructions:

#### WARNING

This involves modifying the registry, so use this tip at your own risk.

1. Open RegEdit.exe.
2. Go to HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\<version>\Find.
3. Add a new string called Find Result Format, with a value of \$f\${l},\${c} : \${t}\r\n.

		REG_SZ	person.vb
▷	ab Find 9	REG_SZ	\$f\${l},\${c} : \${t}\r\n
▷	ab Find result format	REG_SZ	1
▷	ab FindTextFromEditor	REG_SZ	

4. In Visual Studio, run a Find In Files search.

#### NOTE

You do not need to restart Visual Studio to see the changes made in the registry, which is great for testing different string combinations.

## Variables

For your reference, the following are valid values you can use when creating your own custom values.

### Files

- **\$p** path
- **\$f** filename
- **\$v** drive/unc share
- **\$d** directory
- **\$n** name
- **\$e** .extension

### Location

- **\$l** line
- **\$c** col

- **\$x** end col if on first line, else end of first line
- **\$L** span end line
- **\$C** span end col

## Text

- **\$0** matched text
- **\$t** text of first line
- **\$s** summary of hit
- **\$T** text of spanned lines

## Char

- **\n** newline
- **\s** space
- **\t** tab
- **\\\** backslash
- **\\$ \$**

# Chapter 6. Writing Code

*“We will never be rid of code, because code represents the details of the requirements. At some level those details cannot be ignored or abstracted; they have to be specified. And specifying requirements in such detail that a machine can execute them is programming. Such a specification is code.”*

— Robert C. Martin “*Clean Code: A Handbook of Agile Software Craftsmanship*”

Writing code and debugging code are the two activities we tend to do more than any other as developers. It’s no accident that this chapter is one of the two largest in the book. Within these pages, you’ll find tips from older versions all the way through to the great new features in Visual Studio 2010. Really take your time to absorb the material here, and find those pieces that are most relevant to your situation.

As you are reading, make sure to pay particular attention to the new IntelliSense and box selection improvements. In my travels, thousands of people have found these particularly useful for daily work. Also, take some time to review the tips on code snippets and discover how they can accelerate your code writing.

## 06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel

<b>WINDOWS</b>	Ctrl,Mouse Wheel
<b>COMMAND</b>	View.ZoomIn; View.ZoomOut
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEdit0002

The editor has a new feature that allows you to instantly change the zoom factor of text. It's particularly useful for pair programming and doing code demos for your team. Just hold down your Ctrl key and use the wheel on your mouse to zoom in or out.

```
└─ using System;
   └─ using System.Collections;
   └─ using System.Linq;
   └─ using System.Text;

└─ using System;
   └─ using System.Collections;
   └─ using System.Linq;
   └─ using System.Text;
```

If you don't like this feature, you can disable it by installing an extension called "Disable Mouse Wheel Zoom," which you can find at <http://visualstudiogallery.msdn.microsoft.com/en-us/d088791c-150a-4834-8f28-462696a82bb8?SRC=VSIDE>.

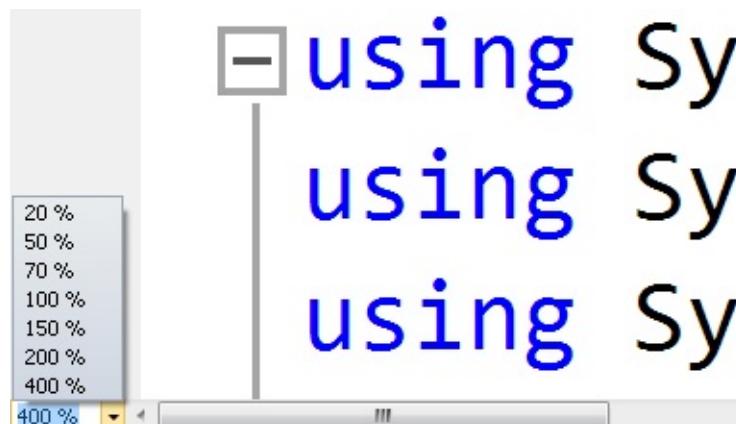
## 06.02 Zoom In or Out of Text in the Editor

<b>DEFAULT</b>	Ctrl+Shift+> (zoom in); Ctrl+Shift+< (zoom out)
<b>VISUAL BASIC 6</b>	Ctrl+Shift+> (zoom in); Ctrl+Shift+< (zoom out)
<b>VISUAL C# 2005</b>	Ctrl+Shift+> (zoom in); Ctrl+Shift+< (zoom out)
<b>VISUAL C++ 2</b>	Ctrl+Shift+> (zoom in); Ctrl+Shift+< (zoom out)
<b>VISUAL C++ 6</b>	Ctrl+Shift+> (zoom in); Ctrl+Shift+< (zoom out)
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+> (zoom in); Ctrl+Shift+< (zoom out)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	View.ZoomIn; View.ZoomOut
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEdit0003

In vstipEdit0002, [06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel](#), on page 209, I showed you how to use the mouse to zoom in and out of your text. Now I'll show you two additional ways to accomplish this goal.

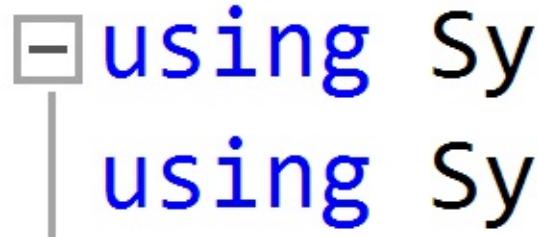
### Combo Box

First, you can change the zoom factor for text by using the Zoom combo box in the lower-left corner of the editor. Simply choose a pre-determined size, or type in a value of your own. To my knowledge there is no keyboard shortcut for getting to this area.



## Keyboard

Second, you can use the keyboard shortcuts Ctrl+Shift+Greater Than (>), to zoom in, and Ctrl+Shift+Less Than (<), to zoom out.



## Universal Zoom

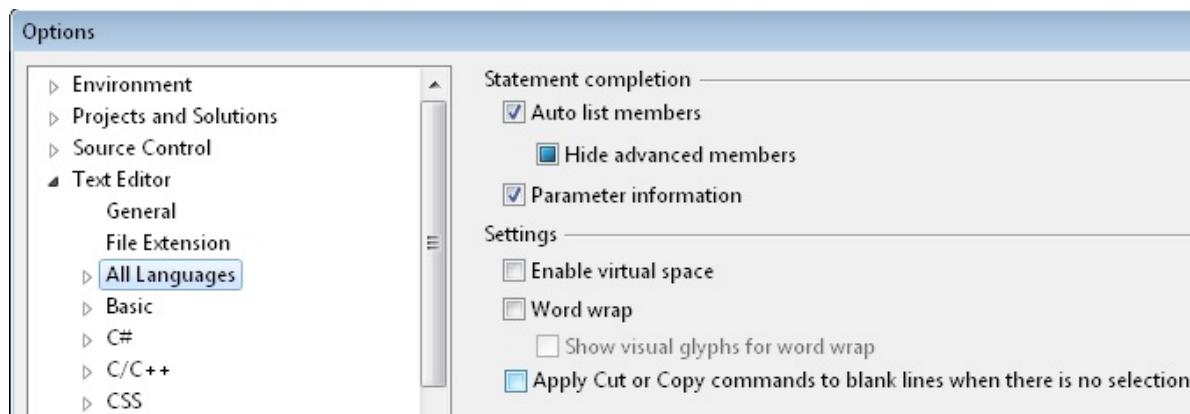
The zoom factor is per tab and doesn't apply across each file, so you need to manually set it on each tab. Alternatively, you can use an extension called "Presentation Zoom," located at <http://visualstudiogallery.msdn.microsoft.com/en-us/6a7a0b57-7059-470d-bcfa-60ceb78dc752?SRC=VSLIDE>.

## 06.03 How to Keep from Accidentally Copying a Blank Line

WINDOWS	Alt,T, O
MENU	Tools   Options   Text Editor   All Languages   General
VERSIONS	2005, 2008, 2010
CODE	vstipEdit0004

Ever cut something and accidentally cut a blank line? I can't really think of a good reason to cut or copy just a blank line, and yet you are still allowed to do it.

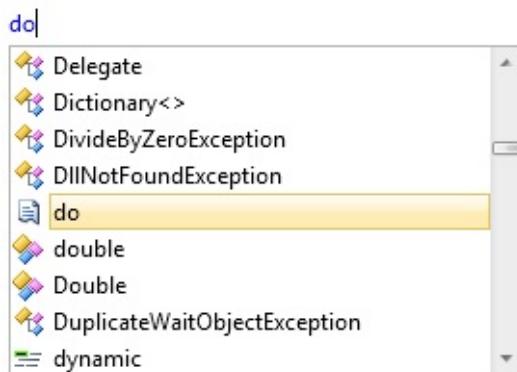
The good news is that you can keep this from ever happening again by simply going to Tools | Options | Text Editor | All Languages | General and clearing the Apply Cut Or Copy Commands To Blank Lines When There Is No Selection check box.



## 06.04 Make IntelliSense Transparent

<b>WINDOWS</b>	Ctrl
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEdit0077

Sometimes when you are writing code, you find yourself in a situation where IntelliSense is covering up something you want to see, as shown in the following illustration.



You can make it temporarily transparent simply by pressing and holding the Ctrl key.



Using this tip, you don't have to get rid of IntelliSense, look at the code, and then bring IntelliSense back. Just press the Ctrl key, get the information you need, and move on.

## 06.05 Cut or Delete the Current Line

<b>DEFAULT</b>	Ctrl+L (cut line); Ctrl+Shift+L (delete line); Shift+Del (cut line); Ctrl+X (cut line)
<b>VISUAL BASIC 6</b>	Ctrl+Y (cut line); [no shortcut] (delete line); Shift+Del (cut line); Ctrl+X (cut line)
<b>VISUAL C# 2005</b>	Ctrl+Y (cut line); Ctrl+Shift+L (delete line); Shift+Del (cut line); Ctrl+X (cut line)
<b>VISUAL C++ 2</b>	Ctrl+Y (cut line); Ctrl+L (cut line); Ctrl+Shift+L (delete line); Shift+Del (cut line); Ctrl+X (cut line); Ctrl+Alt+W (cut line)
<b>VISUAL C++ 6</b>	Ctrl+L (cut line); Shift+Alt+L (cut line); Ctrl+Shift+L (delete line); Shift+Del (cut line); Ctrl+X (cut line)
<b>VISUAL STUDIO 6</b>	Ctrl+M (cut line); Ctrl+L (cut line); Ctrl+Shift+L (delete line); Ctrl+Shift+M (delete line); Shift+Del (cut line); Ctrl+X (cut line)
<b>WINDOWS</b>	Alt,E, T

<b>MENU</b>	Edit   Cut
<b>COMMAND</b>	Edit.LineCut; Edit.LineDelete; Edit.Cut
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0038

You will often find times when you want to cut a line to be pasted somewhere else or just delete a line entirely. Let's take a look at how you can quickly cut or delete any line.

Regardless of what you are trying to do, begin by placing your cursor inside the line anywhere:

```
throw new ArgumentException("userName");
```

## Cut

If you want to cut the line for use somewhere else, perform one of the following actions:

- Press Ctrl+L to cut the line; this uses the Edit.LineCut command.
- Press Shift+Del or Ctrl+X to cut the line using the Edit.Cut command.

There is no difference in the result just two different commands for accomplishing the same task.

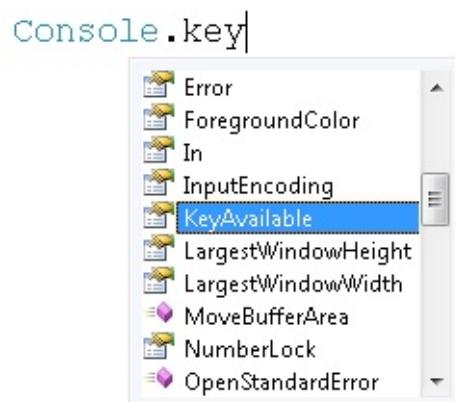
## Delete

If you want to delete the line instead, you can use Ctrl+Shift+L to have it permanently removed from your code.

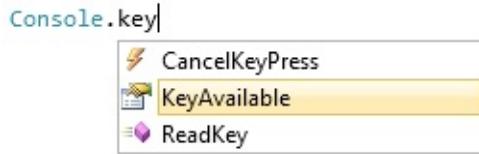
## 06.06 Using the New IntelliSense: Keywords

<b>DEFAULT</b>	Ctrl+J
<b>VISUAL BASIC 6</b>	Ctrl+J
<b>VISUAL C# 2005</b>	Ctrl+J Ctrl+K, L Ctrl+K, Ctrl+L
<b>VISUAL C++ 2</b>	Ctrl+J Ctrl+Alt+T
<b>VISUAL C++ 6</b>	Ctrl+J Ctrl+Alt+T
<b>VISUAL STUDIO 6</b>	Ctrl+J
<b>WINDOWS</b>	Alt,E, I, L
<b>MENU</b>	Edit   IntelliSense   List Members
<b>COMMAND</b>	Edit.ListMembers
<b>VERSIONS</b>	2010
<b>LANGUAGES</b>	VB, C#
<b>CODE</b>	vstipEdit0016

The one feature we use more than just about anything else in Visual Studio is IntelliSense. It has been our friend for many years. Well, it just got friendlier. To show you the new features, let's take a closer look at Visual Studio 2008 IntelliSense. Notice what happens when you type **Console.Key**:



It does what you would expect it to do and highlights the first (in this case, *only*) item that *begins with* the word “Key” in a huge alphabetical list of item names. That’s great, but what if you don’t know what you are looking for but you do know that it has the word “Key” somewhere in it? Well, you can go search in the Object Browser, of course, or better yet, you can use the new IntelliSense in Visual Studio 2010. Look what happens when you do the same thing in Visual Studio 2010:

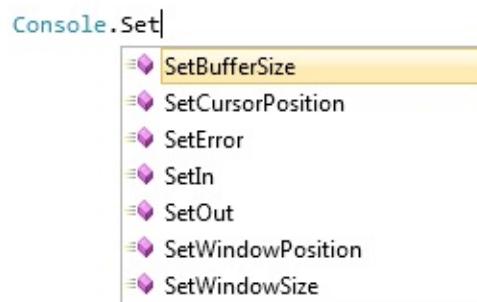


It now shows only those items that have the word “Key” in them and doesn’t care where the word is in the name of the member. This results in a significantly smaller list of items in IntelliSense and an easier way to find names even if you don’t know what they begin with.

## 06.07 Using the New IntelliSense: Pascal Case

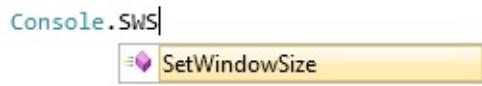
<b>DEFAULT</b>	Ctrl+J
<b>VISUAL BASIC 6</b>	Ctrl+J
<b>VISUAL C# 2005</b>	Ctrl+J Ctrl+K, L Ctrl+K, Ctrl+L
<b>VISUAL C++ 2</b>	Ctrl+J Ctrl+Alt+T
<b>VISUAL C++ 6</b>	Ctrl+J Ctrl+Alt+T
<b>VISUAL STUDIO 6</b>	Ctrl+J
<b>WINDOWS</b>	Alt,E, I, L
<b>MENU</b>	Edit   IntelliSense   List Members
<b>COMMAND</b>	Edit.ListMembers
<b>VERSIONS</b>	2010
<b>LANGUAGES</b>	VB, C#
<b>CODE</b>	vstipEdit0017

Have you ever been in a situation where you wanted to use IntelliSense to get a method but you're faced with a *ton* of methods that start with same word, meaning that you have to type almost the entire method name?

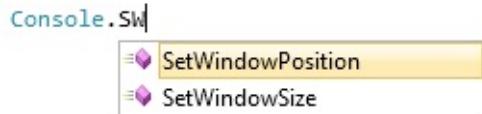


Well, those days are over. Let's say you want the SetWindowSize method, but you really, really don't want to type it out or even scroll down to get the method.

IntelliSense now supports Pascal case. All you have to do is type **SWS**, and you are all set:



What if you don't remember all the uppercase letters in a name? No problem. Just type what you know (they don't even have to be in the correct order), and IntelliSense narrows the list down for you:



## 06.08 Comment and Uncomment in Web Pages

<b>DEFAULT</b>	Ctrl+K, Ctrl+C (comment); Ctrl+K, Ctrl+U (uncomment)
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+C (comment); Ctrl+K, Ctrl+U (uncomment)
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+C (comment); Ctrl+E, Ctrl+C (comment); Ctrl +E, C (comment); Ctrl+K, Ctrl+U (uncomment)
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+C (comment); Ctrl+K, Ctrl+U (uncomment)
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+C (comment); Ctrl+K, Ctrl+U (uncomment)
<b>WINDOWS</b>	Alt,E, V, M (comment) Alt,E, V, E (uncomment)
<b>MENU</b>	Edit   Advanced   Comment Selection; Edit   Advanced   Uncomment Selection
<b>COMMAND</b>	Edit.CommentSelection; Edit.UncommentSelection
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0083

By now, you probably know that you can comment and uncomment your code, but did you know that you can do the same thing with your source during web development? Just put your cursor inside any element you want commented, as shown in the following illustration.

```
<h2>
    Welcome| to ASP.NET!
</h2>
```

## Comment

Now just press Ctrl+K, Ctrl+C to comment out the selection.

```
<%--<h2>
    Welcome to ASP.NET!
</h2>--%>
```

Of course, you can also select multiple elements:

```
<p>
    To learn more about .
</p>
<p>
    You can also find <a
        title="MSDN ASP.
    </p>|
```

And you can comment them out too:

```
<%--<p>
    To learn
</p>
<p>
    You can
        titl
    </p>--%>|
```

## Uncomment

Naturally, you can put your cursor inside any commented area:

```
<%--<p>
    To learn
</p>|
<p>
    You can
        titl
    </p>--%>
```

And you can uncomment that area by pressing Ctrl+K, Ctrl+U:

```
<p>
    To learn
</p>|
<p>
    You can
        titl
    </p>
```

## 06.09 Insert a Blank Line Above or Below the Current Line

<b>DEFAULT</b>	Ctrl+Enter (line above); Ctrl+Shift+Enter (line below)
<b>VISUAL BASIC 6</b>	Ctrl+Enter (line above); Ctrl+Shift+Enter (line below)
<b>VISUAL C# 2005</b>	Ctrl+Enter (line above); Ctrl+Shift+Enter (line below)
<b>VISUAL C++ 2</b>	Ctrl+Enter (line above); Ctrl+Shift+Enter (line below)
<b>VISUAL C++ 6</b>	Ctrl+Enter (line above); Ctrl+Shift+Enter (line below)
<b>VISUAL STUDIO 6</b>	Ctrl+Enter (line above); Ctrl+Shift+Enter (line below)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.LineOpenAbove; Edit.LineOpenBelow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0005

This is a great feature for adding extra white space when you need it. Go to any line in the editor, and press Ctrl+Enter to insert a blank line above or press Ctrl+Shift+Enter to insert a blank line below the current line, as shown in the following illustration.

```
// i love public tips and tricks
public| int x= 10;
public int y = 30;
```

## 06.10 Transpose Lines, Words, and Characters

<b>DEFAULT</b>	Alt+Shift+T (line); Ctrl+Shift+T (word); Ctrl+T (character)
<b>VISUAL BASIC 6</b>	Alt+Shift+T (line); Ctrl+Shift+T (word); [no shortcut] (character)
<b>VISUAL C# 2005</b>	Alt+Shift+T (line); Ctrl+Shift+T (word); Ctrl+T (character)
<b>VISUAL C++ 2</b>	Alt+Shift+T (line); Ctrl+Shift+T (word); Ctrl+T (character)
<b>VISUAL C++ 6</b>	Alt+Shift+T (line); Ctrl+Shift+T (word); Ctrl+T (character)
<b>VISUAL STUDIO 6</b>	Alt+Shift+T (line); Ctrl+Shift+T (word); Ctrl+T (character)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.LineTranspose; Edit.WordTranspose; Edit.CharTranspose
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0042

Ever have a line you want to move down or switch with another line? Just find a couple of lines you want to switch, and put your cursor in the top line:

```
'seriously cool comment #2  
'seriously cool comment #1
```

Then press Alt+Shift+T to transpose the two lines:

```
'seriously cool comment #1  
'seriously cool comment #2
```

You can also do something similar with words. Place the cursor in the word you want to transpose:

```
'cool seriously comment
```

Press Ctrl+Shift+T:

```
'seriously cool| comment
```

Just want to transpose a character? Just place the cursor to the right of the character you want to transpose:

```
'seriu|osly
```

And then press Ctrl+T to transpose that character with the character at its immediate right:

'seriously

## 06.11 How to Cycle Through the Clipboard Ring

<b>DEFAULT</b>	Ctrl+Shift+V; Ctrl+Shift+Insert
<b>VISUAL BASIC 6</b>	Ctrl+Shift+V; Ctrl+Shift+Insert
<b>VISUAL C# 2005</b>	Ctrl+Shift+V; Ctrl+Shift+Insert
<b>VISUAL C++ 2</b>	Ctrl+Shift+V; Ctrl+Shift+Insert
<b>VISUAL C++ 6</b>	Ctrl+Shift+Insert
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+V; Ctrl+Shift+Insert
<b>WINDOWS</b>	Alt+E, Y
<b>MENU</b>	Edit   Cycle Clipboard Ring
<b>COMMAND</b>	Edit.CycleClipboardRing
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0001

The Clipboard Ring keeps track of the past 20 items you've either cut or copied so that you can reuse them over and over again. After you hit item 20, it goes back to the first item. This is why the feature is called the Clipboard Ring.

This is particularly useful when you move between several code windows and need to copy and paste different items. You can copy all of the code from the original window and then go to one of the other windows and, using the Clipboard Ring, paste all of your items.



Try it out. Copy a few pieces of text into your clipboard, and then keep pressing

Ctrl+Shift+V to repeat-paste them into the editor. This is one seriously cool time-saving feature. Unfortunately, there is no way to see the contents of the Clipboard Ring in Visual Studio without cycling through the list.

## 06.12 Using the Undo and Redo Stack

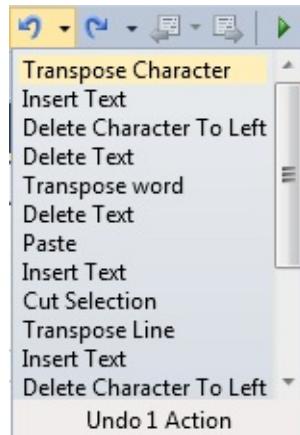
<b>DEFAULT</b>	Ctrl+Z (undo); Alt+Backspace (undo); Ctrl+Shift+Z (redo); Alt+Shift+Backspace (redo); Ctrl+Y (redo)
<b>VISUAL BASIC 6</b>	Ctrl+Shift+Z (redo); Alt+Shift+Backspace (redo)
<b>VISUAL C# 2005</b>	Ctrl+Z (undo); Alt+Backspace (undo); Ctrl+Shift+Z (redo); Alt+Shift+Backspace (redo); Ctrl+Y (redo)
<b>VISUAL C++ 2</b>	Ctrl+Shift+Z (redo); Alt+Shift+Backspace (redo); Ctrl+Y (redo); Ctrl+A
<b>VISUAL C++ 6</b>	Ctrl+Z (undo); Alt+Backspace (undo); Ctrl+Shift+Z (redo); Alt+Shift+Backspace (redo); Ctrl+Y (redo)
<b>VISUAL STUDIO 6</b>	Ctrl+Z (undo); Alt+Backspace (undo); Ctrl+Shift+Z (redo); Alt+Shift+Backspace (redo); Ctrl+Y (redo)
<b>WINDOWS</b>	Alt,E, U (undo); Alt,E, R (redo)
<b>MENU</b>	Edit   Undo; Edit   Redo
<b>COMMAND</b>	Edit.Undo; Edit.Redo
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0045

We have all used Ctrl+Z (undo) at one time or another to fix a mistake. Did you know you don't have to press Ctrl+Z a billion times to go back to a particular action? The Undo and Redo stacks are readily available for you to use for quick, multiple undo or redo operations.

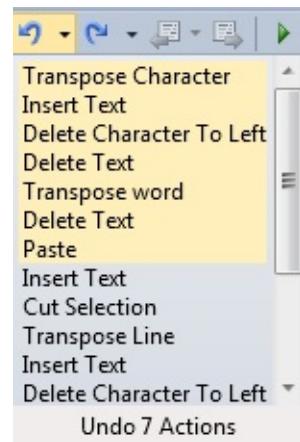
It's easy to use. Just locate the Undo and Redo section of the toolbar shown in the following illustration.



Next, click the drop-down arrow for the action you want to perform. In this case, let's look at the Undo stack:



By default, it undoes only the last action. However, notice what happens when I put my mouse over the list:



It automatically selects multiple actions to undo or redo. Also notice that it shows you at the bottom how many actions you are about to undo. When you have all the actions selected that you want to undo or redo, just click the left mouse button.

## 06.13 Undo and Redo Global Actions

<b>WINDOWS</b>	Alt,E, N (undo); Alt,E, U (redo)
<b>MENU</b>	Edit   Undo Last Global Action; Edit   Redo Last Global Action
<b>COMMAND</b>	Edit.UndoLastGlobalAction; Edit.RedoLastGlobalAction
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0020

Everyone knows about undo and redo, but did you know that the Edit menu offers global versions of these actions?



By definition, global actions impact multiple files. Global actions include renaming a class or namespace, performing a find-and-replace operation across a solution, refactoring a database, or any other action that changes multiple files.

You can apply the global undo and redo commands to actions in the current Visual Studio session, even after you close the solution that an action applies to.

## 06.14 How to Use Reference Highlighting

<b>DEFAULT</b>	Ctrl+Shift+Down Arrow (next); Ctrl+Shift+Up Arrow (previous)
<b>VISUAL BASIC 6</b>	Ctrl+Shift+Down Arrow (next); Ctrl+Shift+Up Arrow (previous)
<b>VISUAL C# 2005</b>	Ctrl+Shift+Down Arrow (next); Ctrl+Shift+Up Arrow (previous)
<b>VISUAL C++ 2</b>	Ctrl+Shift+Down Arrow (next); Ctrl+Shift+Up Arrow (previous)
<b>VISUAL C++ 6</b>	Ctrl+Shift+Down Arrow (next); Ctrl+Shift+Up Arrow (previous)
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+Down Arrow (next); Ctrl+Shift+Up Arrow (previous)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.NextHighlightedReference; Edit.PreviousHighlightedReference
<b>VERSIONS</b>	2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipEdit0010

This is absolutely one of my favorite tips because you don't have to do anything to make it work. The MSDN documentation (at <http://msdn.microsoft.com/en-us/library/ee349251.aspx>) describes reference highlighting this way:

“When you click a symbol in the Code Editor, all instances of the symbol are highlighted in the document. [...] Highlighted symbols may include declarations and references, and generally anything else that Find All References would return. This includes the names of classes, objects, variables, methods, and properties.”

```

private void SomeCoolMethod()
{
    // blah
}

private void A()
{
    SomeCoolMethod();
}

private void B()
{
    SomeCoolMethod();
}

private void C()
{
    SomeCoolMethod();
}

```

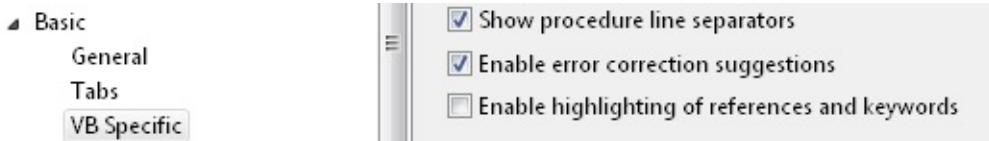
## Navigation

All you have to do is click in any symbol, and it automatically highlights any references in the current document. You can navigate through the highlights by using Ctrl+Shift+Down Arrow (next) or Ctrl+Shift+Up Arrow (previous).

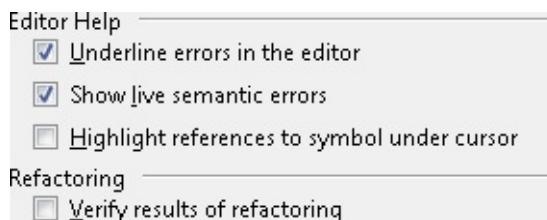
## Turning it Off

What if you don't like this feature? You can always turn it off by using following instructions, depending on whether you're working in VB or C#.

VB: Go to Tools | Options | Text Editor | Basic | VB Specific, and clear the Enable Highlighting Of References And Keywords check box.



C#: Go to Tools | Options | Text Editor | C#| Advanced, and clear the Highlight References To Symbol Under Cursor check box.



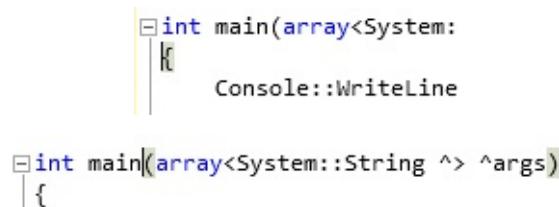
## 06.15 Moving or Selecting Between Matching Braces (C++, C# Only)

<b>DEFAULT</b>	Ctrl+] (move); Ctrl+Shift+] (select)
<b>VISUAL BASIC 6</b>	Ctrl+] (move); Ctrl+Shift+] (select)
<b>VISUAL C# 2005</b>	Ctrl+] (move); Ctrl+Shift+] (select)
<b>VISUAL C++ 2</b>	Ctrl+] (move); Ctrl+M (move); Ctrl+Shift+] (select); Ctrl+Shift+M (select)
<b>VISUAL C++ 6</b>	Ctrl+] (move); Ctrl+Shift+] (select)
<b>VISUAL STUDIO 6</b>	Ctrl+] (move); Ctrl+Shift+] (select)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.GotoBrace; Edit.GotoBraceExtend
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C++, C#
<b>CODE</b>	vstipEdit0075

When you are working with the C\* languages, you sometimes want to move to the bottom or top of a method or code block. You can quickly travel or select between matching braces by performing the following steps.

### Moving

Click next to an opening or closing brace:



```
int main(array<System::String ^> ^args)
{
```

Now press Ctrl+] to go from one brace to the other one. You should find yourself easily navigating between them. This technique is very useful when you have braces far apart from each other. Also notice that the term “brace” is very loose here and applies to curly braces, square brackets, and parentheses.

## Selecting

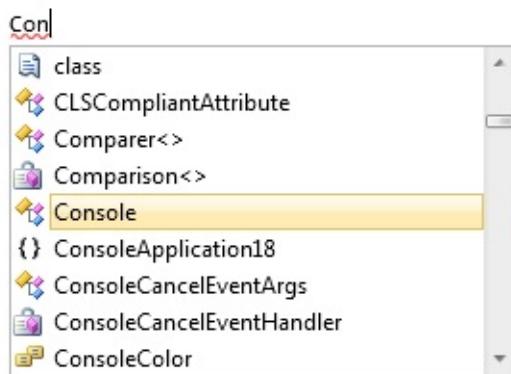
In addition to moving, you can also select everything between matching braces by pressing Ctrl+Shift+] , as shown in the following illustration.

```
int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hello World");
    DoSomething();
    return 0;
}
```

## 06.16 Invoke Statement Completion

<b>DEFAULT</b>	Ctrl+J
<b>VISUAL BASIC 6</b>	Ctrl+J
<b>VISUAL C# 2005</b>	Ctrl+J; Ctrl+K, Ctrl+L; Ctrl+K, L
<b>VISUAL C++ 2</b>	Ctrl+J; Ctrl+Alt+T
<b>VISUAL C++ 6</b>	Ctrl+J; Ctrl+Alt+T
<b>VISUAL STUDIO 6</b>	Ctrl+J
<b>WINDOWS</b>	Alt,E, I, L
<b>MENU</b>	Edit   IntelliSense   List Members
<b>COMMAND</b>	Edit.ListMembers
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C++ (not available in VS2010), C#, VB
<b>CODE</b>	vstipEdit0061

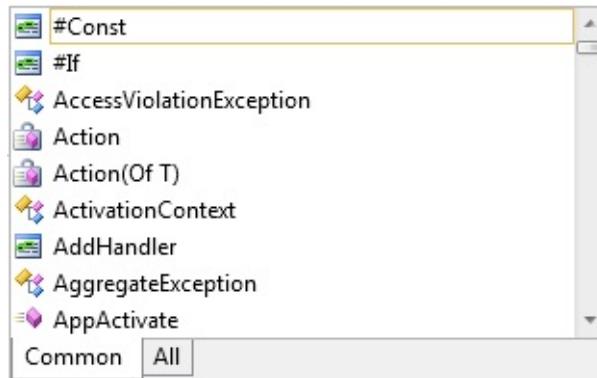
This little trick is particularly useful if you lose IntelliSense and want to get it back. You can invoke statement completion from the command line by using Ctrl+J.



## 06.17 Move Between the Common Tab and All Tab in Statement Completion (VB)

<b>DEFAULT</b>	Alt+. (All Tab); Alt+, (Common Tab)
<b>COMMAND</b>	Edit.IncreaseFilterLevel; Edit.DecreaseFilterLevel
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	VB
<b>CODE</b>	vstipEdit0064

You don't have to take your hands off the keyboard when switching between Common and All in statement completion for Visual Basic. Just press Alt+Period (.) to go to the All tab, and press Alt+Comma (,) to go to the Common tab.



## 06.18 Using Parameter Information

<b>DEFAULT</b>	Ctrl+Shift+Space (show Parameter Info); Up or Down Arrow (move to next or previous overload)
<b>VISUAL BASIC 6</b>	Ctrl+Shift+Space (show Parameter Info); Ctrl+Shift+I (show Parameter Info); Up or Down Arrow (move to next or previous overload)
<b>VISUAL C# 2005</b>	Ctrl+Shift+Space (show Parameter Info); Ctrl+K, Ctrl+P (show Parameter Info); Ctrl+K, P (show Parameter Info); Up or Down Arrow (move to next or previous overload)
<b>VISUAL C++ 2</b>	Ctrl+Shift+Space (show Parameter Info); Up or Down Arrow (move to next or previous overload)
<b>VISUAL C++ 6</b>	Ctrl+Shift+Space (show Parameter Info); Up or Down Arrow (move to next or previous overload)
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+Space (show Parameter Info); Ctrl+Shift+I (show Parameter Info); Up or Down Arrow (move to next or previous overload)
<b>WINDOWS</b>	Alt,E, I, P
<b>MENU</b>	Edit   IntelliSense   Parameter Info
<b>COMMAND</b>	Edit.ParameterInfo
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C++ (not available in VS2010); C#, VB
<b>CODE</b>	vstipEdit0062

When working with a function, you automatically get parameter information, as shown in the following illustration.



If it goes away and you want it back, you can always press Ctrl+Shift+Space. Also notice that the function in this example has 19 overloads. You can iterate through them by pressing your Up or Down Arrow keys. If you are using your mouse, you can go forward by clicking anywhere in the parameter information area. You do not have to click the tiny up and down arrows inside the box unless you want to go backward in the list.

## 06.19 Using Quick Info

<b>DEFAULT</b>	Ctrl+K, Ctrl+I
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+I; Ctrl+I
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+I; Ctrl+K, I
<b>VISUAL C++ 2</b>	Ctrl+T
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+I
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+I; Ctrl+I
<b>WINDOWS</b>	ALT,E, I, Q
<b>MENU</b>	Edit   IntelliSense   Quick Info
<b>COMMAND</b>	Edit.QuickInfo
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C++ (not available in VS2010), C#, VB
<b>CODE</b>	vstipEdit0063

The Quick Info option helps you identify the details of a particular function. It comes up automatically as you type, but if it goes away, you can quickly bring it back by placing the cursor in a method and pressing Ctrl+K, Ctrl+I.

```
static void Main(string[] args)
{
    Console.WriteLine
}

void Console.WriteLine(string format, params object[] arg) (+ 18 overload(s))
Writes the text representation of the specified array of objects, followed by the

Exceptions:
System.IO.IOException
System.ArgumentNullException
System.FormatException
```

## 06.20 Word Completion

<b>DEFAULT</b>	Ctrl+Space; Alt+Right Arrow
<b>VISUAL BASIC 6</b>	Ctrl+Space; Alt+Right Arrow
<b>VISUAL C# 2005</b>	Ctrl+Space; Alt+Right Arrow Ctrl+K, Ctrl+W; Ctrl+K, W
<b>VISUAL C++ 2</b>	Ctrl+Space; Alt+Right Arrow
<b>VISUAL C++ 6</b>	Ctrl+Space; Alt+Right Arrow
<b>VISUAL STUDIO 6</b>	Ctrl+Space; Alt+Right Arrow
<b>WINDOWS</b>	Alt,E, I, W
<b>MENU</b>	Edit   IntelliSense   Complete Word
<b>COMMAND</b>	Edit.CompleteWord
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C++ (not available in VS2010), C#, VB
<b>CODE</b>	vstipEdit0065

This one is very popular and one of the core skills to use in Visual Studio. Let's say you are typing and you have the situation shown in the following illustration.

```
Console.WriteLine|
```

You can press Ctrl+Space or Alt+Right Arrow to complete the word:

```
Console.WriteLine|
```

This works only if there are no other possible matches. Suppose, for example, that you have something similar to the following:

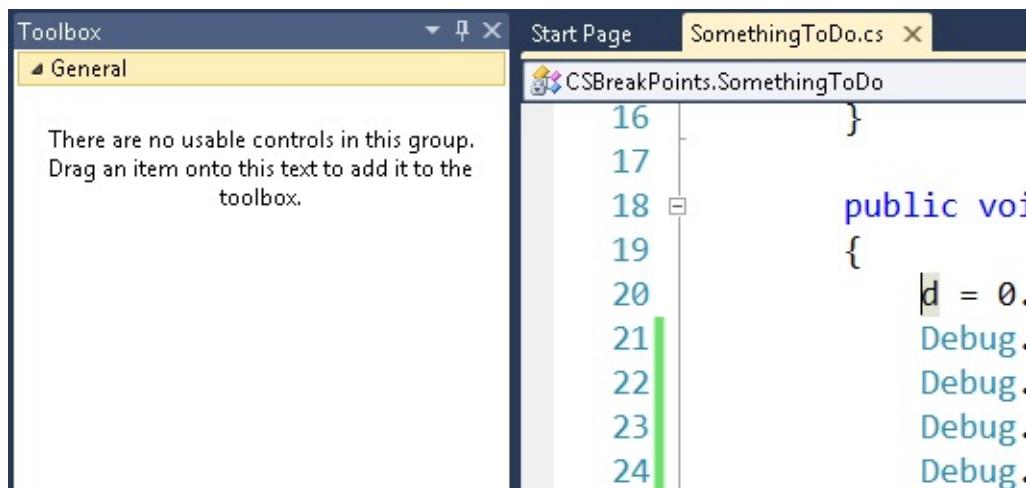
```
Console.Wr|
```

If you now press Ctrl+Space or Alt+Right Arrow to complete the word, you would just get statement completion instead.

## 06.21 Drag and Drop Code into the Toolbox

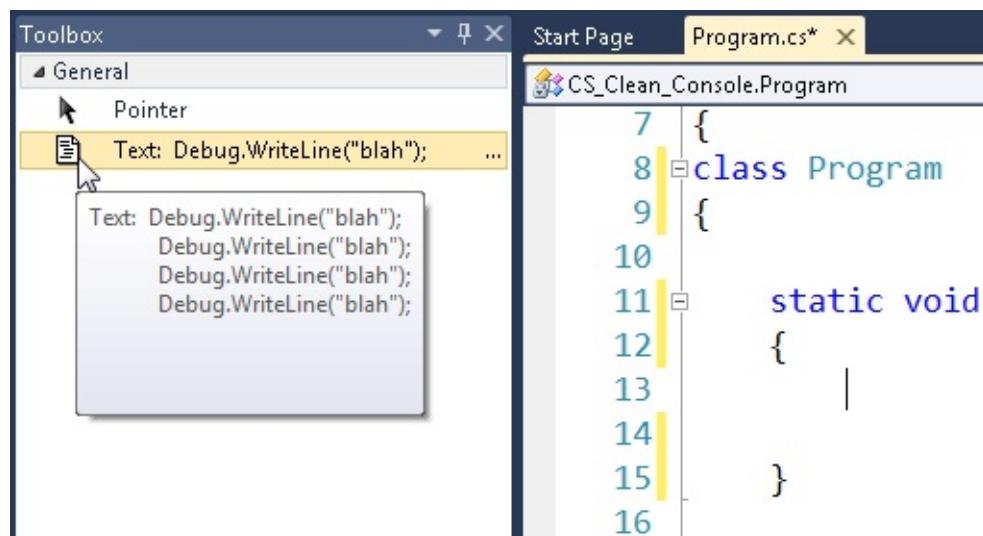
VERSIONS	2005, 2008, 2010
CODE	vstipTool0007

Got code you use all the time? Start using the Toolbox for more than just controls. When you are in the Editor, the Toolbox appears as shown in the following illustration.



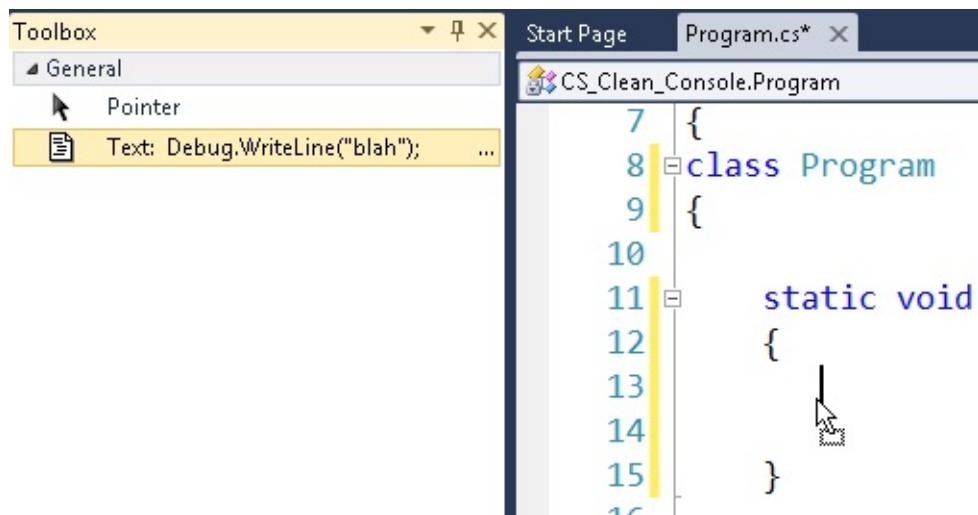
This is a vast expanse of opportunity! Just select some text from the editor, and drag it into the Toolbox and see the magic happen.

Now you have code ready to go anytime. Just place the cursor where you want the code to go, and double-click the item in your Toolbox.

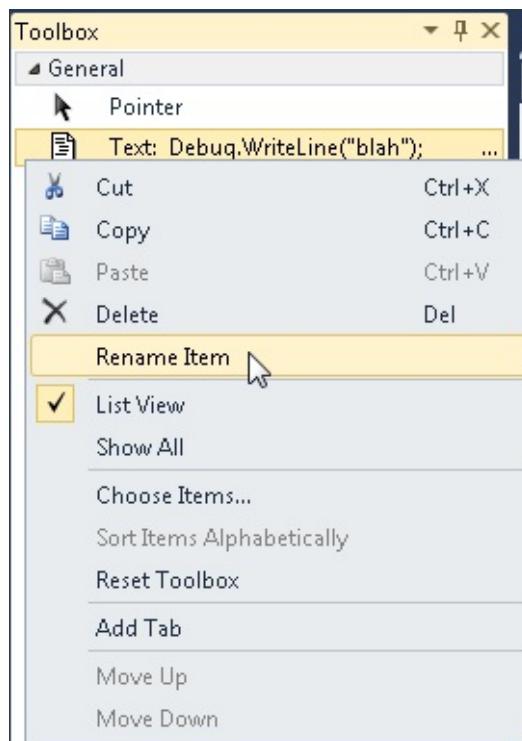


You can also click and drag where you want the code to go, as shown in the

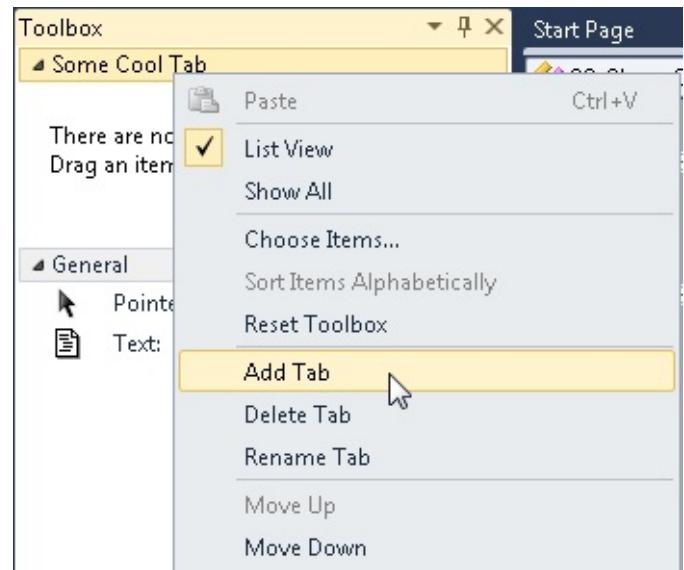
following illustration.



You also have a couple of extra things you can do. For example, you can rename (among other tasks) the item in the Toolbox to whatever you would like by right-clicking on it:



You can also create additional tabs, delete existing tabs, and rename tabs to organize the Toolbox better.



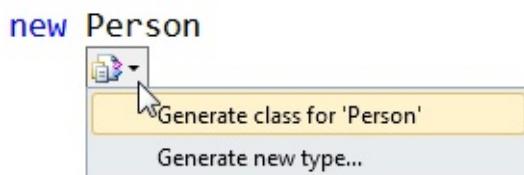
## 06.22 Using Smart Tags from the Keyboard

<b>DEFAULT</b>	Ctrl+. (period); Shift+Alt+F10
<b>VISUAL BASIC 6</b>	Ctrl+. (period); Shift+Alt+F10
<b>VISUAL C# 2005</b>	Ctrl+. (period); Shift+Alt+F10
<b>VISUAL C++ 2</b>	Ctrl+. (period); Shift+Alt+F10
<b>VISUAL C++ 6</b>	Ctrl+. (period); Shift+Alt+F10
<b>VISUAL STUDIO 6</b>	Ctrl+. (period); Shift+Alt+F10
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	View.ShowSmartTag
<b>VERSIONS</b>	2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipEdit0076

You have probably seen the smart tag indicator before. It's a blue or red bar that shows up in your code when you use certain items.

`new Person`

Most people just put their mouse over the line and choose the option(s) listed in the tag.

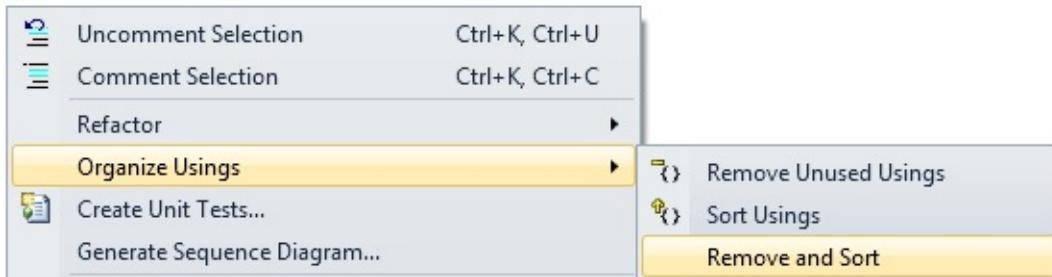


However, did you know that you can use a keyboard shortcut to bring up the options? Just press Ctrl+Period (.), and it brings up the option(s) without using the mouse.

## 06.23 Organize Using Statements (C# Only)

WINDOWS	Alt,E, I, [R (remove), U (sort), A (remove and sort)]
MENU	Edit   IntelliSense   Organize Usings   [Remove Unused Usings, Sort Usings, Remove and Sort]; Right Click   Organize Usings   [Remove Unused Usings, Sort Usings, Remove and Sort]
COMMAND	EditorContextMenus.CodeWindow.OrganizeUsings.[RemoveUnusedUsings, SortUsings, RemoveAndSort]
VERSIONS	2008, 2010
LANGUAGES	C#
CODE	vstipEdit0070

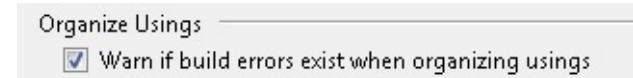
You can easily organize your using statements. Simply right-click anywhere in the editor to get the context menu, choose Organize Usings, and then Remove, Sort, or Remove And Sort.



## Remove Unused Usings

Before removing unused using directives, using aliases, and extern aliases, you should consider a couple things about this feature.

First, it should be used only on code that builds, because it could remove required using statements if activated on code that does not build. As shown in the following illustration, you have an option in Visual Studio 2008 (*not* Visual Studio 2010), found at Tools | Options | Text Editor | C# | Advanced | Organize Usings, that prevents you from removing using statements if your code doesn't build.



Second, it works only on the active set of code. For example, suppose you have a

set of using statements such as the following:

```
❑using System;
| using System.Collections.Generic;
| using System.Linq;
| using System.Text;
| using Microsoft.CSharp.RuntimeBinder;
| using System.Text.RegularExpressions;
| using System.Collections.Concurrent;
```

If you remove the unused using statements, you see the following result:

```
❑using System;
```

However, suppose those using statements are organized into code that has active and inactive blocks:

```
#define DEBUG

#if DEBUG
❑using System;
| using System.Collections.Generic;
| using System.Linq;
| using System.Text;

#else
❑using Microsoft.CSharp.RuntimeBinder;
| using System.Text.RegularExpressions;
| using System.Collections.Concurrent;
#endif
```

When you perform the action to remove them, only the extra using statements in the active block are removed:

```
#define DEBUG

#if DEBUG
using System;

#else
❑using Microsoft.CSharp.RuntimeBinder;
| using System.Text.RegularExpressions;
| using System.Collections.Concurrent;
#endif
```

Generally accepted reasons for removing unused using statements include the following:

- Results in cleaner code
- Significantly reduces the size of the IntelliSense list because there are fewer namespaces

- Enables potentially faster compilation because the compiler has fewer namespaces to resolve
- Avoids potential naming collisions when new types are added to the unused namespaces that might have the same name as types you are currently using

## Sort Usings

When you perform a sort action on your using statements, they sort in the following order: extern aliases, using directives, using aliases. Also, by default, using statements that reference the System namespace are sorted before the other using directives. So, suppose you have some using statements such as the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.CSharp.RuntimeBinder;
using System.Text.RegularExpressions;
using System.Collections.Concurrent;
```

When you sort them, you get the result shown in the following illustration.

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using Microsoft.CSharp.RuntimeBinder;
```

Notice that the Microsoft namespace is below all the System namespaces. If you don't like this behavior, you can change it by going to Tools | Options | Text Editor | C# | Advanced | Organize Usings and clearing the Place 'System' Directives First When Sorting Usings check box.

Now, when you sort, you see the following result:

```
using Microsoft.CSharp.RuntimeBinder;
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
```

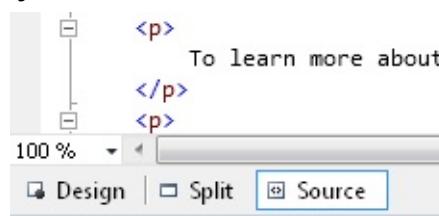
## Remove and Sort

When you click the Remove And Sort command, it first performs a remove operation and then it performs a sort operation.

## 06.24 Switch Between Design and Source in Web Projects

<b>DEFAULT</b>	Shift+F7 (view designer and view markup toggle)
<b>VISUAL BASIC 6</b>	Shift+F7 (view designer and view markup toggle)
<b>VISUAL C# 2005</b>	Shift+F7 (view designer and view markup toggle)
<b>VISUAL C++ 2</b>	Shift+F7 (view designer and view markup toggle)
<b>VISUAL C++ 6</b>	Shift+F7 (view designer and view markup toggle)
<b>VISUAL STUDIO 6</b>	Shift+F7 (view designer and view markup toggle)
<b>WINDOWS</b>	Alt,V, D (view designer); Alt, V, K (view markup)
<b>MENU</b>	View   Designer; View   Markup
<b>COMMAND</b>	View.ViewDesigner; View.ViewMarkup
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0045

When working with web projects, you can switch between Design and Source (Markup) views quite easily.



In any view, simply press Shift+F7 to switch to the other view. In other words, you can press Shift+F7 as a toggle between views.

## Split View

This keyboard shortcut does not work the way you might think in Split view.

Pressing Shift+F7 takes you to either the Design or Source view (depending on which one you were in last) and does not return to Split view when you press Shift+F7 again.

## 06.25 Toggle Designer

<b>DEFAULT</b>	F7 (view designer); F7 (view code)
<b>VISUAL BASIC 6</b>	[no shortcut] (view designer); F7 (view code)
<b>VISUAL C# 2005</b>	[no shortcut] (view designer); F7 (view code)
<b>VISUAL C++ 2</b>	[no shortcut] (view designer); F7 (view code)
<b>VISUAL C++ 6</b>	[no shortcut] (view designer); F7 (view code)
<b>VISUAL STUDIO 6</b>	[no shortcut] (view designer); F7 (view code)
<b>WINDOWS</b>	Alt,V, D (view designer); Alt,V, C (view code)
<b>MENU</b>	View   Designer; View   Code
<b>COMMAND</b>	View.ToggleDesigner; View.ViewCode
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0044

You often find that there is a need to switch from Design view (web, windows, or WPF) to code. You can use F7 to go from the current Source or Design view to the Code view.

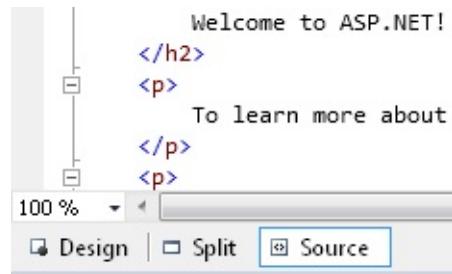
To go back, just press F7 again. It's a toggle, so it jumps back and forth between the views.

In WPF applications, you can use F7 to view code but not to go back to Design view, so it isn't a full toggle operation in that scenario.

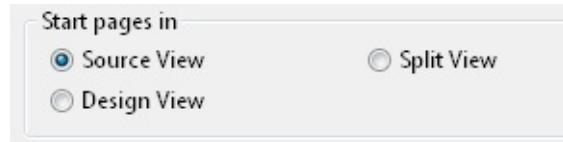
## 06.26 Change the Default View in the HTML Editor

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   HTML Designer   General
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0088

The default view when you use the HTML Designer is Source view.



If you don't like this or just want another view to be the default, you can go to Tools | Options | HTML Designer | General and choose a new view in the Start Pages In area.



### NOTE

In Visual Studio 2005, you don't have the Split View option.

## 06.27 Jump Back to the Editor from Just About Anywhere

<b>DEFAULT</b>	Esc
<b>VISUAL BASIC 6</b>	Esc
<b>VISUAL C# 2005</b>	Esc
<b>VISUAL C++ 2</b>	Esc
<b>VISUAL C++ 6</b>	Esc; Alt+0
<b>VISUAL STUDIO 6</b>	Esc
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Window.ActivateDocumentWindow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0066

Did you know that you can usually get to the active document window by just pressing Esc? For example, if you are in any tool window (like the Output window) and you press Esc, you go back to the active document window.

You might have to press Esc multiple times, depending on the situation, but it should almost always wind up at the active document window.

## 06.28 Replacing Text with a Box Selection

<b>DEFAULT</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL BASIC 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C# 2005</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C++ 2</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C++ 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL STUDIO 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.Line[Up, Down]ExtendColumn; Edit.Char[Left, Right]ExtendColumn
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEdit0006

You have been able to select a block of text for some time in Visual Studio by holding down the Alt+Shift+Arrow keys (or Alt+Left Mouse Button) and making a selection. In Visual Studio 2010, you can now do multiline replacements of a box selection.

Simply select a block of text:

```
public int x = 10;
public int y = 30;
public| int z = 50;
```

Now type your replacement text:

```
private| int x = 10;
private| int y = 30;
private| int z = 50;
```

The Editor turns your box selection into a zero-length box selection (a multiline cursor) that you can use to type in what you like.

## 06.29 Pasting the Contents of One Box Selection into Another

<b>DEFAULT</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL BASIC 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C# 2005</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C++ 2</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C++ 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL STUDIO 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.Line[Up, Down]ExtendColumn; Edit.Char[Left, Right]ExtendColumn
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0007

Box selection is a powerful tool that is often overlooked. One of the things you can do, for example, is paste one box selection into another of the same size. Let's assume you have the following code:

```
public int x = 10;
public int y = 30;
public int z = 50;

public double a = 30;
public Int32 b = 50;
public Int64 c = 60;
```

However, you want the first set of variables to be the same data types as the second set (and in the same order). Simply box select (Alt+Left Mouse Button) the second set of data types, and then copy the selection (Ctrl+C).

```
public int x = 10;
public int y = 30;
public int z = 50;

public double a = 30;
public Int32 b = 50;
public Int64 c = 60;
```

Next, select the first set of data types (Alt+Left Mouse Button):

```
public int x = 10;  
public int y = 30;  
public int| z = 50;  
  
public double a = 30;  
public Int32 b = 50;  
public Int64 c = 60;
```

And paste (Ctrl+V):

```
public double x = 10;  
public Int32 y = 30;  
public Int64 z = 50;  
  
public double a = 30;  
public Int32 b = 50;  
public Int64 c = 60;
```

## 06.30 Pasting a Single Selection into a Box Selection

<b>DEFAULT</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL BASIC 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C# 2005</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C++ 2</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C++ 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL STUDIO 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.Line[Up, Down]ExtendColumn; Edit.Char[Left, Right]ExtendColumn
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEdit0008

In this tip, we see how to take a single selection and put it into a box selection. Let's assume you have the following code:

```
public int x = 10;
public int y = 30;
public int z = 50;

public double a = 30;
```

But you realize that you need change all the ints to doubles.

Select the double keyword and copy it (Ctrl+C):

```
public int x = 10;
public int y = 30;
public int z = 50;

public double a = 30;
```

Then box select the destination (Alt+left mouse button):

```
public int x = 10;  
public int y = 30;  
public int z = 50;  
  
public double a = 30;
```

Finally, do a paste (Ctrl+V) to see the following result:

```
public double x = 10;  
public double y = 30;  
public double| z = 50;  
  
public double a = 30;
```

## 06.31 Using Zero-Length Box Selection

<b>DEFAULT</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL BASIC 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C# 2005</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C++ 2</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL C++ 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>VISUAL STUDIO 6</b>	Shift+Alt+[Up, Down, Left, Right] Arrow
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.Line[Up, Down]ExtendColumn; Edit.Char[Left, Right]ExtendColumn
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEdit0009

The power of box selection is even more powerful with the new zero-length box selection. Let's assume that you have a situation with some variables, like the ones shown in the following illustration, and you want to make them all public.

```
| double a = 10;  
| double b = 20;  
| double c = 30;  
| double d = 40;  
| double e = 50;
```

The answer is a zero-length box selection. Hold down your Alt key, and use your Down Arrow key to extend straight down. A line is created, as shown in the following illustration.

```
double a = 10;  
double b = 20;  
double c = 30;  
double d = 40;  
double e = 50;
```

Release the keys, and now just start typing.

```
public double a = 10;  
public double b = 20;  
public double c = 30;  
public double d = 40;  
public double e = 50;
```

This feature acts just like any cursor, so you can go forward and backward at will, plus you can put one of these anywhere you want to create or edit multiple lines of text.

## 06.32 View White Space

<b>DEFAULT</b>	Ctrl+R, Ctrl+W
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	Ctrl+R, Ctrl+W; Ctrl+E, Ctrl+S; Ctrl+E, S
<b>VISUAL C++ 2</b>	Ctrl+R, Ctrl+W; Ctrl+Alt+T
<b>VISUAL C++ 6</b>	Ctrl+R, Ctrl+W; Ctrl+Shift+8
<b>VISUAL STUDIO 6</b>	Ctrl+R, Ctrl+W
<b>WINDOWS</b>	Alt+E, V, W
<b>MENU</b>	Edit   Advanced   View White Space
<b>COMMAND</b>	Edit.ViewWhiteSpace
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0043

Ever want to see the white space you are working with? Maybe you want to know whether you have tabs or extra spaces on lines? It's easy to find out. Just go to Edit | Advanced | View White Space (Ctrl+R, Ctrl+W).

The diagram illustrates the representation of white space in code. It shows two code snippets with annotations:

- The first snippet shows two green comments: // some comment and // another comment. A vertical line with an arrow points from the start of the first comment to the start of the second.
- The second snippet shows a line of code with a series of dots (.....) followed by // some comment and then another series of dots (.....). Arrows point from the start of the first set of dots to the start of the second set, and from the start of the second set to the start of the comment.
- The third snippet shows a line of code with arrows pointing from the start of each word to the start of the next word, indicating tab stops.

Spaces are represented as dots, and tabs are the arrows you see in the preceding illustration.

## 06.33 Collapsing Your Code with Outlining

<b>DEFAULT</b>	Ctrl+M, Ctrl+M
<b>VISUAL BASIC 6</b>	Ctrl+M, Ctrl+M
<b>VISUAL C# 2005</b>	Ctrl+M, Ctrl+M; Ctrl+M, M
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	[no shortcut]
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt, E, O, T
<b>MENU</b>	Edit   Outlining   Toggle Outlining Expansion
<b>COMMAND</b>	Edit.ToggleOutliningExpansion
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0029

By default, outlining is enabled in Visual Studio. It's the line you see with the boxes to indicate the status of the area (collapsed or expanded):

```
namespace CSBreakPoints
{
    class SomethingToDo
    {
        public void Do(Int32 iParam, Int32 iAnotherParam)
        {
            iAnotherParam = 6;
            int jLocal = iParam;
            Re(3.145 * jLocal);
        }
    }
}
```

You can collapse code to get it out of your way so that you can focus on other areas. You have four ways to do it.

### Minus Sign

Click the minus sign to collapse an area of code.

A screenshot of the Visual Studio code editor. A specific region of code is highlighted with a light gray background and a darker gray border, indicating it is collapsed. The code within this region is:

```
namespace CSBreakPoints
{
    class SomethingToDo
    {
        public void Do(Int32 iParam, Int32 iAnotherParam)
        {
            iAnotherParam = 6;
            int jLocal = iParam;
            Re(3.145 * jLocal);
        }
    }
}
```

#### NOTE

Visual Studio 2010, if Visual Experience is turned on in Tools | Options | Environment | General, now highlights the area that will be collapsed. If you don't like the highlighting color, you can go to Tools | Options | Environment | Fonts and Colors and change the color for the Collapsible Region setting.

## Vertical Line

In Visual Studio 2010 only, click *anywhere* on the vertical line in the highlighted region.

A screenshot of the Visual Studio code editor. A vertical gray line is positioned to the left of the collapsed code region, serving as a visual cue for where to click. The code within the region is identical to the one shown in the previous screenshot.

## Click Anywhere in Area (Keyboard Shortcut)

Click anywhere in the area to be collapsed, and press Ctrl+M, Ctrl+M.

## Click Anywhere in Area (Menu Item)

Click anywhere in the area to be collapsed, and go to Edit | Outlining | Toggle Outlining Expansion on the menu bar.

After collapsing, the code area looks like this:

A screenshot of the Visual Studio code editor after the code has been collapsed. The collapsed region is represented by an ellipsis (...).

```
namespace CSBreakPoints
{
    class SomethingToDo
    {
        public void Do(Int32 iParam, Int32 iAnotherParam)...
    }
}
```

## 06.34 Using Hide Selection

<b>DEFAULT</b>	Ctrl+M, Ctrl+H
<b>VISUAL BASIC 6</b>	Ctrl+M, Ctrl+H
<b>VISUAL C# 2005</b>	Ctrl+M, Ctrl+H
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+M, Ctrl+H
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt, E, O, H
<b>MENU</b>	Edit   Outlining   Hide Selection
<b>COMMAND</b>	Edit.HideSelection
<b>VERSIONS</b>	2005 (C++ Only), 2008 (C++ Only), 2010
<b>CODE</b>	vstipEdit0036

Let's say you have a chunk of code such as the following:

```
public void ValidateForCheckout() {  
  
    //Must have 1 or more items  
    if (this.Items.Count <= 0)  
        throw new InvalidOperationException(Messages.OrderZeroItems);  
  
    //every order must have a shipping address - used for tax calcs  
    if (this.ShippingAddress == null)  
        throw new InvalidOperationException(Messages.NoAddress);  
  
    //make sure there's a payment method  
    if (this.PaymentMethod == null)  
        throw new InvalidOperationException(Messages.NoPaymentMethod);  
  
    if (this.PaymentMethod is CreditCard) {  
        CreditCard cc = this.PaymentMethod as CreditCard;  
  
        if (!cc.IsValid())  
            throw new InvalidOperationException(Messages.InvalidCreditCard);  
    }  
}
```

Outlining allows you to collapse only the entire method by default (minus sign in

the upper-left corner of the preceding illustration). What if you want to collapse only the if statements at the top? First, select the chunk of code you want to hide.

#### NOTE

You don't have to select entire lines for this feature to work. Just select as much or as little as you want to hide.

```
public void ValidateForCheckout() {  
  
    //Must have 1 or more items  
    if (this.Items.Count <= 0)  
        throw new InvalidOperationException(Messages.OrderZeroItems);  
  
    //every order must have a shipping address - used for tax calcs  
    if (this.ShippingAddress == null)  
        throw new InvalidOperationException(Messages.NoAddress);  
  
    //make sure there's a payment method  
    if (this.PaymentMethod == null)  
        throw new InvalidOperationException(Messages.NoPaymentMethod);  
  
    if (this.PaymentMethod is CreditCard) {  
        CreditCard cc = this.PaymentMethod as CreditCard;  
  
        if (!cc.IsValid())  
            throw new InvalidOperationException(Messages.InvalidCreditCard);  
    }  
}
```

Now either press Ctrl+M, Ctrl+H or go to Edit | Outlining | Hide Selection on your menu bar.

```
public void ValidateForCheckout() {  
  
    ...  
  
    //make sure there's a payment method  
    if (this.PaymentMethod == null)  
        throw new InvalidOperationException(Messages.NoPaymentMethod);  
  
    if (this.PaymentMethod is CreditCard) {  
        CreditCard cc = this.PaymentMethod as CreditCard;  
  
        if (!cc.IsValid())  
            throw new InvalidOperationException(Messages.InvalidCreditCard);  
    }  
}
```

You have successfully collapsed a selected region of code. Now you can expand and collapse the area as long as you like. Also, if you ever want to get rid of the new region, just press Ctrl+M, Ctrl+U or go to Edit | Outlining | Stop Hiding Current to remove the region.

## 06.35 Collapse to Definitions with Outlining

<b>DEFAULT</b>	Ctrl+M, Ctrl+O
<b>VISUAL BASIC 6</b>	Ctrl+M, Ctrl+O
<b>VISUAL C# 2005</b>	Ctrl+M, Ctrl+O; Ctrl+M, O
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+M, Ctrl+O
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt, E, O, O
<b>MENU</b>	Edit   Outlining   Collapse to Definitions
<b>COMMAND</b>	Edit.CollapseToDefinitions
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0032

One of the features of outlining is the Collapse To Definitions option. This feature collapses the areas for all members. Let's suppose you have the following code:

```
public decimal TaxableGoodsSubtotal
{
    get
    {
        //one of the many reasons to love LINQ :)
        return this.Items.Where(x => x.Item.IsTaxable)
            .Sum(x => x.LineTotal);
    }
}

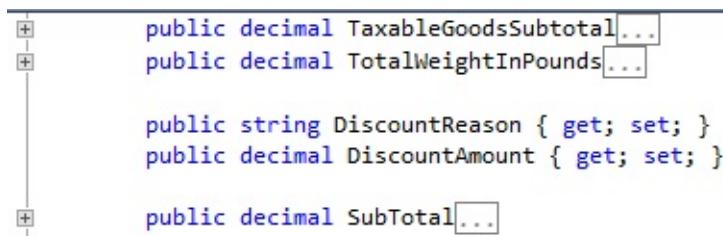
public decimal TotalWeightInPounds
{
    get
    {
        return this.Items.Sum(x => x.LineWeightInPounds);
    }
}

public string DiscountReason { get; set; }
public decimal DiscountAmount { get; set; }

public decimal SubTotal
{
    get
    {

        return this.Items.Sum(x => x.LineTotal)-DiscountAmount;
    }
}
```

You can press Ctrl+M, Ctrl+O, or you can go to Edit | Outlining | Collapse To Definitions on your menu bar to see the following result:

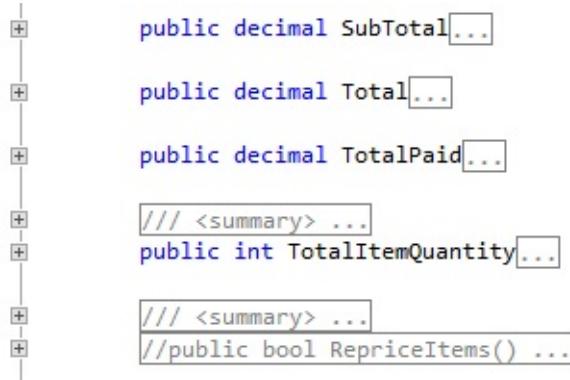


## 06.36 Cut, Copy, and Paste Collapsed Code with Outlining

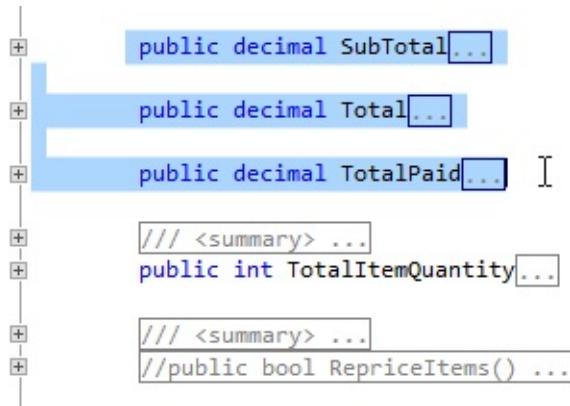
VERSIONS	2005, 2008, 2010
CODE	vstipEdit0035

When working with outlining, you can perform many timesaving operations. One of these is the ability to take a piece of code and work with it in a collapsed state.

When you collapse code using outlining (click the minus sign to the left of the signature), you get the result shown in the following illustration.



Now we can select all of that code in one compact unit:



Then simply cut or copy the code, and paste it where you want.

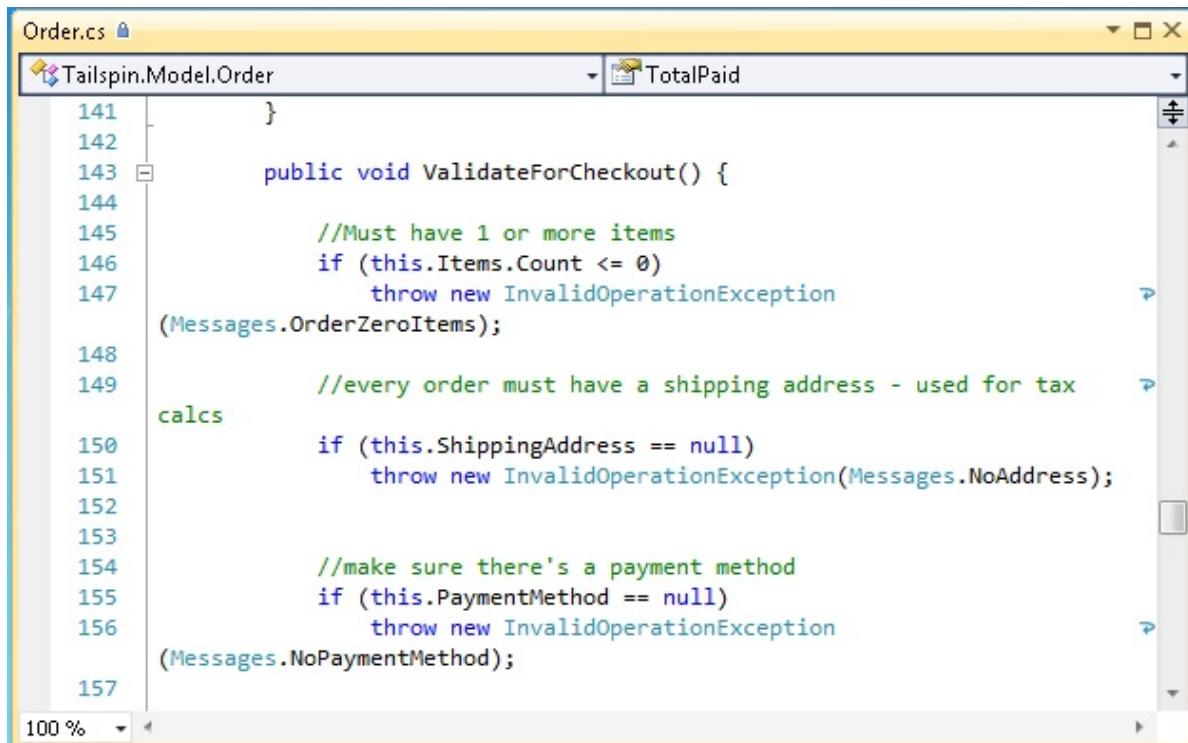
## 06.37 Understanding Word Wrap

<b>DEFAULT</b>	Ctrl+E, Ctrl+W
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	Ctrl+E, Ctrl+W; Ctrl+E, W
<b>VISUAL C++ 2</b>	Ctrl+E, Ctrl+W
<b>VISUAL C++ 6</b>	Ctrl+E, Ctrl+W
<b>VISUAL STUDIO 6</b>	Ctrl+E, Ctrl+W
<b>WINDOWS</b>	Alt, E, V, R
<b>MENU</b>	Edit   Advanced   Word Wrap; Tools   Options   Text Editor   All Languages   General
<b>COMMAND</b>	Edit.ToggleWordWrap
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0022

You can turn on or off the word wrap feature by going to Tools | Options | Text Editor | All Languages | General and selecting or clearing the Word Wrap check box.

Word wrap automatically makes sure that your text is always in the visible space. So if you already have a lot of visible space, it probably doesn't need to wrap.

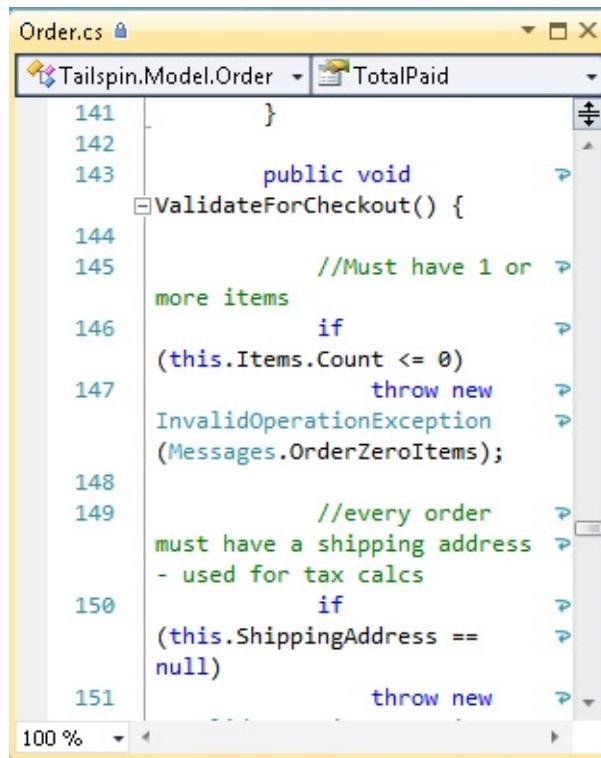
But if you have less space and wrap is turned on, it wraps automatically, as shown in the following illustration.



A screenshot of a Windows-style code editor window titled "Order.cs". The tab bar shows "Tailspin.Model.Order" and "TotalPaid". The code editor displays lines 141 through 157 of the Order.cs file. The scroll bar on the right side of the editor is partially visible, indicating that the code extends beyond the current viewable area.

```
141     }
142
143     public void ValidateForCheckout() {
144
145         //Must have 1 or more items
146         if (this.Items.Count <= 0)
147             throw new InvalidOperationException
148             (Messages.OrderZeroItems);
149
150         //every order must have a shipping address - used for tax
151         calcs
152
153         if (this.ShippingAddress == null)
154             throw new InvalidOperationException(Messages.NoAddress);
155
156         //make sure there's a payment method
157         if (this.PaymentMethod == null)
158             throw new InvalidOperationException
159             (Messages.NoPaymentMethod);
```

And, naturally, if you have almost no space, it still makes sure all the text is in the visible space.



A screenshot of a Windows-style code editor window titled "Order.cs". The tab bar shows "Tailspin.Model.Order" and "TotalPaid". The code editor displays lines 141 through 151 of the Order.cs file. The scroll bar on the right side of the editor is partially visible, indicating that the code extends beyond the current viewable area.

```
141     }
142
143     public void
144     ValidateForCheckout() {
145
146         //Must have 1 or
147         more items
148         if
149             (this.Items.Count <= 0)
150                 throw new
151                 InvalidOperationException
152                 (Messages.OrderZeroItems);
153
154         //every order
155         must have a shipping address
156         - used for tax
157         calcs
158         if
159             (this.ShippingAddress ==
160                 null)
161                 throw new
```

The arrows on the far right of each line (see the preceding illustration) show that

the lines are being wrapped. This can be turned on or off by going to Tools | Options | Text Editor | All Languages | General and checking the Show Visual Glyphs For Word Wrap check box.



## 06.38 Properties Window Keyboard Shortcuts

<b>DEFAULT</b>	F4 (properties window); Shift+F4 (property pages)
<b>VISUAL BASIC 6</b>	F4 (properties window); Shift+F4 (property pages)
<b>VISUAL C# 2005</b>	F4 (properties window); Ctrl+W, Ctrl+P (properties window); Ctrl+W, P (properties window)
<b>VISUAL C++ 2</b>	Alt+Enter (properties window); Ctrl+W (properties window); [no shortcut] (property pages)
<b>VISUAL C++ 6</b>	Alt+Enter (properties window); [no shortcut] (property pages)
<b>VISUAL STUDIO 6</b>	F4 (properties window); Shift+F4 (property pages)
<b>WINDOWS</b>	Alt,V, W (properties window); Alt,V, Y (property pages)
<b>MENU</b>	View   Properties; View   Property Pages
<b>COMMAND</b>	View.PropertiesWindow; View.PropertyPages
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0111

The Properties window allows you to view and change the design-time properties and events of selected objects that are located in editors and designers. You can also use the Properties window to edit and view file, project, and solution properties. The Properties window is available from the View menu.

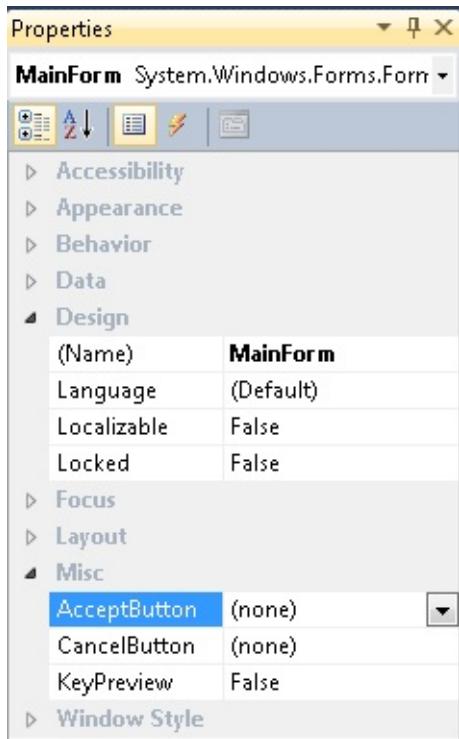
For more information, see “Properties Window” on MSDN at  
<http://msdn.microsoft.com/en-us/library/ms171352.aspx>.

## Working with the Tool Window

The following table lists the keyboard shortcuts that you can use with the Properties window:

ACTION	SHORTCUT
Open / Show	F4 or Alt+Enter
Close	Shift+Esc

# Working with Categories



The following table lists the keyboard shortcuts that you can use when working with the categories in the Properties window:

## NOTE

The category needs to be selected as shown in the preceding illustration, using the **Misc** category. You can use your Arrow keys to select the node.

ACTION	SHORTCUT
Collapse	Left Arrow or - (minus)
Expand	Right Arrow or + (plus)
Show next set of properties	Page Up
Show previous set of properties	Page Down
Move to next item in list	Down or Left Arrow
Move to previous item in list	Up or Right Arrow
Move to first property	Home

| Move to last property | End |

## Property Items



The following table lists the keyboard shortcuts that you can use when you are working with individual items in the Properties window:

ACTION	SHORTCUT
Cancel current changes	Esc
Show drop-down list	Alt+Down Arrow
Previous/next option in list	Up / Down Arrow
Cut selection	Ctrl+X
Copy selection	Ctrl+C
Paste	Ctrl+V
Undo	Ctrl+Z

## 06.39 Document Outline: Web Projects

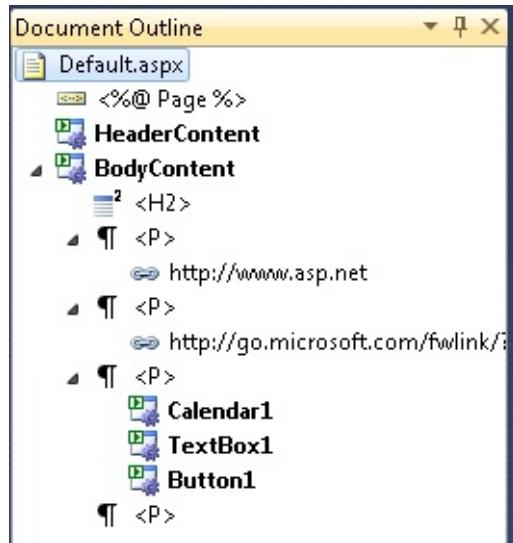
<b>DEFAULT</b>	Ctrl+Alt+T
<b>VISUAL BASIC 6</b>	Ctrl+Alt+T
<b>VISUAL C# 2005</b>	Ctrl+Alt+T; Ctrl+W, Ctrl+U; Ctrl+W, U
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+Alt+D
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+T
<b>WINDOWS</b>	Alt,V, E, D
<b>MENU</b>	View   Other Windows   Document Outline
<b>COMMAND</b>	View.DocumentOutline
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipTool0116

If you are working with web projects, you need to know about the Document Outline feature (Ctrl+Alt+T). It can be used to do the following:

- View the logical structure of your document.
- Determine which elements are HTML elements and which ones are web server controls.
- Navigate to specific elements, in Design view and in Source view.

If you are working in Source view, it shows you the body element and the child elements of the head element, the page directive, and any script elements and code elements.

I created a default web application in Visual Studio 2010, added a couple of server controls, and switched to Source view. The following illustration shows what the Document Outline looks like.



Notice that you can clearly distinguish between the HTML and server content (server content has the little gears in them). It can also be used to select items in the Source.

Document Outline

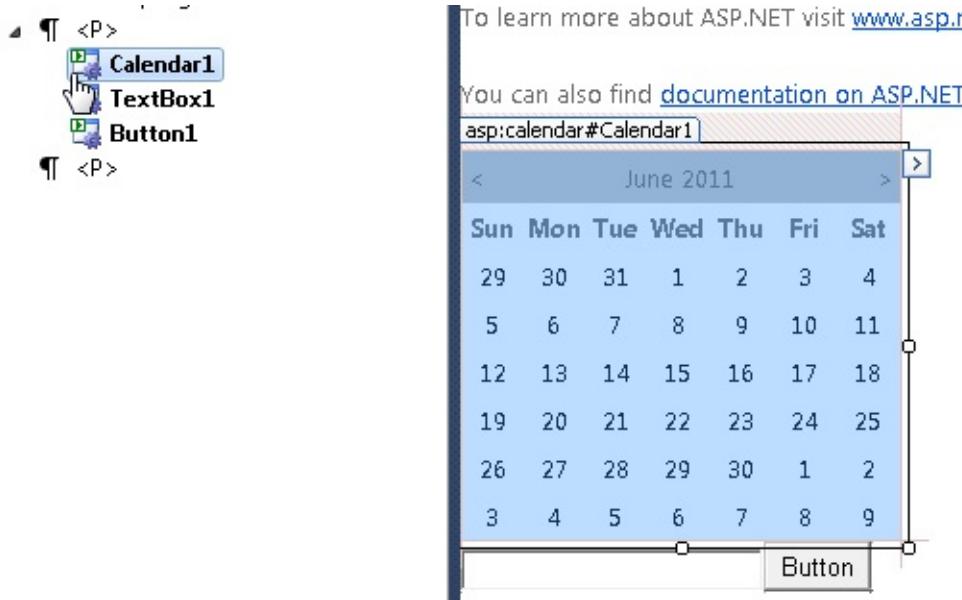
Default.aspx

- <%@ Page %>
- HeaderContent
- BodyContent
  - <H2>
  - <P>
    - http://www.asp.net
    - http://go.microsoft.com/fwlink/?
  - <P>
    - Calendar1
    - TextBox1
    - Button1
  - <P>

Client Objects & Events

Line Number	Object / Event	Code
7		<h2>
8		We1
9		</h2>
10		<p>
11		To
12		</p>
13		<p>
14		You
15		</p>
16		</p>
17		<p>
18		<as>
19		<as>
20		<as>

In Design view, it essentially does the same thing and allows you to get a bird's-eye view of the layout and can be used to select controls.



If you have a scenario with many controls, this makes getting to specific items very easy.

The screenshot shows the ASP.NET designer interface with a more complex visual tree. The `BodyContent` section contains an `<H2>` element, several `<P>` elements with links, and a `Calendar1` control along with `TextBox1` and `Button1`. The source code pane shows the generated HTML and server-side code:

```

    <H2>
        <P>
            <a href="http://www.asp.net">To learn more about ASP.NET visit www.asp.net</a>
        <P>
            <a href="http://go.microsoft.com/fwlink/?LinkId=169433" style="color:blue">Documentation on MSDN</a>
        <P>
            <Calendar1>
            <TextBox1>
            <Button1>
        <P>

```

Additionally, many people like to use this feature with Split view so that it can show both Design and Source.

## 06.40 Inserting Code Snippets

<b>DEFAULT</b>	Ctrl+K, Ctrl+X
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+X
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+X; Ctrl+K, X
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+X
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+X
<b>WINDOWS</b>	Alt+E, I, I
<b>MENU</b>	Edit   IntelliSense   Insert Snippet
<b>COMMAND</b>	Edit.InsertSnippet
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipEdit0051

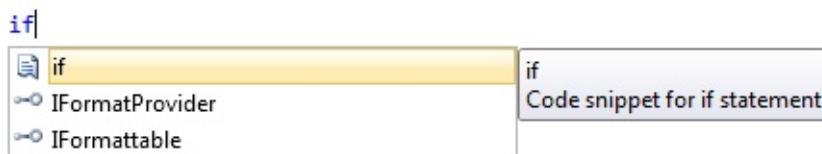
Even though these have been around for a while, a lot of people still don't fully understand snippets. Let's assume you are in the editor and you want to create an if statement. You can type it out, but that would be the hard way. Why not just use a snippet? You have several ways to do this.

### Tab

Just type **if**, and press the Tab key twice.

### C#

The paper-like icon, shown in the following illustration, indicates that this is a snippet in C#.

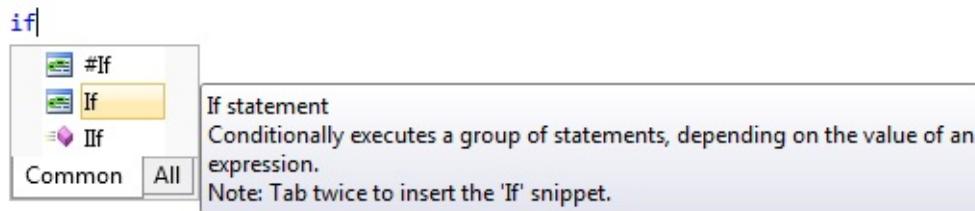


### VB

In VB, you don't have the paper icon, but you do have a note in the tooltip that indicates this is a snippet.

#### WARNING

Don't be fooled by the `#if` directive. You want the if statement for this example.

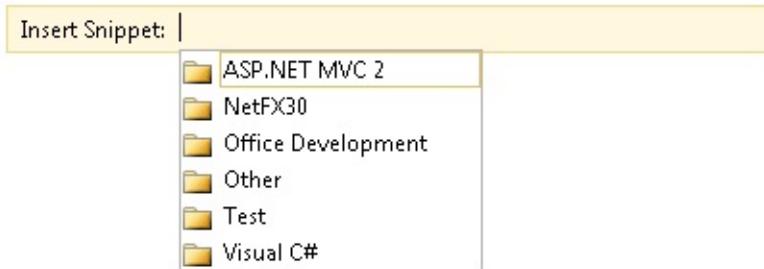


## Keyboard Shortcut and Context Menu

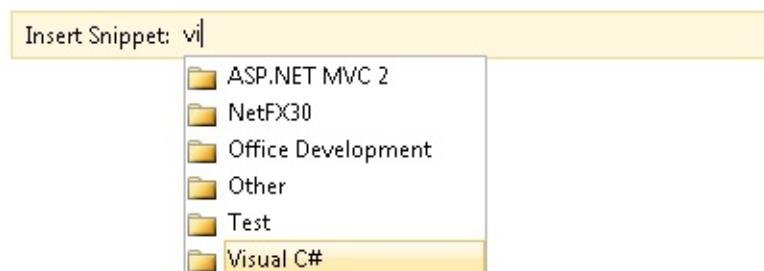
Press **Ctrl+K**, **Ctrl+X** or right-click and choose Insert Snippet, or go to **Edit | IntelliSense | Insert Snippet** on the menu bar.

### C#

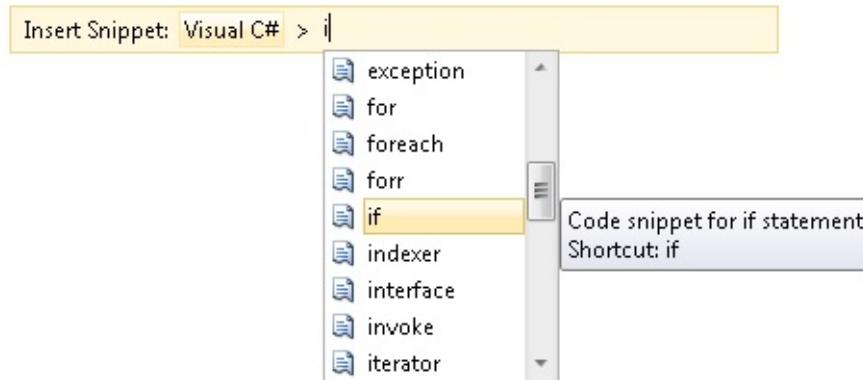
First you see a prompt to insert the snippet, as shown in the following illustration.



Choose **Visual C#** from the list, and then press **Enter**.

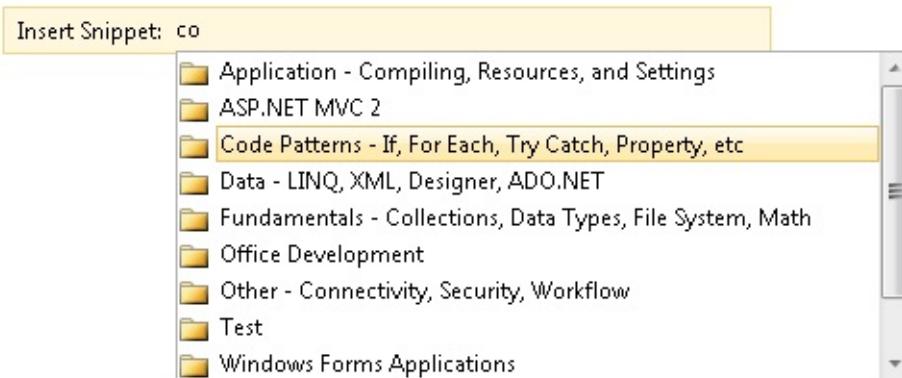


Then type **if**, and press **Enter**.

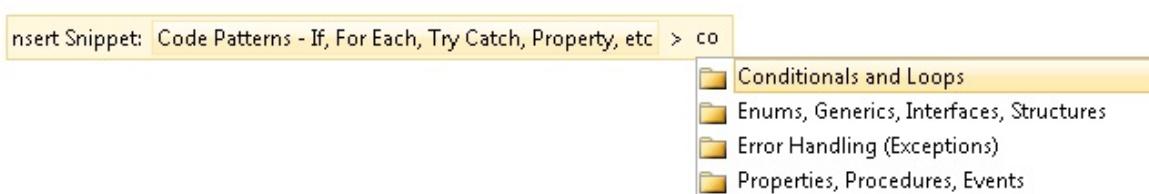


## VB

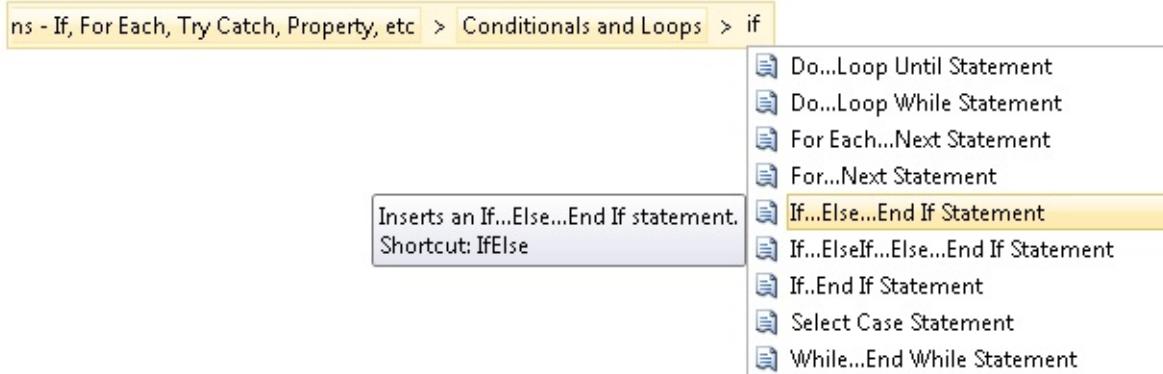
Similar to C#, you see a prompt to insert the snippet, but the path is different. From the list, choose Code Patterns – If, For Each, Try Catch, Property, Etc, and then press Enter:



Then choose Conditionals And Loops, and press Enter:



Finally, pick the if statement you want from the list:



## Result

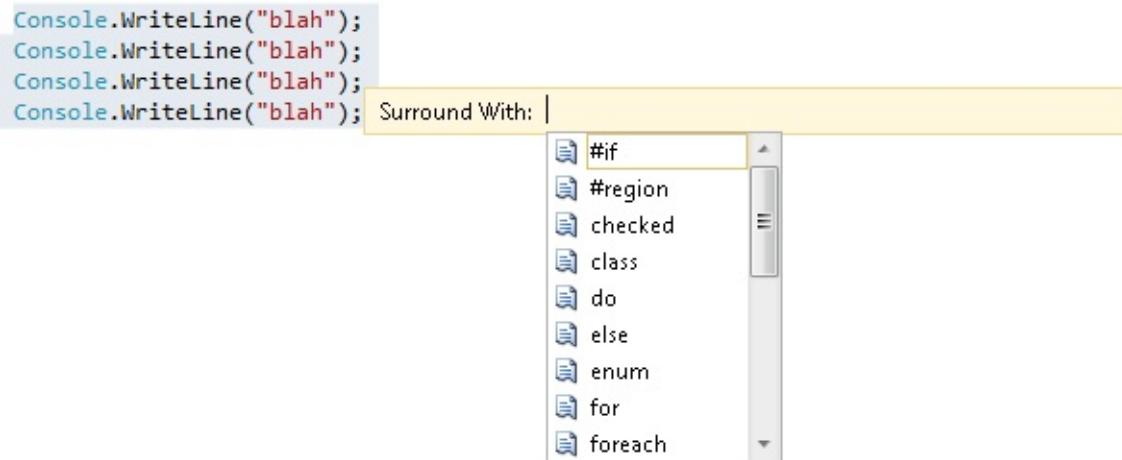
No matter which option you choose, the result is essentially the same. You can now type in your condition and the rest of your logic.

## 06.41 Surround with a Code Snippet

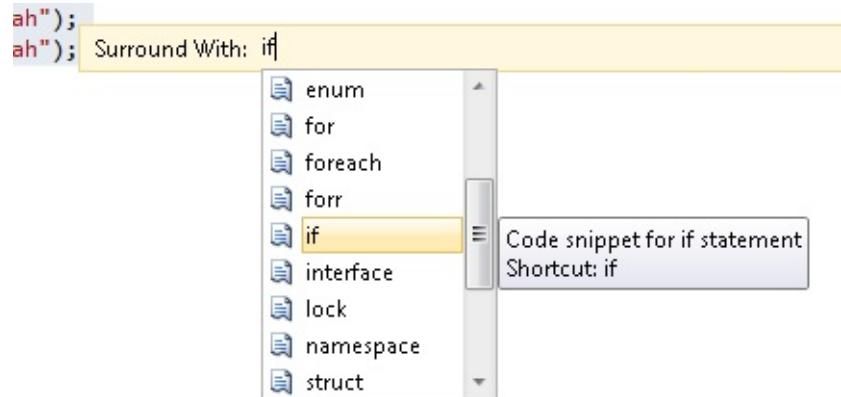
<b>DEFAULT</b>	Ctrl+K, Ctrl+S
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+S
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+S; Ctrl+K, S
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+S
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+S
<b>WINDOWS</b>	Alt,E, I, S
<b>MENU</b>	Edit   IntelliSense   Surround With
<b>COMMAND</b>	Edit.SurroundWith
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#
<b>CODE</b>	vstipEdit0052

This is one that even people who know about snippets tend to forget. You can actually put a snippet around existing code, assuming you have some code selected.

Just press Ctrl+K, Ctrl+S:



Then type the statement you want to surround the code with. In this case, let's use an if statement:



Press your Tab key once, and you see the result shown in the following illustration.

```
if (true)
{
    Console.WriteLine("blah");
    Console.WriteLine("blah");
    Console.WriteLine("blah");
    Console.WriteLine("blah");
}
```

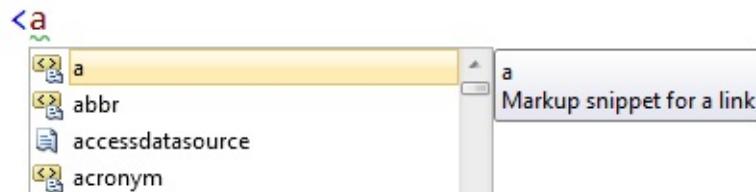
Now you can put in your condition and any additional logic you want.

## 06.42 Using Code Snippets

<b>DEFAULT</b>	Ctrl+K, Ctrl+X
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+X
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+X; Ctrl+K, X
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+X
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+X
<b>WINDOWS</b>	Alt+E, I, I
<b>MENU</b>	Edit   IntelliSense   Insert Snippet
<b>COMMAND</b>	Edit.InsertSnippet
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipEdit0053

I thought it would be a good idea to review how to use snippets. This example uses tags in ASP.NET, but the concepts apply across the board to all snippets.

Let's say you are in an ASP.NET application and you start to type an anchor tag, as shown in the following illustration.



You press Tab twice to insert a snippet and see the following result:

```
<a href="#">content</a>
```

Snippets have special areas that you can change the values in by pressing Tab to cycle through them. For example, the following snippet has two special areas. Let's put in a URL:

```
<a href="http://blah.com">content</a>
```

Now press Tab to go to the next area, and type in some text as shown in the following illustration.

```
<a href="http://blah.com">this is cool</a>
```

You can keep pressing Tab to toggle between these two locations as much as you want. When you are done, you can press Enter to put the cursor at the end of the element and continue typing more code.

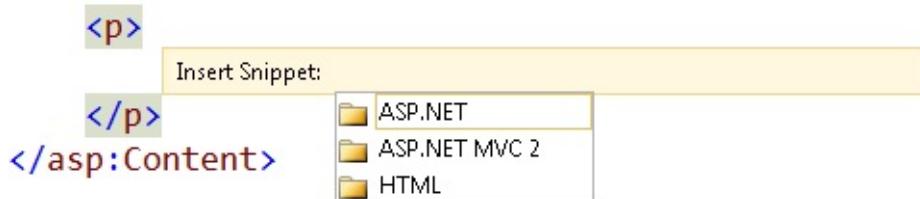
So, in a nutshell, that's how to work with snippets. They take a little getting used to, but they are great timesavers.

## 06.43 HTML Code Snippets

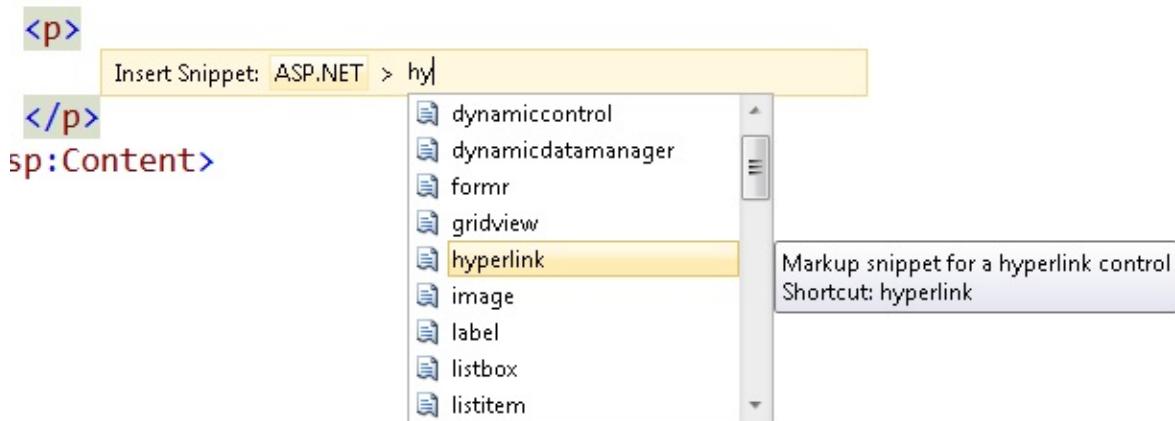
<b>DEFAULT</b>	Ctrl+K, Ctrl+X
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+X
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+X; Ctrl+K, X
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+X
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+X
<b>WINDOWS</b>	Alt,E, I, I
<b>MENU</b>	Edit   IntelliSense   Insert Snippet
<b>COMMAND</b>	Edit.InsertSnippet
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEdit0018

You might have used code snippets for a while now, but did you know that Visual Studio 2010 has added support for HTML snippets?

From any webpage, go to Edit | IntelliSense | Insert Snippet on the menu bar (Ctrl+K, Ctrl+X). You should see the Insert Snippet prompt:



Next, choose your category and the specific snippet you want:



Type in the details, pressing Tab if there are multiple areas where you need to put in details.

```
<p>
    <a href="http://blah.com">content</a>
</p>
</asp:Content>
```

Press Enter when you are done, and continue putting in your code.

#### NOTE

This is the long way to insert a snippet the quicker way, in most cases, is to just start typing an element and hit tab twice to insert the snippet for that element.

## 06.44 JavaScript Code Snippets

<b>DEFAULT</b>	Ctrl+K, Ctrl+B
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+B
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+B
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+B
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+B
<b>WINDOWS</b>	Alt,E, I, I
<b>MENU</b>	Edit   IntelliSense   Insert Snippet
<b>COMMAND</b>	Edit.InsertSnippet
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEdit19

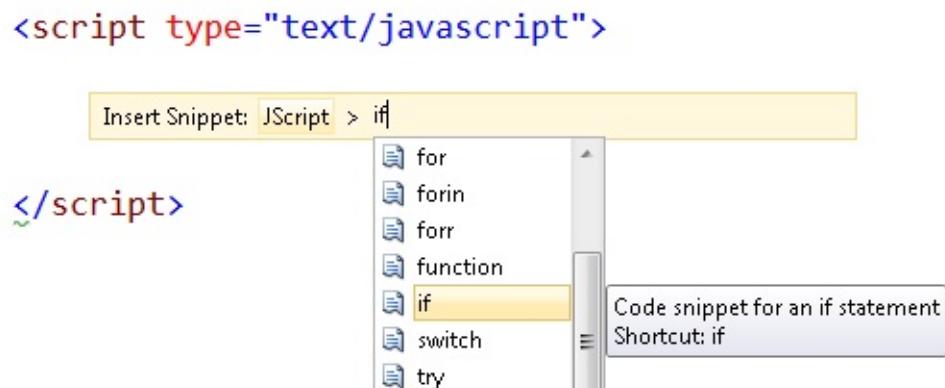
Want to get that JavaScript code created more quickly? Use JavaScript snippets.

When you are inside any script block, go to Edit | IntelliSense | Insert Snippet on the menu bar (Ctrl+K, Ctrl+X), choose the JScript category, as shown in the following illustration, and then press Tab.

```
<script type="text/javascript">
```



Now pick what you want to do, and then press Tab again.



Now fill in any details, pressing Tab to cycle between details if needed.

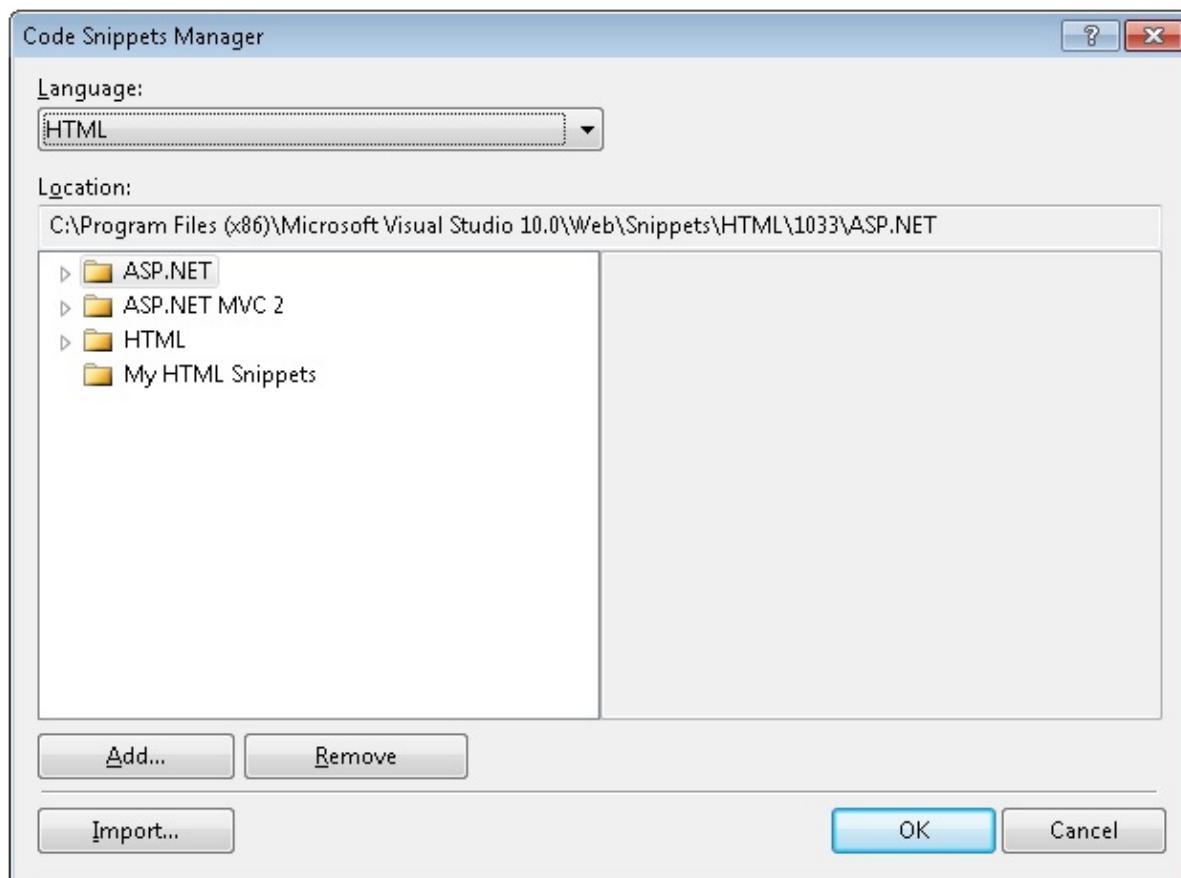
```
<script type="text/javascript">  
    if (true) {  
    }  
</script>
```

Then press Enter when you are done, and keep typing your code.

## 06.45 Using the Code Snippets Manager

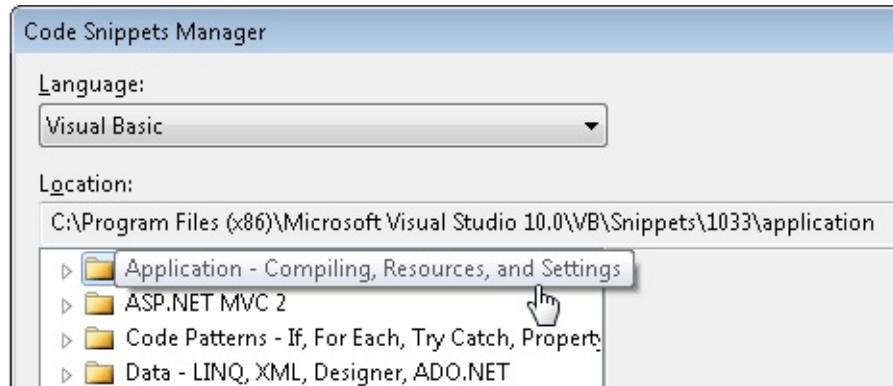
<b>DEFAULT</b>	Ctrl+K, Ctrl+B
<b>VISUAL C++ 2</b>	[no shortcut]
<b>WINDOWS</b>	Alt,T, T
<b>MENU</b>	Tools   Code Snippets Manager
<b>COMMAND</b>	Tools.CodeSnippetsManager
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0015

Ever wonder where you can get a list of code snippets just to browse? The Code Snippets Manager is your friend. Start by pressing Ctrl+K, Ctrl+B to see the following dialog box.

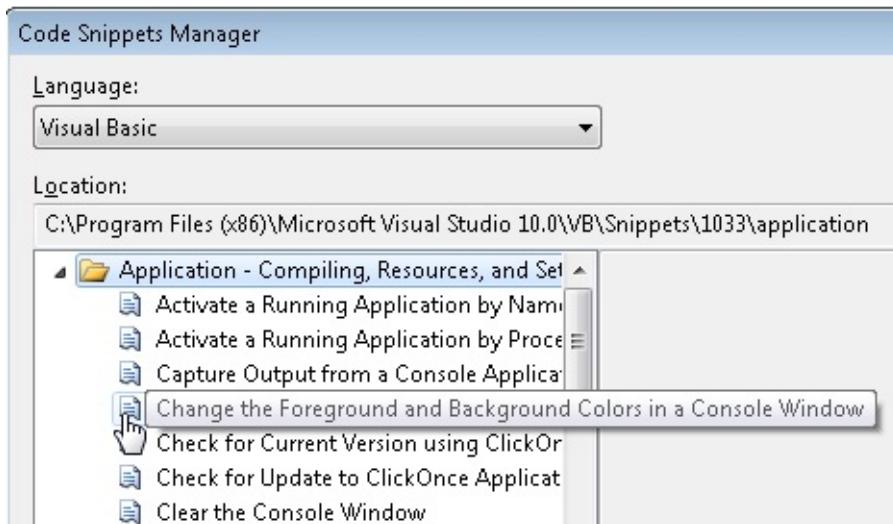


The first thing you need to do is pick the language you want to browse through. Go

ahead and choose Visual Basic for this example. This action gives you several folders (categories) to choose from. Let's choose the Application – Compiling, Resources, And Settings folder, as shown in the following illustration.



From here, you can browse individual snippets and put your mouse over each of them to get the full description:



Click the Change The Foreground And Background Colors In A Console Window snippet. Notice that summary information about the snippet is provided:

A screenshot of the 'Code Snippets Manager' window showing detailed information for the selected snippet. The 'Description' field states: 'Changes the background and text color of the console window.' The 'Shortcut' field is 'appChanCol'. The 'Author' field is 'Microsoft Corporation'. The left pane shows the expanded 'Application - Compiling, Resources, and Settings' category with its sub-snippets.

To use this snippet, you have to copy the shortcut name, “appChanCol” in this case. Now click OK or Cancel to close the dialog box, and paste your shortcut name into the editor.

```
Sub Main()  
    appChanCol|  
End Sub
```

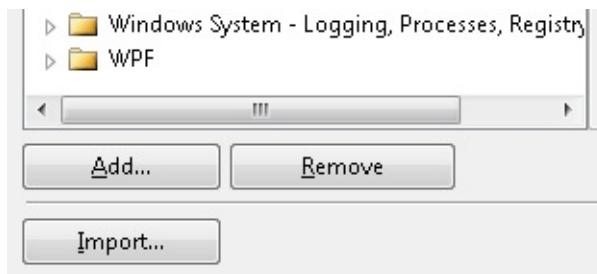
Finally, press Tab twice to get the results shown in the following illustration.

**NOTE**

Technically, you have to press Tab only once in this case, but pressing it twice preserves muscle memory for the snippets and has no downside.

```
Sub Main()  
    Console.BackgroundColor = ConsoleColor.DarkRed  
    Console.ForegroundColor = ConsoleColor.Gray  
    Console.Clear()  
  
End Sub
```

But what about the buttons in the Code Snippet Manager we didn’t talk about?



Following are quick descriptions of what they do.

- **Add**—Lets you pick a folder to add to the current list of folders for snippets. The folder can be any accessible location, including network drives. This is useful for shared snippets among team members where everyone uses snippets on a shared drive.

- **Remove**—Takes a folder out of the list but does not physically delete the folder.
- **Import**—Used for including individual .snippet files in the folder you are currently viewing.

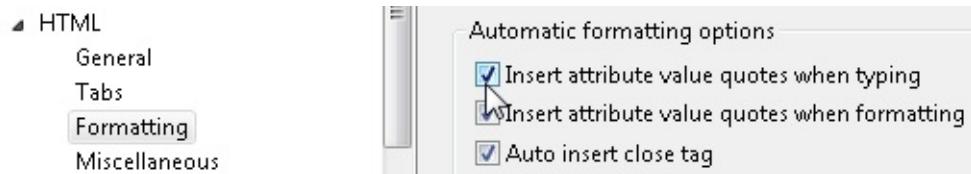
## 06.46 Insert Quotes When Typing Attribute Values

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0082

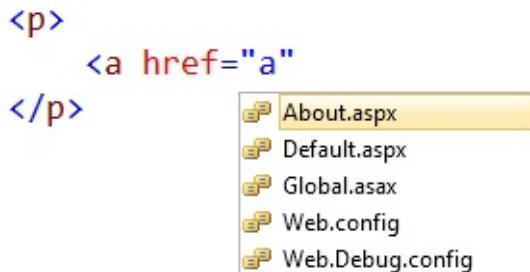
One of the more annoying things about web development in Visual Studio is that when you type in attributes it doesn't include the quotes automatically.



You can have quotes included automatically when you type in attributes if you go to Tools | Options | Text Editor | HTML | Formatting and select Insert Attribute Value Quotes When Typing:



From now on, you will see the quotes when you type in the attributes:



## 06.47 Format the Current Document or Selection (Web)

<b>DEFAULT</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+F (selection)
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+F (selection)
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+E (document); Ctrl+K, E (document); Ctrl+K, Ctrl+F (selection); Ctrl+E, Ctrl+F (selection); Ctrl+E, F (selection)
<b>VISUAL C++ 2</b>	[no shortcut] (document); Ctrl+Shift+F (selection); Ctrl+Alt+I (selection)
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+F (selection); Alt+F8 (selection)
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+F (selection); Alt+F8 (selection)
<b>WINDOWS</b>	Alt,E, V, A (document); Alt,E, V, F (selection)
<b>MENU</b>	Edit   Advanced   Format Document; Edit   Advanced   Format Selection
<b>COMMAND</b>	Edit.FormatDocument; Edit.FormatSelection
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEdit0089

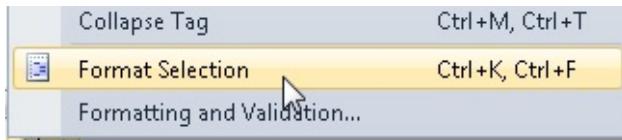
In vstipEdit0057 ([AX.63 Format the Current Document or Selection](#), in Appendix B [<http://go.microsoft.com/fwlink/?LinkId=223758>]), I showed you how to format code. I thought it would be a good idea to look at the same task from a web project perspective. Let's say you have the following HTML:

```
<div> something cool here </div>
```

### WARNING

You need to have spaces around the <DIV> tags, as I have shown here, or the formatting will not work correctly.

But you want to clean it up a bit. Just select everything you want formatted, right-click, and choose Format Selection (Ctrl+K, Ctrl+F), as shown in the following illustration.



You see the following result:

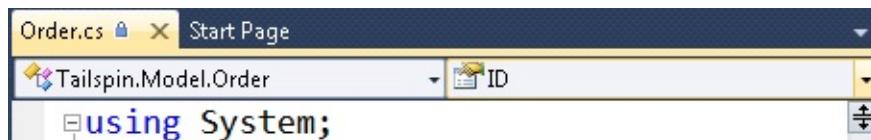
```
<div>
    something cool here
</div>
```

The formatting rules that govern this operation can be found at Tools | Options | Text Editor | HTML | Formatting. You can also format the entire document by going to Edit | Advanced | Format Document or using the shortcut Ctrl+K, Ctrl+D.

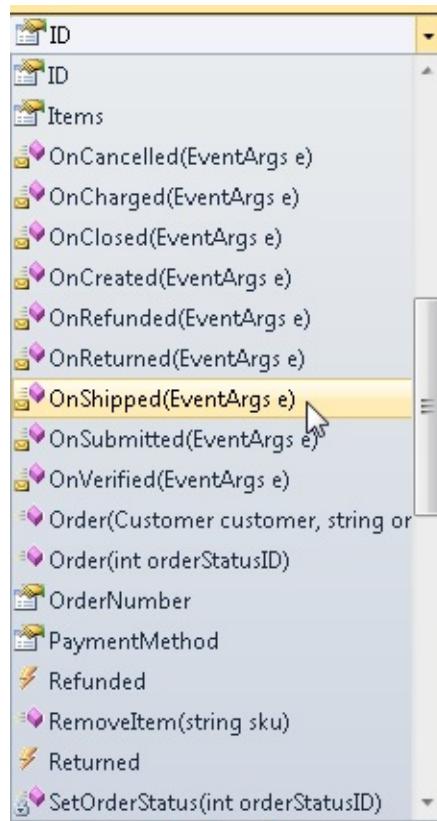
## 06.48 Using the Navigation Bar

<b>DEFAULT</b>	Ctrl+F2 (navigation bar); Tab (move between Objects and Members drop-down lists)
<b>VISUAL BASIC 6</b>	Ctrl+F2 (navigation bar); Tab (move between Objects and Members drop-down lists)
<b>VISUAL C# 2005</b>	Ctrl+F2 (navigation bar); Tab (move between Objects and Members drop-down lists)
<b>VISUAL C++ 2</b>	Ctrl+F2 (navigation bar); Tab (move between Objects and Members drop-down lists)
<b>VISUAL C++ 6</b>	Ctrl+F8 (navigation bar); Tab (move between Objects and Members drop-down lists)
<b>VISUAL STUDIO 6</b>	Ctrl+F2 (navigation bar); Tab (move between Objects and Members drop-down lists)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Window.MoveToNavigationBar
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0026

Ever notice the two drop-down lists just below the file tab channel? The one on the left is the Objects list (classes and objects), and the one on the right is the Members list.



You probably don't have much use for the Objects list unless you have a lot of classes in one file. The Members list is extremely useful when you want to jump around in your code. If you have a class with several functions, you can click the Members list to see them all:



Now you can just click a member to instantly go to that section in your code.

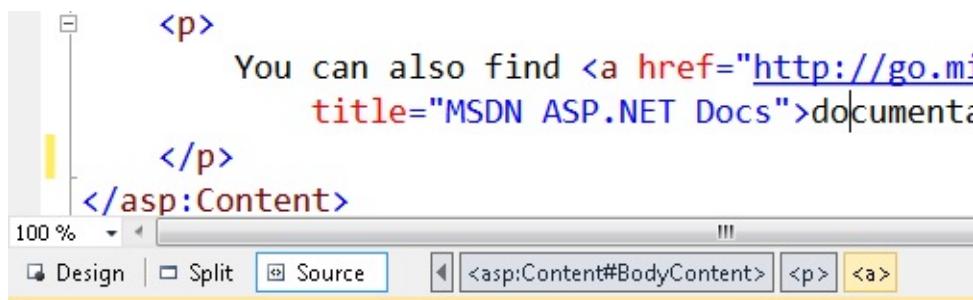
## 06.49 HTML Editor Tag Navigation

VERSIONS	2005, 2008, 2010
CODE	vstipEdit0091

According to the documentation, “the tag navigator is a representation of the element that is currently selected in the document, along with the hierarchy of parent tags to which it belongs.”

For more detailed information, see the topic “HTML Editor Tag Navigation in Visual Web Developer” at <http://msdn.microsoft.com/en-us/library/b53y76zk.aspx>.

The tag navigator is particularly useful for working with deeply nested structures such as tables within tables. You can use the tag navigator to determine which element in the document has the focus.



In addition, you can use the tag navigator to move from the current element to an element that is located higher in the current hierarchy:



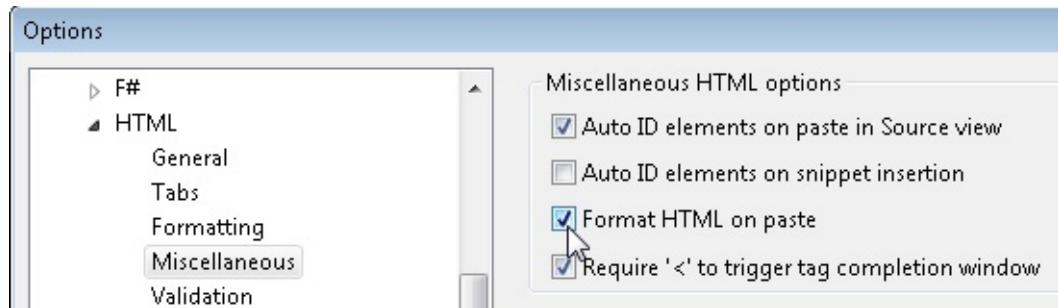
### NOTE

The tag navigator does not display all of the elements in the current document. Instead, it shows the path from the current element to the outermost parent. For information about how to see all of the elements in the document, see vstipTool0116 (06.39 Document Outline: Web Projects, page 251).

## 06.50 Format HTML on Paste

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options Text Editor   HTML   Miscellaneous
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0092

If you find yourself copying and pasting unformatted HTML a lot, this tip is for you. Just go to Tools | Options | Text Editor | HTML | Miscellaneous, and select Format HTML On Paste.



From now on, when you paste it, your HTML is formatted according to the rules set up in Tools | Options | Text Editor | HTML | Formatting.

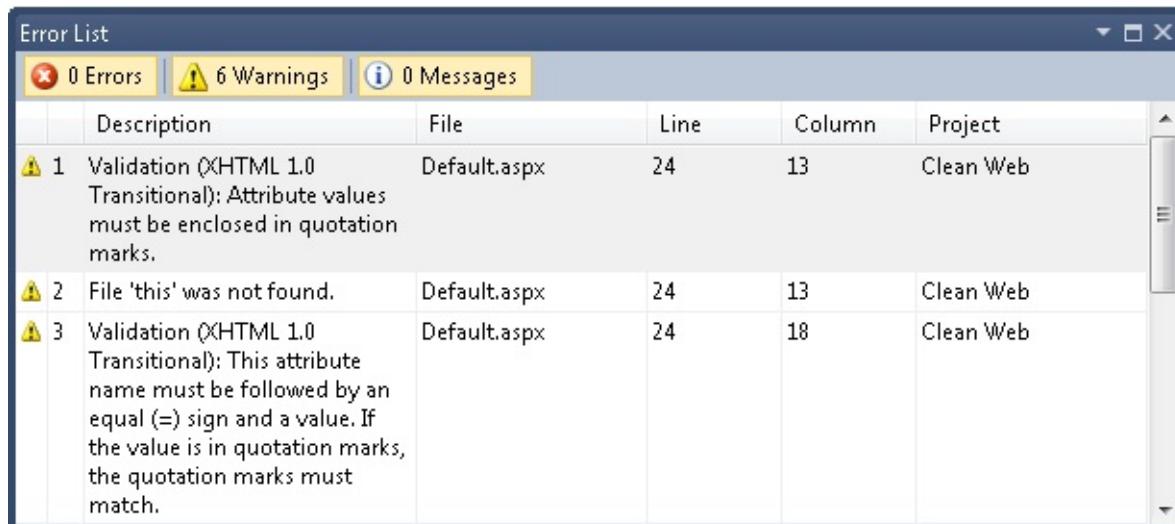
## 06.51 Display HTML/CSS Warnings as Errors

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options Text Editor   HTML   Validation
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0084

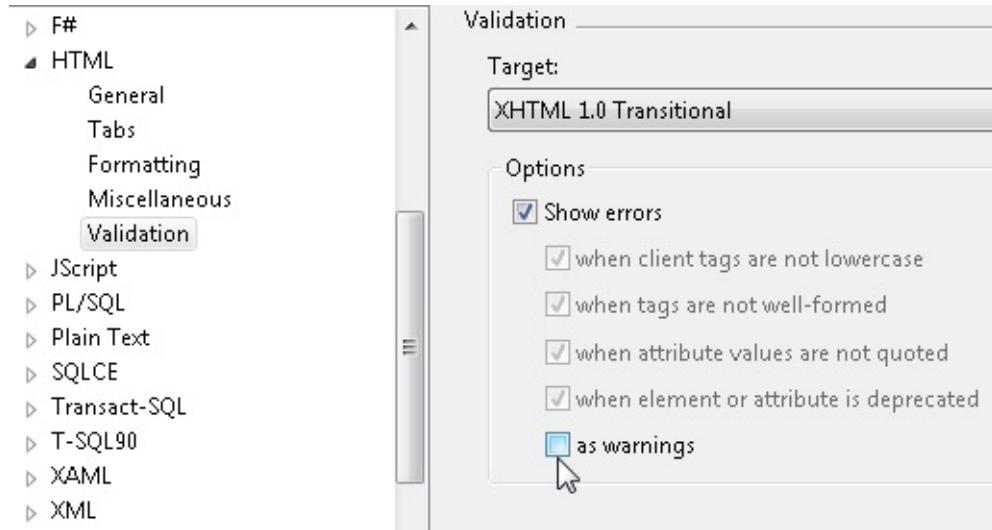
Normally, HTML and CSS syntax problems show up as warnings (green squiggles):

```
<a href=this is cool.aspx></a>
```

These syntax problems also show up as warnings in the Error List window:



This means that you can build and run the application if you choose to ignore the warnings. Or you can have them show up as errors instead, by going to Tools | Options | Text Editor | HTML | Validation and clearing the As Warnings check box.



Now the green squiggles will be red:

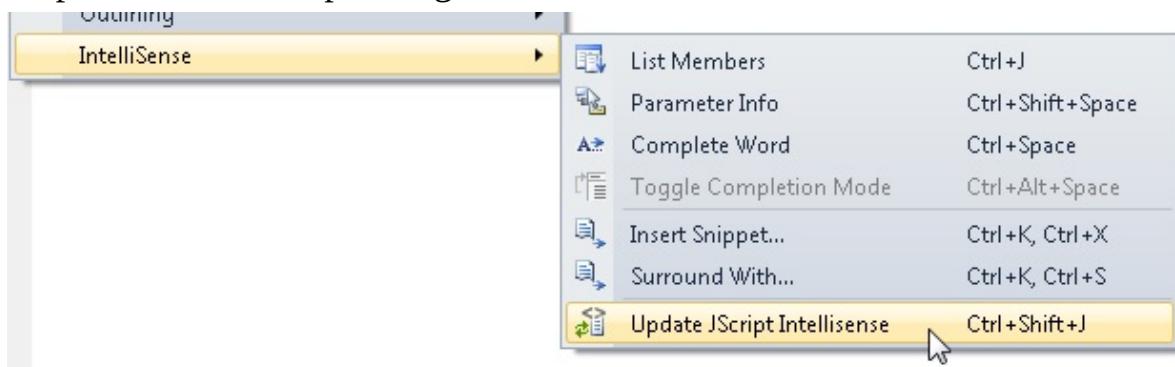
```
<a href=this is cool.aspx></a>
```

The previous warnings now show up as errors in the Error List window and you are notified when you try to build that a problem exists.

## 06.52 Updating JScript IntelliSense

<b>DEFAULT</b>	Ctrl+Shift+J
<b>WINDOWS</b>	Alt+E, I, J
<b>MENU</b>	Edit   IntelliSense   Update JScript IntelliSense
<b>COMMAND</b>	Edit.UpdateJScriptIntellisense
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEdit0086

If you find that your IntelliSense isn't showing for JScript items you recently put in, you can update the JScript IntelliSense by going to Edit | IntelliSense | Update JScript IntelliSense or pressing Ctrl+Shift+J.



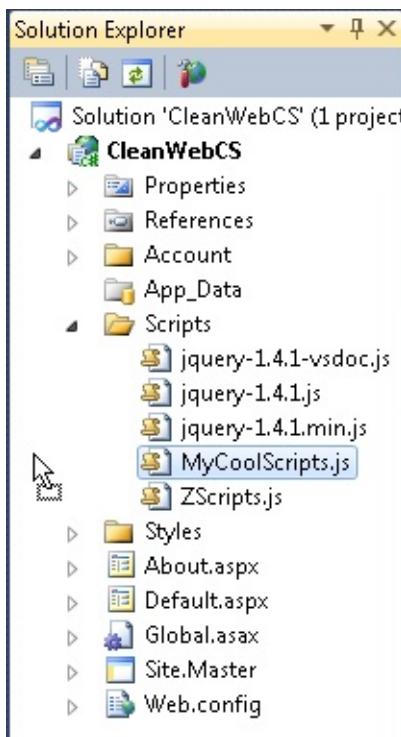
It goes by quickly, but you might see the **06.52 Updating JScript IntelliSense** message in the Status Bar at the lower-left corner of your screen.

When it is done, you should be able to see the newly added items.

## 06.53 Using JScript Libraries in Other JScript Files

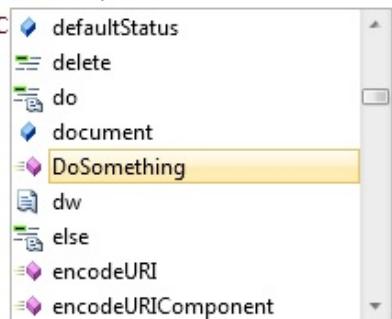
VERSIONS	2008, 2010
CODE	vstipProj0025

When you want to use your own custom JScript library, it's a very straightforward process. You just click and drag the file from Solution Explorer into your web page.



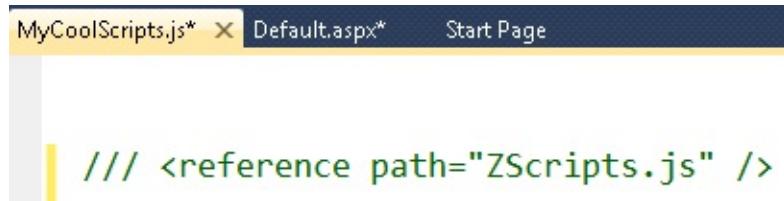
Then start using the library:

```
<script src="Scripts/MyCoolScripts.js" type="text/javascript">
    dosom|
```

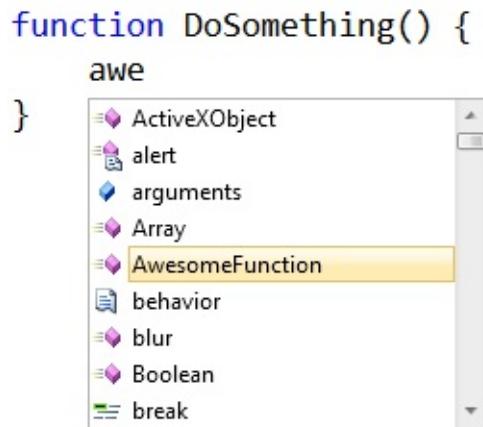


But what if you want to use the library in another JScript file? No problem. To have one file used by another, just click and drag the file name from the resource file into the consuming file.

When you do, it makes a reference automatically:



And you can start using it right away with IntelliSense now aware of the resource contents:

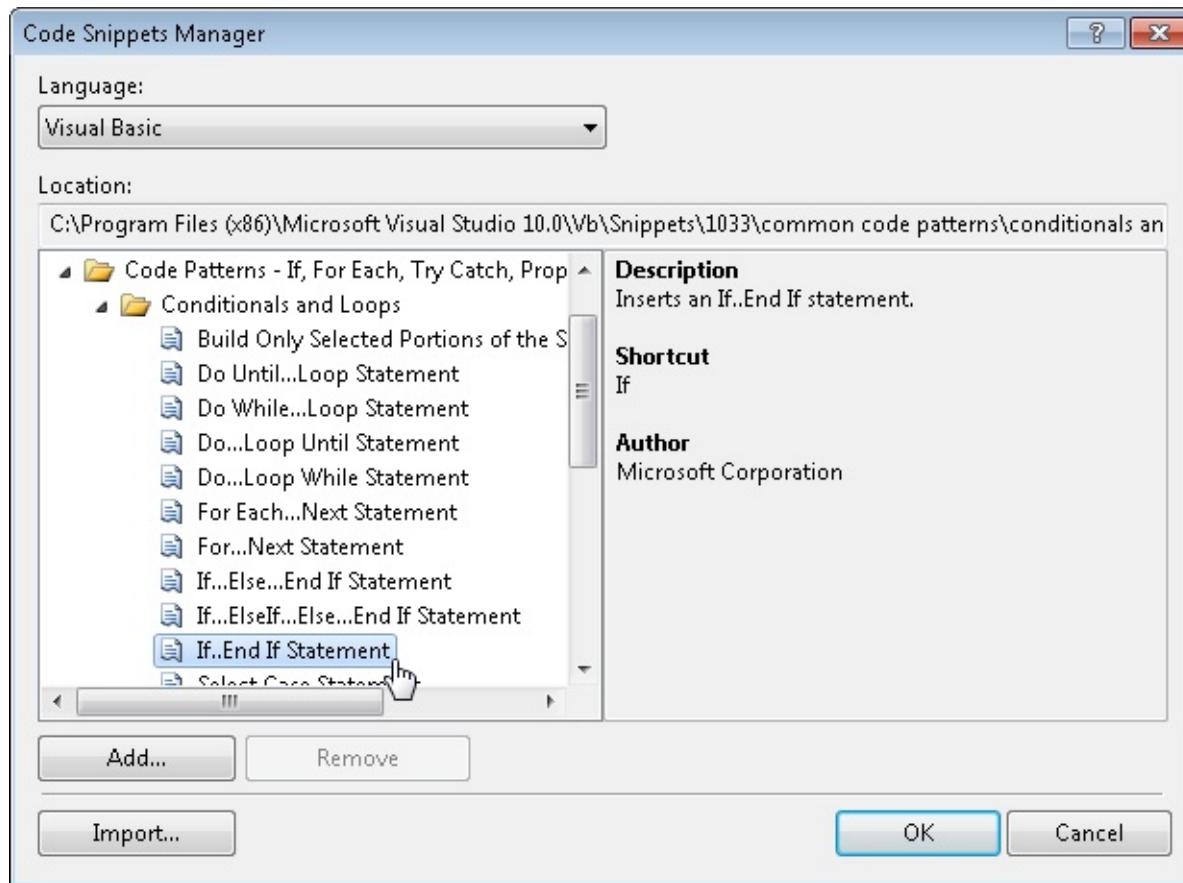


## 06.54 Create New Code Snippets from Existing Ones

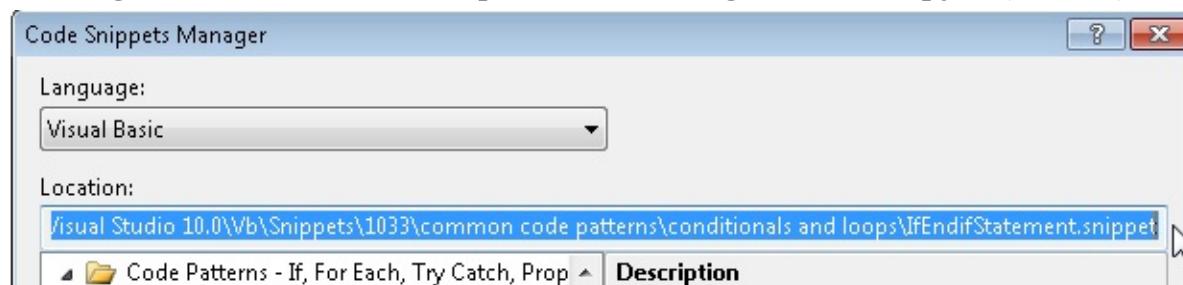
<b>DEFAULT</b>	Ctrl+K, Ctrl+B
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+B
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+B
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+B
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+B
<b>WINDOWS</b>	Alt,T, T
<b>MENU</b>	Tools   Code Snippets Manager
<b>COMMAND</b>	Tools.CodeSnippetsManager
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0016

When you use snippets, you often need to change the default values. Naturally, you can modify the values after the snippet is used. But what if you want to permanently change the value and make a new snippet out of it?

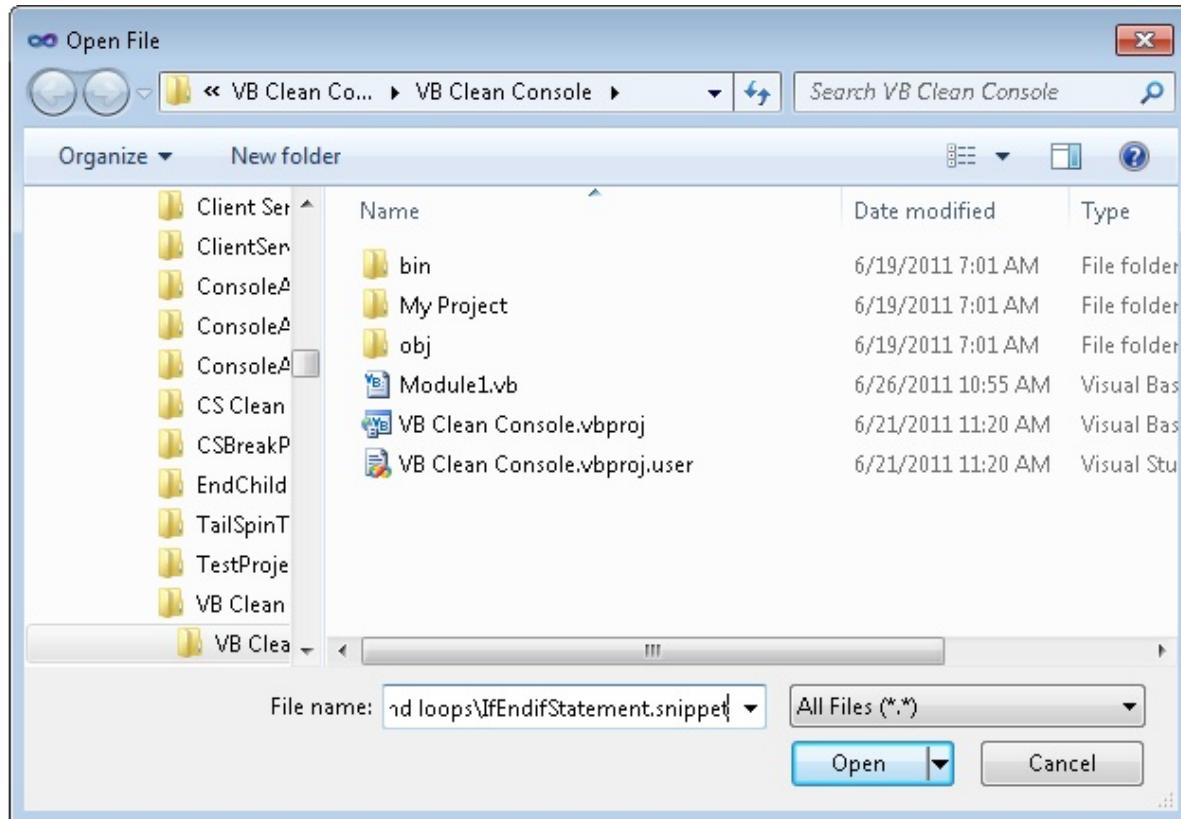
First, figure out which snippet you want to modify by finding it in the Code Snippets Manager (see vstipTool0015, [06.45 Using the Code Snippets Manager](#), in Appendix B [<http://go.microsoft.com/fwlink/?LinkId=223758>]). In this case, let's modify the If..End If snippet for Visual Basic.



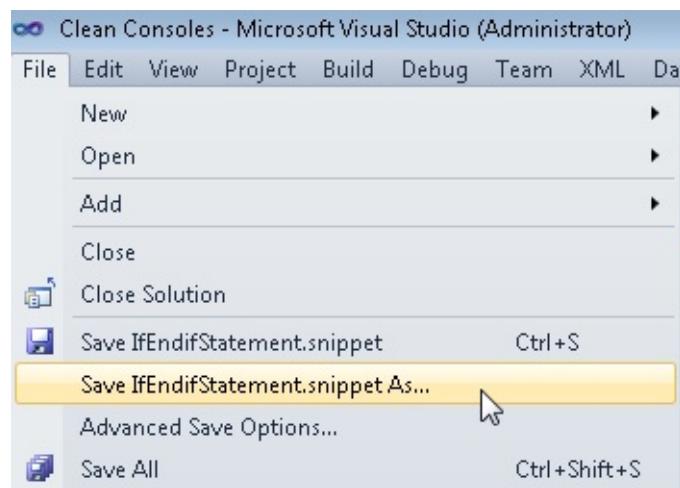
What you are after is the location of the snippet. See the path under Location in the following illustration. Select the path in the dialog box and Copy it (Ctrl+C).



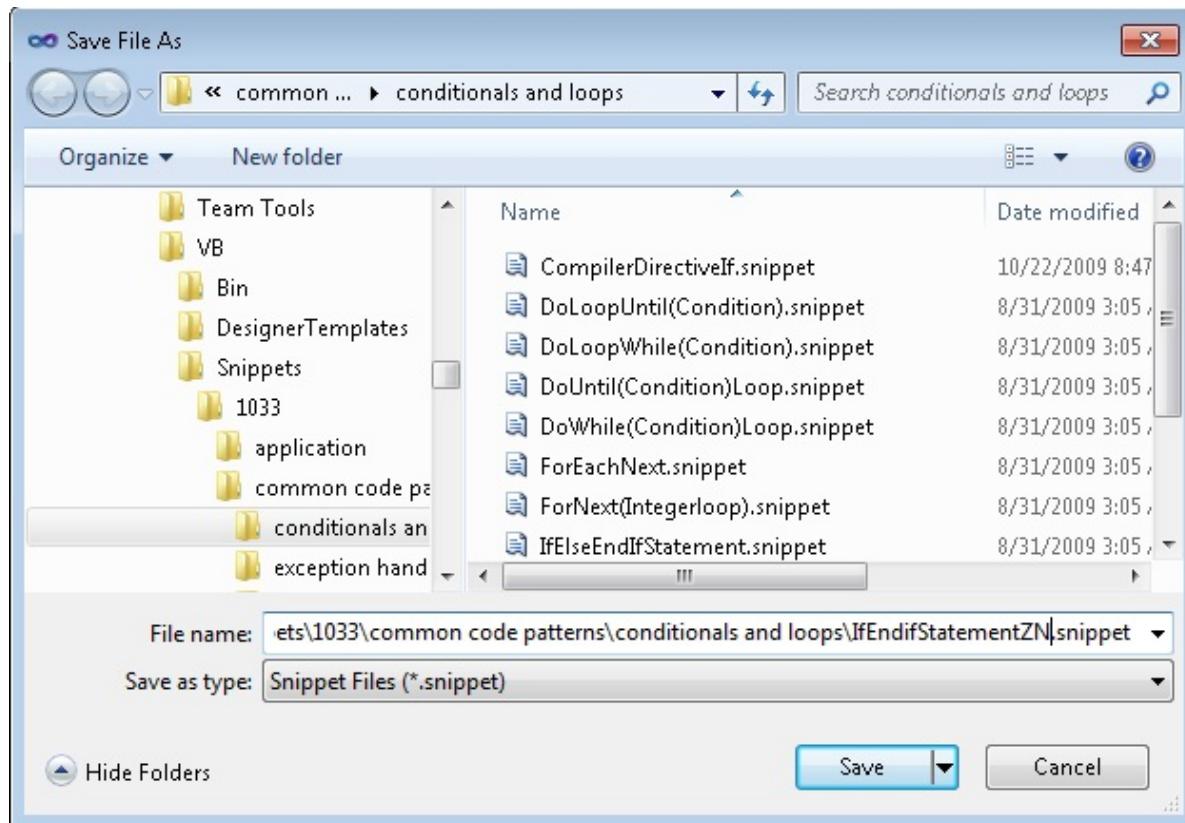
Now, go to File | Open | File, paste in the file path, and click Open.



This next part is really just to make sure we don't modify the original snippet. As you gain more experience doing modifications, you might choose to change the original, but for now, you want to make sure you can keep your existing set of snippets intact. Go to **File | Save IfEndifStatement.snippet As:**



Now just save the file with the original file name and your initials appended to the end.



You are ready to modify the snippet. Start with the header information:

```
<Header>
  <Title>If..End If Statement</Title>
  <Author>Microsoft Corporation</Author>
  <Description>Inserts an If..End If statement.</Description>
  <Shortcut>If</Shortcut>
</Header>
```

For now, just change the Title and Shortcut to include your initials:

```
<Title>If..End If Statement (ZN)</Title>

<Shortcut>Ifzn</Shortcut>
```

#### NOTE

The key here is the shortcut name. It is the name you will use most often to invoke this snippet so make it something that makes sense.

Because you are here to change the default value as well, locate the Default tag:

```
<Declarations>
  <Literal>
    <ID>Condition</ID>
    <Type>Boolean</Type>
    <ToolTip>Replace with an
    <Default>True</Default>
  </Literal>
</Declarations>
```

Change “True” to “False,” and then save the file and close it.

You are ready to use your new snippet. Go into your code, type in your new snippet shortcut, and press Tab (once or twice, both work the same).

```
Sub Main()
```

```
  ifzn|
```

```
End Sub
```

And you get the following result:

```
Sub Main()
```

```
  If False| Then
```

```
End If
```

```
End Sub
```

You now have a new snippet you can use. You can verify this by going back into the Code Snippets Manager (Ctrl+K, Ctrl+B) and finding your snippet in the list.

## 06.55 Understanding the Navigation Stack

<b>DEFAULT</b>	Ctrl+Shift+8 (back); Ctrl+Shift+7 (forward)
<b>VISUAL BASIC 6</b>	[no shortcut] (back); Ctrl+Shift+7 (forward)
<b>VISUAL C# 2005</b>	Ctrl+Shift+8 (back); Ctrl+Shift+7 (forward)
<b>VISUAL C++ 2</b>	Ctrl+Shift+8 (back); Ctrl+Num * (back); Ctrl+Shift+7 (forward)
<b>VISUAL C++ 6</b>	Ctrl+Num * (back); Ctrl+Shift+7 (forward)
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+8 (back); Ctrl+Shift+7 (forward)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	View.PopBrowseContext; View.ForwardBrowseContext
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C++, C#
<b>CODE</b>	vstipEdit0078

Did you know that a dedicated stack is available to you, just for going to definitions? Let's look at a couple of examples.

Suppose you are looking at a method call, as shown in the following illustration.

```
Re(3.145 * jLocal);
Doh();
```

You can press F12 to go to its definition:

```
public void Doh()
{
    Console.WriteLine("Hit da Doh!");
}
```

How do you go back? Just press Ctrl+Shift+8, and it takes you back:

```
Re(3.145 * jLocal);
Doh();
```

Pressing Ctrl+Shift+7 moves you to the definition again.

```
public void Doh()
{
    Console.WriteLine("Hit da Doh!");
}
```

So what's going on? Well, every time you go to a definition it keeps track of where you come from. This works every time you go to definition so that you can always find your way back. The following illustrations show another example.

```
Mi("A me so la ti do");
AddStuff(5, 3);
```

Press F12 (go to definition):

```
public int AddStuff(int a, int b)
{
    a += GetNumber();
    return a + b;
}
```

From here, you could press Ctrl+Shift+8 to go back. But what if you want to keep digging into definitions? In this example, let's say you've clicked in the GetNumber method:

```
public int AddStuff(int a, int b)
{
    a += GetNumber();
    return a + b;
}
```

Now press F12:

```
public int GetNumber()
{
    return 5;
}
```

Now, if you want to go back, press Ctrl+Shift+8:

```
public int AddStuff(int a, int b)
{
    a += GetNumber();
    return a + b;
}
```

Then press Ctrl+Shift+8 again:

```
Mi("A me so la ti do");
AddStuff(5, 3);
```

## 06.56 Navigate Backward and Navigate Forward Using Go Back Markers

<b>DEFAULT</b>	Ctrl+- (back); Ctrl+Shift+- (forward)
<b>VISUAL BASIC 6</b>	Ctrl+- (back); Ctrl+Shift+F2 (back); Ctrl+Shift+- (forward)
<b>VISUAL C# 2005</b>	Ctrl+- (back); Ctrl+Shift+- (forward)
<b>VISUAL C++ 2</b>	Ctrl+- (back); Ctrl+Shift+- (forward)
<b>VISUAL C++ 6</b>	Ctrl+- (back); Ctrl+Shift+- (forward)
<b>VISUAL STUDIO 6</b>	Ctrl+- (back); Ctrl+Shift+- (forward)
<b>WINDOWS</b>	Alt,V, B (back); Alt,V, F (forward)
<b>MENU</b>	View   Navigate Backward; View   Navigate Forward
<b>COMMAND</b>	View.NavigateBackward; View.NavigateForward
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0024

If you move in a single command more than several lines away from where you are currently working or if you edit in a particular location that is not adjacent to the last place you edited, the editor remembers locations. It does this by creating Go Back markers based on specific conditions. The goal is to remember interesting locations so that you can recall where you have been working without remembering so many locations that the feature is not useful (such as every character typed, or every line entering several new lines of code one right after the other).

A Go Back marker is dropped under the following conditions:

- An incremental search (including reverse) leaves a Go Back marker at the beginning of the search and another one at the end.
- A Go To Line action, like Ctrl+G, or a mouse-click that moves the cursor 11 lines or more from the current position drops a Go Back marker at the new location.
- A destructive action (like pressing backspace) after having moved the cursor to a new location drops a Go Back marker.

- Doing a search, like Ctrl+F, drops a Go Back marker at the found location.
- Opening a file drops a Go Back marker wherever the cursor was on the old file and drops another on the opened file.

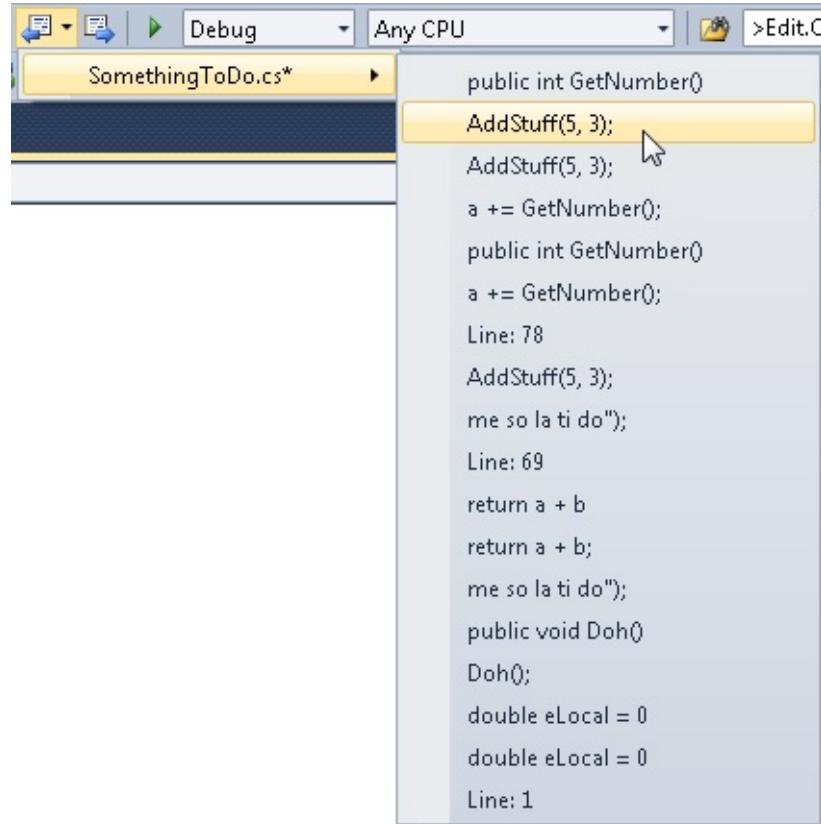
That now brings us to the navigation buttons on the Standard Toolbar (and keyboard shortcuts, too). These gems make travelling around your code much, much easier:



You can use the buttons to quickly navigate among the Go Back markers that have been dropped throughout Visual Studio. To quickly look at the list of places you can go, just click the drop-down arrow for the back button.

#### NOTE

The navigation buttons in the Standard Toolbar are on the upper-left of the image below.



## 06.57 Select from the Current Cursor Location to the Last Go Back Marker

<b>DEFAULT</b>	Ctrl+=
<b>VISUAL BASIC 6</b>	Ctrl+=
<b>VISUAL C# 2005</b>	Ctrl+=
<b>VISUAL C++ 2</b>	Ctrl+=
<b>VISUAL C++ 6</b>	Ctrl+=
<b>VISUAL STUDIO 6</b>	Ctrl+=
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.SelectToLastGoBack
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0069

In vstipEdit0024 ([06.56 Navigate Backward and Navigate Forward Using Go Back Markers](#), on page 277), we defined Go Back markers. In this tip, we look at how they can be useful for selecting text. Let's say you are writing some code and want to quickly select a chunk of text. In the following illustration, the cursor is at your starting location (line 15).

```
15 Console.WriteLine("Blah");|  
16 Console.WriteLine("Blah");  
17 Console.WriteLine("Blah");  
18 Console.WriteLine("Blah");  
19 Console.WriteLine("Blah");  
20 Console.WriteLine("Blah");  
21 Console.WriteLine("Blah");  
22 Console.WriteLine("Blah");  
23 Console.WriteLine("Blah");  
24 Console.WriteLine("Blah");  
25 Console.WriteLine("Blah");  
26 Console.WriteLine("Blah");  
27 Console.WriteLine("Blah");  
28 Console.WriteLine("Blah");  
29 Console.WriteLine("Blah");  
30 Console.WriteLine("Blah");  
31
```

You click on a new location (line 27), which results in a Go Back marker (which you can't see) being placed at the starting location (line 15).

```
15 Console.WriteLine("Blah");  
16 Console.WriteLine("Blah");  
17 Console.WriteLine("Blah");  
18 Console.WriteLine("Blah");  
19 Console.WriteLine("Blah");  
20 Console.WriteLine("Blah");  
21 Console.WriteLine("Blah");  
22 Console.WriteLine("Blah");  
23 Console.WriteLine("Blah");  
24 Console.WriteLine("Blah");  
25 Console.WriteLine("Blah");  
26 Console.WriteLine("Blah");  
27 Console.WriteLine("Blah");|  
28 Console.WriteLine("Blah");  
29 Console.WriteLine("Blah");  
30 Console.WriteLine("Blah");  
31
```

Now we have the Go Back marker at the beginning of line 15 and the current

cursor location at the end of line 27. In this case, you had to jump at least 11 lines to get your Go Back marker. To select from the current cursor location to the last Go Back marker, just press **Ctrl+=** and watch the magic happen.

```
15 Console.WriteLine("Blah");
16 Console.WriteLine("Blah");
17 Console.WriteLine("Blah");
18 Console.WriteLine("Blah");
19 Console.WriteLine("Blah");
20 Console.WriteLine("Blah");
21 Console.WriteLine("Blah");
22 Console.WriteLine("Blah");
23 Console.WriteLine("Blah");
24 Console.WriteLine("Blah");
25 Console.WriteLine("Blah");
26 Console.WriteLine("Blah");
27 Console.WriteLine("Blah");|
28 Console.WriteLine("Blah");
29 Console.WriteLine("Blah");
30 Console.WriteLine("Blah");
31
```

## 06.58 Track Changes in the Editor

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Text Editor   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0048

To use this feature, you need to go to Tools | Options | Text Editor | General and have Track Changes and Selection Margin checked.

Ever wonder how the colored lines to the left of your code (by the line numbers) actually work?

Let's begin with a clean slate.

```
11 static void Main(string[] args)
12 {
13
14
15
16
17 }
```

Let's add a couple of lines of code, and now we see two vertical yellow (you'll have to trust me on the colors) lines to the left of lines 14 and 15:

```
11 static void Main(string[] args)
12 {
13
14 Console.WriteLine("blah");
15 Console.WriteLine("blah");
16
17 }
```

All new code will have a yellow line to show you what part of the document is unsaved. If we save the code, the yellow lines turn green to indicate code that has been saved.

The saved indicator remains as long as you have the file open. When you close

and reopen the file, the line goes away:

```
11 static void Main(string[] args)
12 {
13
14     Console.WriteLine("blah");
15     Console.WriteLine("blah");
16
17 }
```

In Visual Studio 2010, an orange indicator shows a change that is different from the saved version. This was added for the scenario where a user does an undo after saving the changes.

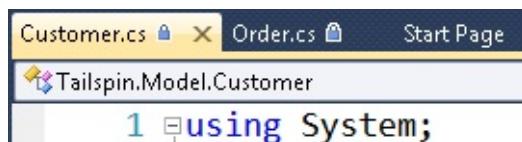
You can use the following grid to help keep the colors straight.

MARKER	DIFFERENT FROM FILE SAVED ON DISK?	DIFFERENT FROM FILE THAT WAS OPENED?
Nothing	No	No
Yellow	Yes	Yes
Green	No	Yes
Orange	Yes	No

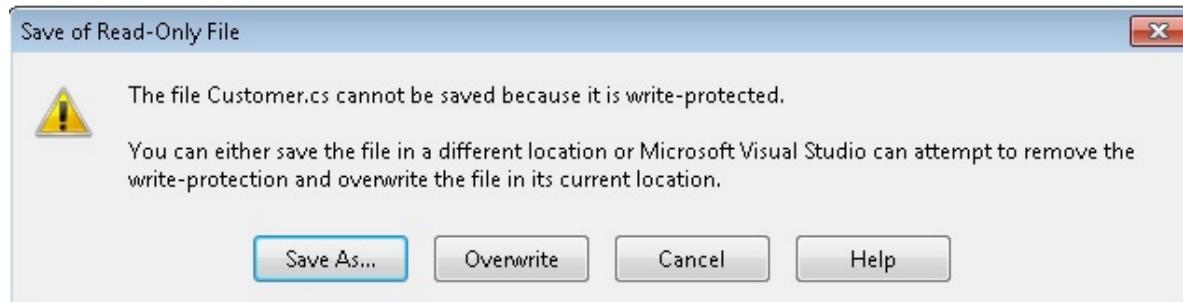
## 06.59 Edit Read-Only Files

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Documents
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEdit0074

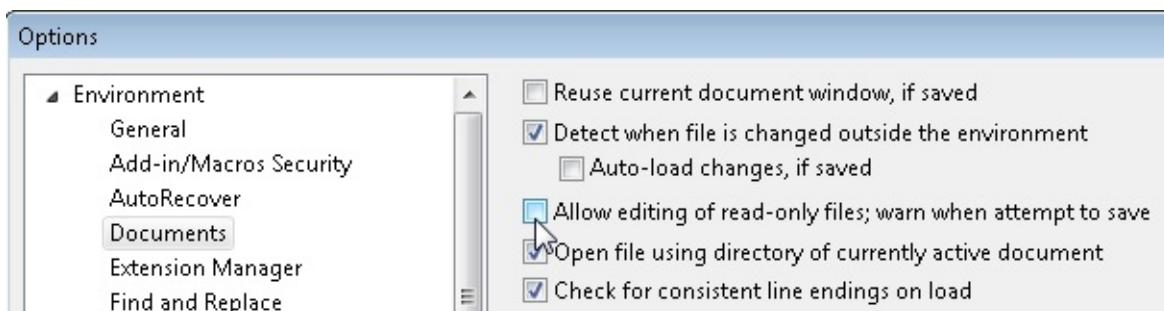
When you open a read-only document, you get an indicator that looks like a tiny lock on the file tab:



If you make changes and try to save them, you are (by default) met with the Save Of Read-Only File dialog box.



At this point, you can save your changes as another copy of the file, overwrite the existing file, or cancel. If you don't like these options, you can turn this feature off by going to Tools | Options | Environment | Documents and clear the Allow Editing Of Read-Only Files; Warn When Attempt To Save check box.



Now if you attempt to make any changes to a read-only file, you see the Edit of

Read-Only File dialog box.



## Edit In-Memory

Allows you to make edits and then display the Save Of Read-Only File dialog box when you save changes.

## Make Writable

If possible, turns off the read-only attribute of the file so that it can be edited.

## 06.60 Choosing CSS Versions

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options Text Editor   HTML   Validation (HTML Schema)
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010, 2010 SP1
<b>CODE</b>	vstipEdit0093

When working with Cascading Style Sheets (CSS), you often find yourself working with specific versions. In this tip, we look at how to set your CSS version for dedicated and embedded styles.

### NOTE

Visual Studio 2010 Service Pack 1 added limited support for HTML5 and CSS3. For more information, see <http://blogs.msdn.com/b/zainnab/archive/2011/04/12/vs2010-sp1-new-features-html-5-and-css-3-support.aspx>.

## Dedicated Style Sheets

When you create a CSS file in Visual Studio, you can choose the version you want by selecting it from the Cascading Style Sheet Version For Validation drop-down list on the Style Sheet Toolbar:



### NOTE

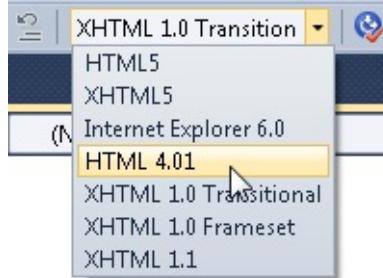
CSS3 doesn't show up in this list even after you install Visual Studio 2010 Service Pack 1.

## Embedded Styles

It is recommended that you use dedicated style sheets; however, you might find that you want to embed styles in your HTML source.

When you do this, it might not be clear how you select the CSS version. Most

people think it is the same as the version chosen for dedicated CSS files. This is not the case. It's bound to the choice you make in the Target Schema For Validation drop-down list located on the HTML Source Editing Toolbar or at Tools | Options | Text Editor | HTML | Validation (HTML Schema).



The following table shows how the CSS version relates to the choices.

HTML SCHEMA	CSS SCHEMA
Internet Explorer 6.0	Internet Explorer 6.0
HTML 4.01	CSS 1.0
XHTML 1.0 Transitional	CSS 2.0
XHTML 1.0 Frameset	CSS 2.0
XHTML 1.1	CSS 2.1
HTML5 (limited)	CSS 3.0 (limited)
XHTML5 (limited)	CSS 3.0 (limited)

## Finally

You have many good reasons to use dedicated style sheets, but if you do find yourself doing embedded CSS, make sure you understand which schema you are using based on your HTML schema choices.

## 06.61 Understanding Tag Specific Options

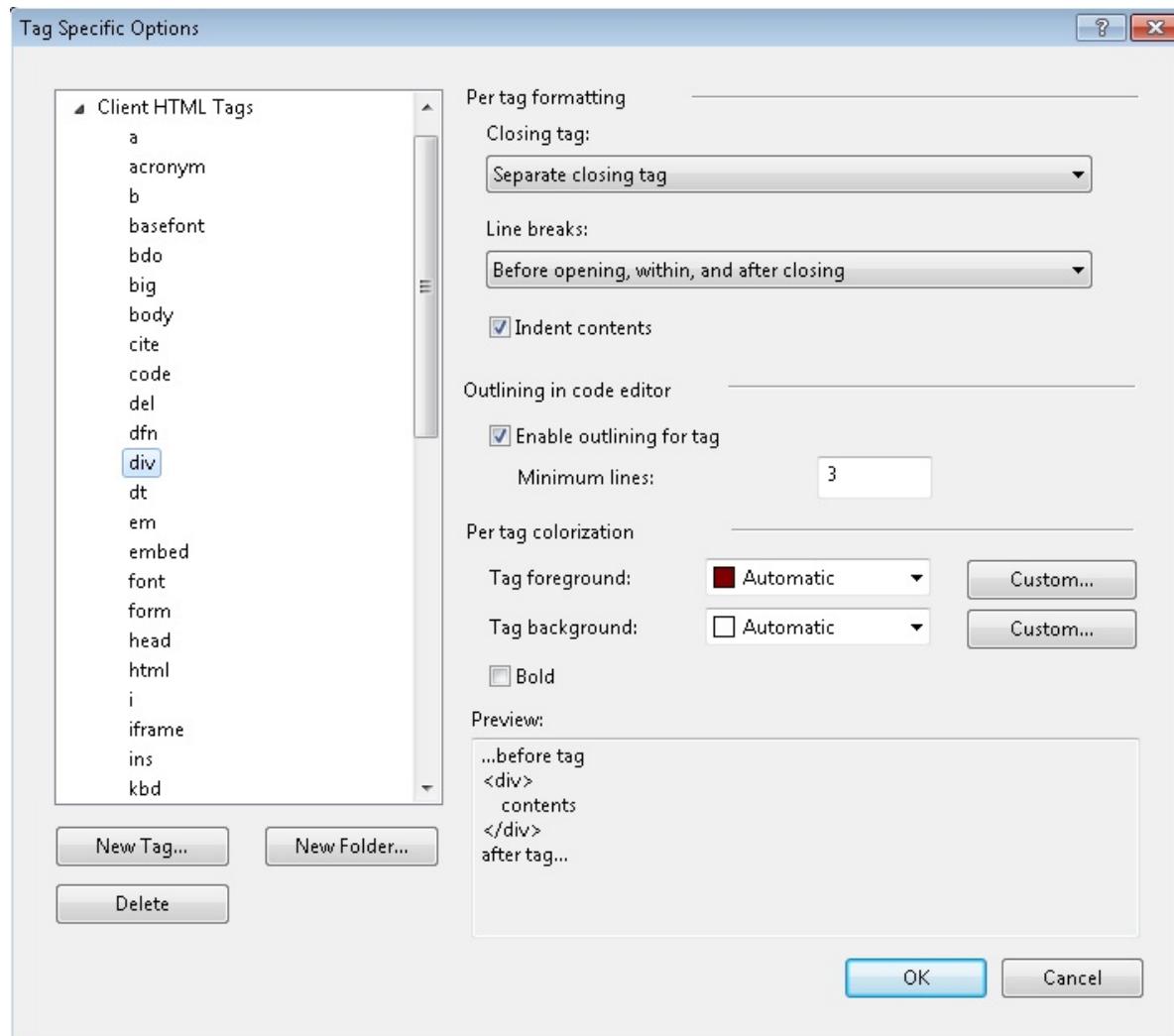
<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   HTML   Formatting
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0090

In vstipEdit0089 ([06.47 Format the Current Document or Selection \(Web\)](#), on page 265), I showed you how to use selection formatting on a sample.

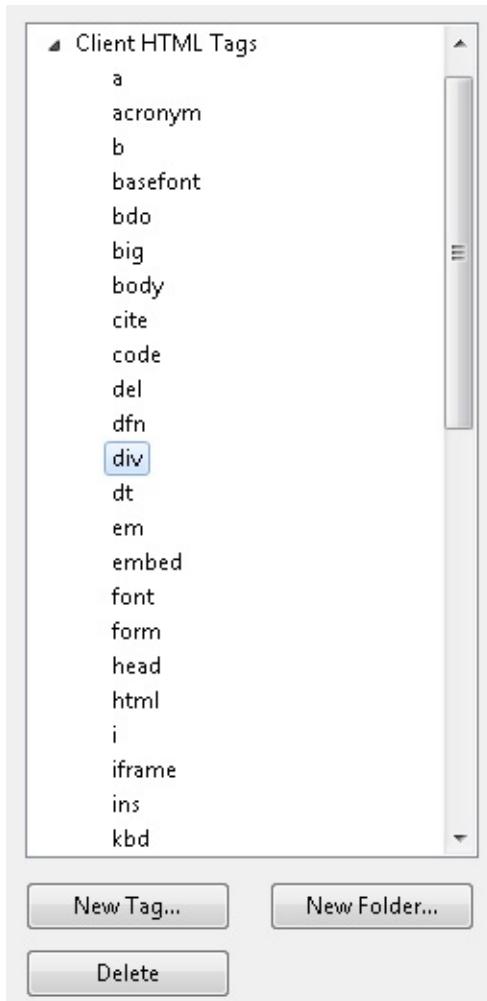
Let's say you don't like the way <DIV> tags are formatted. You can go to Tools | Options | Text Editor | HTML | Formatting and click the Tag Specific Options button.

### Exploring the Tag Specific Options Dialog Box

This dialog box is somewhat complicated, so let's go though it one area at a time. For reference, the following illustration provides a bird's eye view to help us stay oriented as we go along.



## Tree view



Allows you to select either the individual tag to format or a class of tags. By default, the tree contains the following nodes:

### **Default Settings**

Expand this node to set default formatting options for a complete class of tags, such as all server tags that support contents.

### **Client HTML Tags**

Expand this node to set custom formatting options for HTML elements.

### **ASP.NET Controls**

Expand this node to set custom formatting options for ASP.NET server controls.

### **New Tag**

Use to define new tags that aren't already listed.

### **New Folder**

Create a new node on the tree for organizing custom tags.

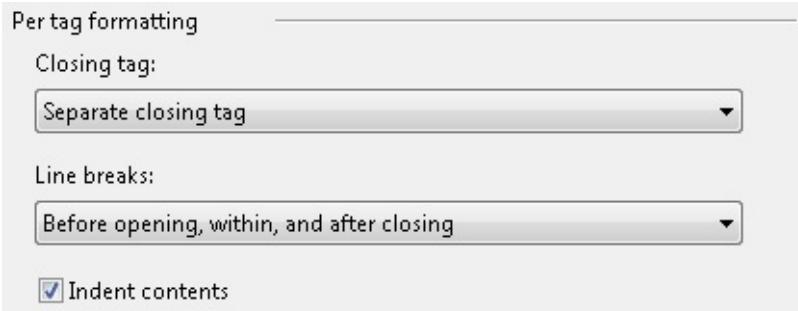
## Delete

Delete a tag or folder.

### NOTE

In the following examples, our working reference is the <DIV> tag from the Client HTML Tags section of the Tag Specific Actions dialog box.

## Per tag formatting



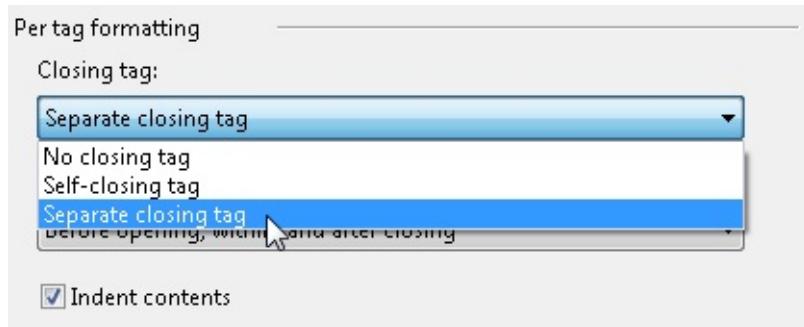
Per tag formatting works on a model with default settings for each tag. These settings provide the base rules by which the tag is formatted. The four categories of tags are as follows:

- Client tags that do not support contents, such as <BR />
- Client tags that support contents, such as <TABLE> or <H1>
- Server tags that do not support contents, such as asp:CheckBox
- Server tags that support contents, such as asp:Repeater

For each tag, the values from the appropriate category in the preceding list are used, unless an override is specified for that tag. By default, some overrides are supplied based on the common usage of the tags.

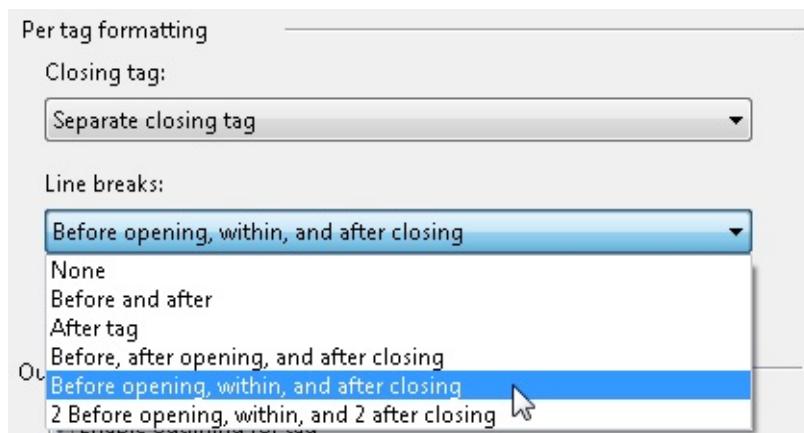
### Closing tag

Indicates how the closing tag should look when automatically created.



## Line breaks

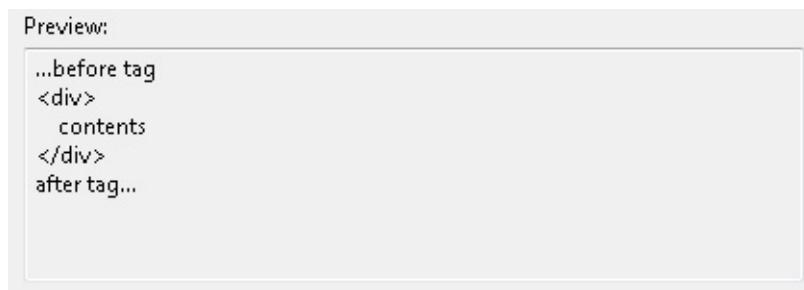
Directs the formatting to place line breaks as specific points in the entire element.



## Indent contents

Indents the contents for the element, if appropriate.

## Preview



The absolute *best* way to see how the per-tag formatting is going to look is to use the Preview area (bottom of the dialog box). It updates based on your choices so that you can make informed decisions on the formatting. Unfortunately it does not show color choices (described later).

## Outlining in code editor



## Enable outlining for tag

When outlining is enabled, the editor monitors the number of lines in the element and then applies outlining to the tag when the element exceeds the specified line threshold. (The editor does not automatically collapse the tag.) This feature is useful for collapsing long elements, such as large tables.

### NOTE

If outlining doesn't show up as expected try closing and re-opening the file to make it appear.

## Minimum lines

Number of lines before outlining is enabled for the element.

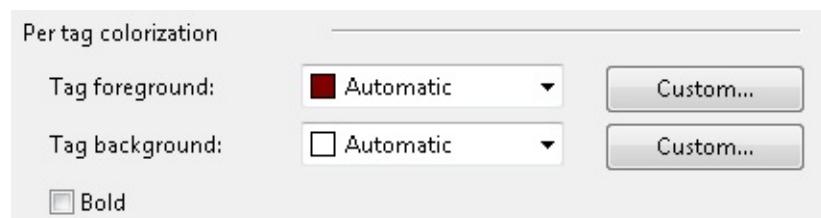
Example: The <DIV> has outlining enabled and minimum lines set to three. For three lines or more in an element, outlining, the minus sign to the left of the opening tag, is enabled for that element:

```
<div>
    something cool is here
</div>
```

Notice the outlining (the minus sign to the left of the opening <DIV> tag, shown in the preceding illustration) that has kicked in because we have the minimum number of lines. In this example, if the minimum lines are changed to five, outlining is not enabled and no minus sign appears:

```
<div>
    something cool is here
</div>
```

## Per tag colorization



### **Tag foreground**

The color of the text for the tags.

### **Tag background**

The color of the text background.

### **Bold**

Makes the tags bold.

## **Finally**

You can see that there is much to learn about tag-specific options. Take your time as you explore the possibilities in the Tag Specific Options dialog box.

# Chapter 7. Debugging

*"It has been just so in all my inventions. The first step is an intuition—and comes with a burst, then difficulties arise. This thing that gives out and then that—'Bugs' as such little faults and difficulties are called show themselves and months of anxious watching, study and labor are requisite before commercial success—or failure—is certainly reached."*

— Thomas Alva Edison *Letter to Theodore Puskas (18 Nov 1878)*

As a developer, you spend a great deal of time debugging. Visual Studio has long provided great tools for helping find errors in your code. This chapter focuses on showing how to take full advantage of common items like the Locals and Autos windows as well as exploring more advanced techniques such as setting tracepoints in the Call Stack window.

Additionally, lots of great new features in Visual Studio 2010 can help with your troubleshooting efforts. Major changes have been made to the Breakpoints window, and new DataTips are available to provide information when and where you need it. There is no shortage of great techniques to cut down the time you spend finding problems in your code.

## 07.01 Setting a Breakpoint with Code

VERSIONS	2005, 2008, 2010
CODE	vstipDebug0036

Sometimes you want to have clear breakpoints in your code that travel with the source. You can do this quite easily.

### Compiler Directive

In C# and VB, you need to set a compiler option that hits the breakpoint only when debugging. If you don't, your release code continues to hit the code-based breakpoint, which is generally considered a bad thing. To do this, you set the #If DEBUG compiler option, as you can see in each of the following code samples.

#### C#

In C#, you set a breakpoint by using the System.Diagnostics.Debugger.Break method:

```
#if DEBUG
    System.Diagnostics.Debugger.Break();
#endif
```

#### VB

In VB, you set a breakpoint by using the Stop command:

```
#If DEBUG Then
    Stop
#End If
```

Now you can use code to set breakpoints. Just as an aside, breakpoints set in this way do not show up in the Breakpoints window.

#### C++

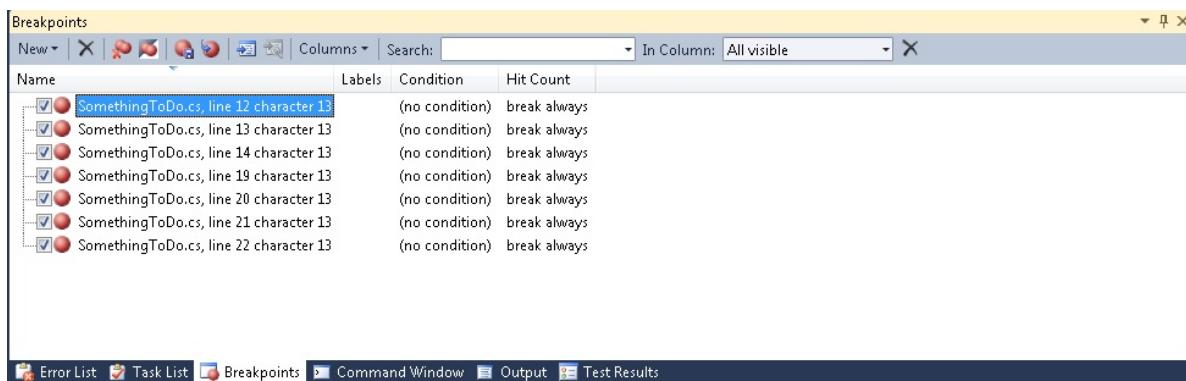
In C++, it's a very similar situation. Based on the type of C++ project you are creating, these commands can change slightly; but with native C++, you can use the \_\_debugbreak command with the #if \_DEBUG compiler option:

```
#if _DEBUG
    __debugbreak;
#endif
```

## 07.02 Using Ctrl+Alt+B to Open the Breakpoints Window

<b>DEFAULT</b>	Ctrl+Alt+B
<b>VISUAL BASIC 6</b>	Ctrl+Alt+B
<b>VISUAL C# 2005</b>	Ctrl+Alt+B; Ctrl+D, Ctrl+B; Ctrl+D, B
<b>VISUAL C++ 2</b>	Ctrl+Alt+B; Ctrl+B
<b>VISUAL C++ 6</b>	Ctrl+Alt+B; Alt+F9
<b>VISUAL STUDIO 6</b>	Ctrl+B
<b>WINDOWS</b>	Alt, D, W, B
<b>MENU</b>	Debug   Windows   Breakpoints
<b>COMMAND</b>	Debug.Breakpoints
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0011

Use Ctrl+Alt+B to open the Breakpoints window or, if it is already open, to give it the focus.

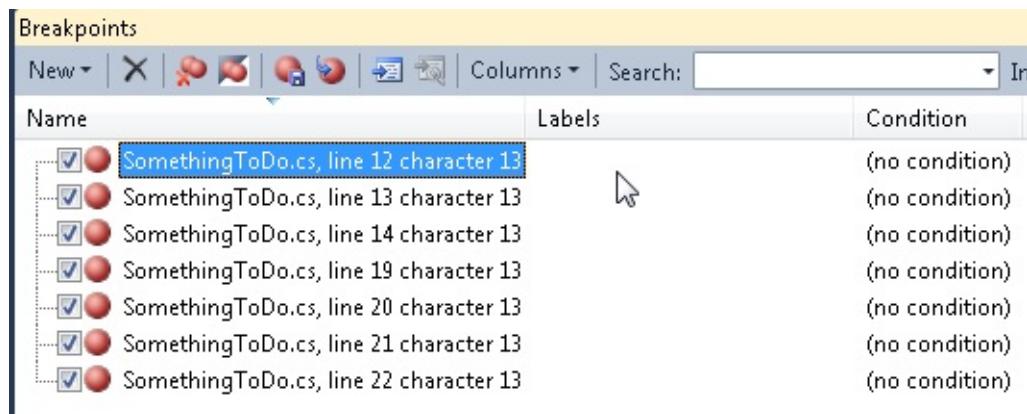


## 07.03 Adding Labels to Breakpoints

<b>DEFAULT</b>	Alt+F9, L
<b>VISUAL BASIC 6</b>	Alt+F9, L
<b>VISUAL C# 2005</b>	Alt+F9, L
<b>VISUAL C++ 2</b>	Alt+F9, L
<b>VISUAL C++ 6</b>	[no shortcut]
<b>VISUAL STUDIO 6</b>	Alt+F9, L
<b>WINDOWS</b>	Shift+F10, A (inside Breakpoints Window)
<b>MENU</b>	[Context Menu]   Edit Labels
<b>COMMAND</b>	EditorContextMenus.CodeWindow.Breakpoint.BreakpointEditlabels
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipDebug0001

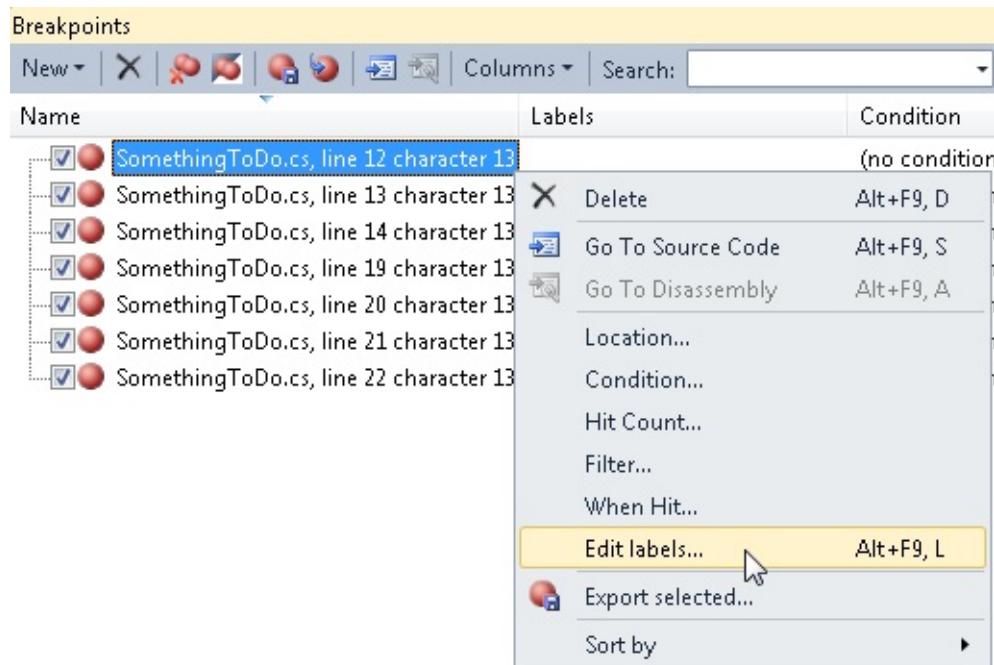
Did you know that breakpoints in Visual Studio 2010 support labels? This tip explains how you can use them. How do labels help you? Well, first, you can now have friendly names for breakpoints to make them easier to understand. Second, you can sort by the label names. And, third, you can search the labels from the Breakpoints window. (See vstipDebug0002, [07.19 Searching Breakpoints](#), on page 312.)

After setting one or more breakpoints in your code, open the Breakpoint window (Ctrl+Alt+B). Notice the new Labels column:



Right-click one or more selected breakpoints to bring up the context menu, and

choose Edit Labels (Alt+F9, L):



You get the following dialog box:



You can type in one (or more) labels for the breakpoint and/or select one of the labels previously used, and then click OK. You should see now your label(s):

Breakpoints			
Name	Labels	Condition	
[ <input checked="" type="checkbox"/> ] SomethingToDo.cs, line 12 character 13	fun with labels,some cool label here	(no condition)	
[ <input checked="" type="checkbox"/> ] SomethingToDo.cs, line 13 character 13		(no condition)	
[ <input checked="" type="checkbox"/> ] SomethingToDo.cs, line 14 character 13		(no condition)	

## 07.04 Enable or Disable All Breakpoints

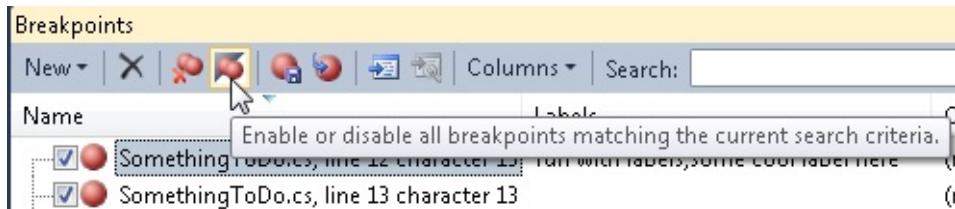
<b>WINDOWS</b>	Alt,D, N
<b>MENU</b>	Debug   Disable All Breakpoints; Debug   Enable All Breakpoints
<b>COMMAND</b>	Debug.DisableAllBreakpoints; Debug.EnableAllBreakpoints
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0018

There are times when you will want to quickly and temporarily turn off all or some of your breakpoints. This tip covers how to disable and enable your breakpoints.

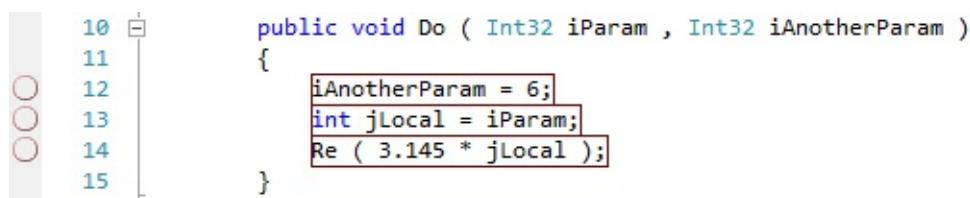
You can go to Debug | Disable All Breakpoints on the menu bar, or in the Breakpoints window, you can click the Enable Or Disable All Breakpoints button:

### NOTE

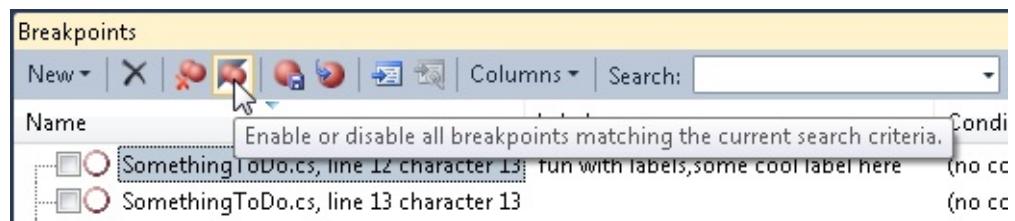
In Visual Studio 2010, only the breakpoints currently visible in the window are disabled when you use this option, so it is very useful for enabling or disabling a subset of your breakpoints when used with the Search feature.



The result, in either case, is the same. One or more breakpoints in the current project are disabled:



To enable them again, just go to Debug | Enable All Breakpoints on the menu bar, or click the Enable Or Disable All Breakpoints button in the Breakpoints window again:



## 07.05 TODO Comments in the Task List

VERSIONS	2005, 2008, 2010
CODE	vstipTool0029

Have you ever written some code and want to leave a reminder to yourself to do something? Did you know about the “To Do” comment feature? It is a great feature, and because the comment goes directly in the source, everyone can have access to the information when you check in code.

So here’s how it works.

### VB

In VB, you just put any comment in that begins with “todo” (case doesn’t matter):

```
'TODO fix this connection string
```

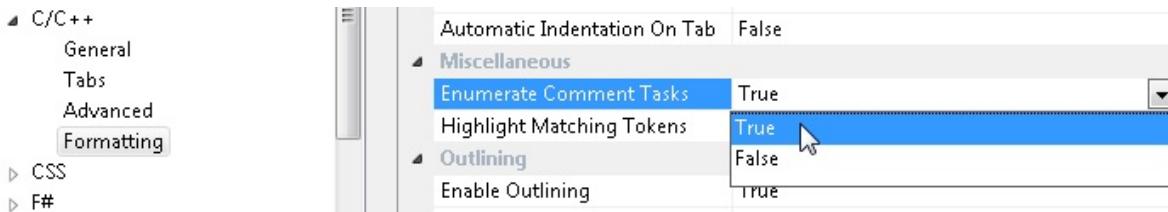
### C#

In C#, it’s the same thing (again, case doesn’t matter):

```
//todo fix this connection string
```

### C++

The C++ version looks just like C#, but you have to explicitly turn this feature on, and the “TODO” must be all uppercase. Go to Tools | Options | Text Editor | C/C++ | Formatting | Miscellaneous, and change Enumerate Comment Tasks to True:



Whichever language you use, the result is a nice entry in your Task List dialog box:

Task List - 1 task			
Comments			
!	Description	File	Line
	TODO fix this connection string	C++ Clean Console.cpp	9

**NOTE**

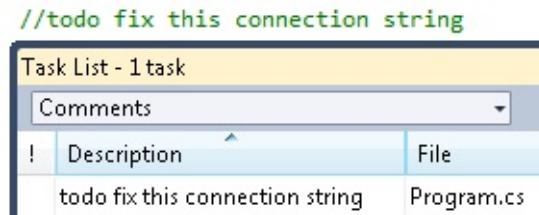
To see these items, you have to click the drop-down list in the Task List dialog box and choose Comments, as shown in the preceding illustration.

Like all Task List items, comments are Solution-wide in scope, so you will see all the shortcuts for the entire solution in the Task List window.

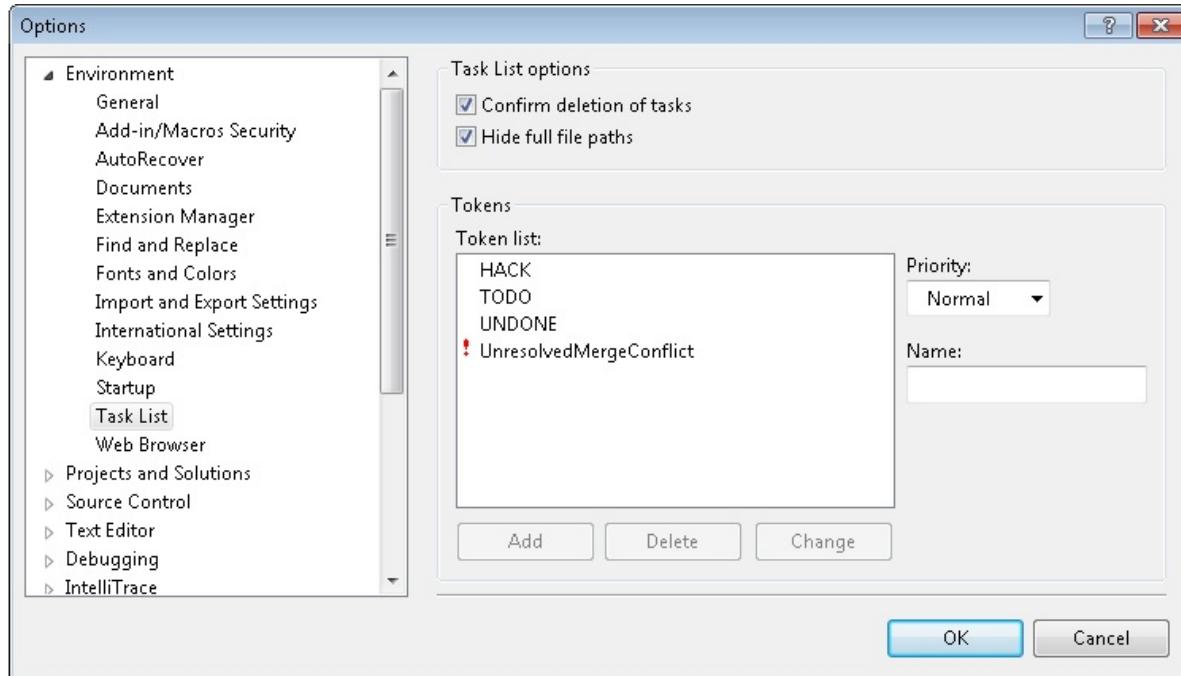
## 07.06 Create Custom Tokens for the Task List

WINDOWS	Alt,T, O
MENU	Tools   Options
COMMAND	Tools.Options
VERSIONS	2005, 2008, 2010
CODE	vstipTool0032

In vstipTool0029 ([07.05 TODO Comments in the Task List](#), page 296), I showed you how to create comments that show up in the Task List:



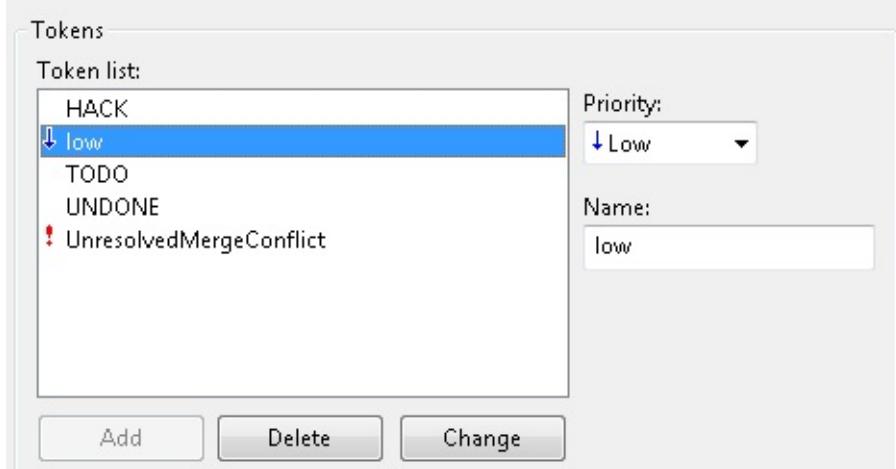
The “TODO” part is known as a token. It’s a trigger to indicate that an item should be put in the Task List. Did you know you can create your own custom tokens? Go to Tools | Options | Environment | Task List | Tokens to see the following window:



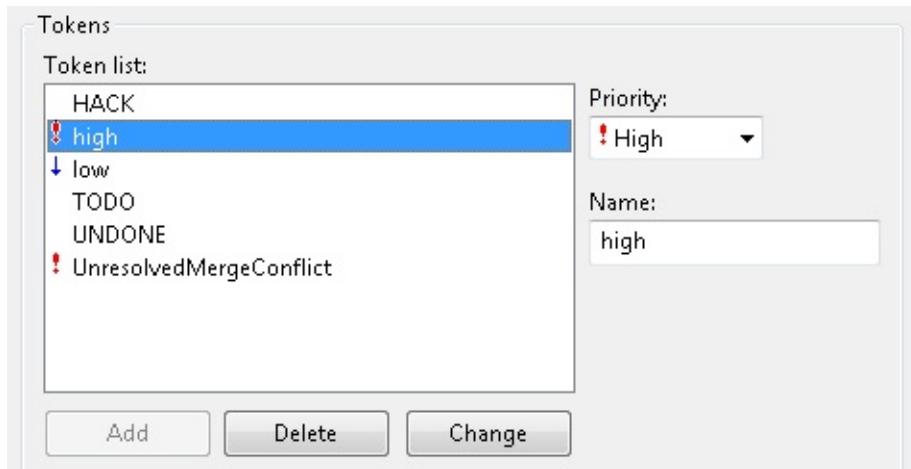
As shown in the preceding illustration, notice the following entries in this area:

HACK, TODO, UNDONE, and UnresolvedMergeConflict. By the way, UnresolvedMergeConflict is not an error; it is an actual token that you can use.

For now, let's create a couple of our own. In the Name text box, type in the word **low**, set the priority to Low, and then click Add. You should see the following:



Now do the same thing again, this time using the word **high** and setting the priority to High:



You now have a couple of custom tokens. Add two comments to your existing code, and use the new "low" and "high" tokens:

```
//todo fix this connection string  
//low tell Joe to fix this part  
//high tell Mary to redo this section
```

You now see the following in the Task List:

Task List - 3 tasks		
Comments		
	Description	File
!	high tell Mary to redo this section	Program.cs
!	todo fix this connection string	Program.cs
↓	low tell Joe to fix this part	Program.cs

## Sharing Tokens

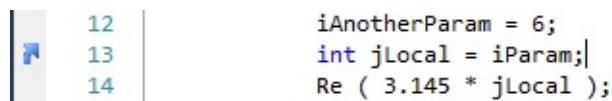
You can create as many tokens as you want to suit your needs, so feel free to experiment with these. Unfortunately, the tokens aren't shared unless you export them and then someone else imports them. Go to Tools | Import And Export Settings, and export the Task List Options found under All Settings, Options, Environment:



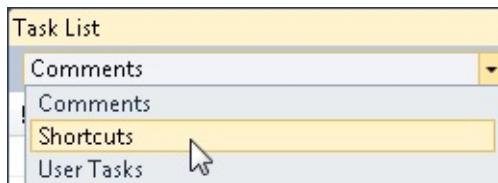
## 07.07 Create Code Shortcuts in the Task List

<b>DEFAULT</b>	Ctrl+K, Ctrl+H
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+H
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+H; Ctrl+E, Ctrl+T; Ctrl+E, T
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+H
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+H
<b>WINDOWS</b>	Alt,E, K, H
<b>MENU</b>	Edit   Bookmarks   Add Task List Shortcut
<b>COMMAND</b>	Command Edit.ToggleTaskListShortcut
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0030

In vstipTool0029 ([07.05 TODO Comments in the Task List](#), page 296), I showed you how to create comments that show up in your Task List window. However, sometimes all you want is a shortcut to a line of code that you visit often. You can create shortcuts to any line of code. Just place the cursor on the line, and press Ctrl+K, Ctrl+H (toggles the shortcut on or off). This creates the shortcut glyph in the margin:



To see all your shortcuts, go to the Task List dialog (Ctrl+\, T), and choose Shortcuts from the drop-down list:



Now you should see all the shortcuts you created:

Task List - 3 tasks	
Shortcuts	
	Description
!	<input checked="" type="checkbox"/> int jLocal = iParam;
!	<input checked="" type="checkbox"/> double eLocal = 0.002;
!	<input checked="" type="checkbox"/> eLocal = eLocal * d;

You can treat them like any other task, and you can set priority levels as well as mark them complete:

Task List - 3 tasks	
Shortcuts	
	Description
!	<input checked="" type="checkbox"/> int jLocal = iParam;
!	<input checked="" type="checkbox"/> double eLocal = 0.002;
↓	<input checked="" type="checkbox"/> eLocal = eLocal * d;

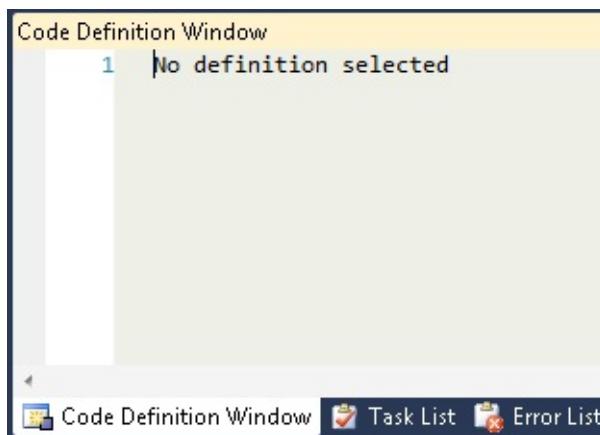
You can double-click any item to go to the line of code referenced in the shortcut, right-click, and then choose Delete to remove it from the list.

Like all Task List items, these are Solution-wide in scope, so you will see all the shortcuts for the entire solution in the Task List window.

## 07.08 Code Definition Window

<b>DEFAULT</b>	Ctrl+\, Ctrl+D; Ctrl+\, D
<b>VISUAL BASIC 6</b>	Ctrl+\, Ctrl+D; Ctrl+\, D
<b>VISUAL C# 2005</b>	Ctrl+\, Ctrl+D; Ctrl+\, D; Ctrl+W, Ctrl+D; Ctrl+W, D
<b>VISUAL C++ 2</b>	Ctrl+\, Ctrl+D; Ctrl+\, D
<b>VISUAL C++ 6</b>	Ctrl+\, Ctrl+D; Ctrl+\, D; Ctrl+Shift+V
<b>VISUAL STUDIO 6</b>	Ctrl+\, Ctrl+D; Ctrl+\, D
<b>WINDOWS</b>	Alt,V, D
<b>MENU</b>	View   Code Definition Window
<b>COMMAND</b>	View.CodeDefinitionWindow
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C++, C#
<b>CODE</b>	vstipTool0012

Ever want to just click on a reference and see the definition as you go without having to use Go To Definition (F12)? For quite some time, the Code Definition window has been available to use just for this purpose. Go to View | Code Definition Window, and you see the following:



Now click on any symbol you would like to get a definition for, and you should get something like this:

The screenshot shows the 'Code Definition Window' for the 'Fa()' method in the file 'SomethingToDo.cs'. The main code editor at the top displays:

```
56     public int Overload ( int i )
57     {
58         Fa();
59         return ( i );
```

The status bar at the bottom left indicates '100 %'. Below the main editor is the title bar 'Code Definition Window - Fa() (SomethingToDo.cs)'. A secondary code editor window titled 'Code Definition Window - Fa() (SomethingToDo.cs)' is open, showing the definition of the 'Fa()' method:

```
31     public void Fa ( )
32     {
33         So ( );
34     }
35 }
```

The Code Definition window provides you with an instant, read-only view of your definition so that you don't have to look for it when you are looking at a symbol.

## 07.09 Save Changes Before Building

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Projects and Solutions   Build and Run
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0015

In Tools | Options | Projects and Solutions | Build And Run, notice the Before Building option:



The default setting is to save all changes to the solution file and to save all project files that changed since the last build, but other options are also available:

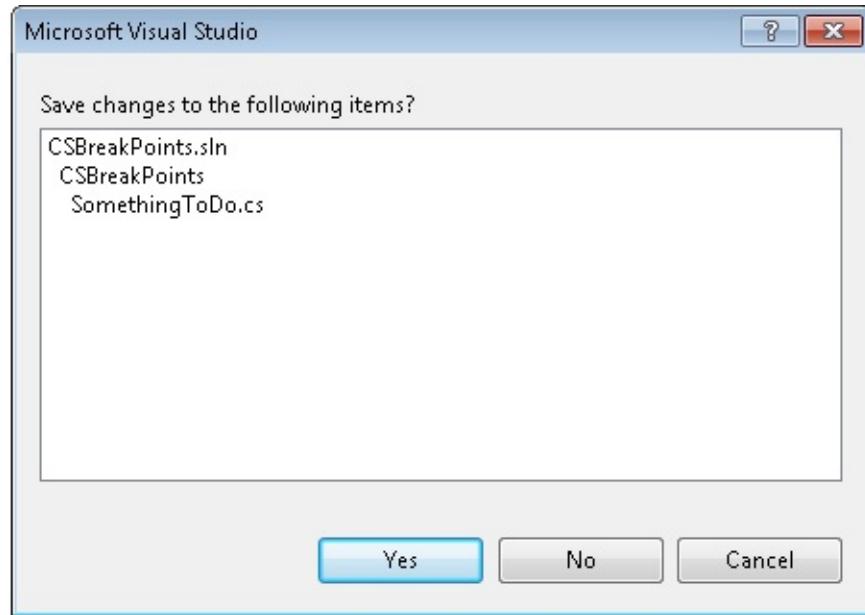


### Save Changes To Open Documents Only

This option does what it says and saves changes to all open documents without any prompt.

### Prompt To Save All Changes

This option prompts you with a choice about saving changes:



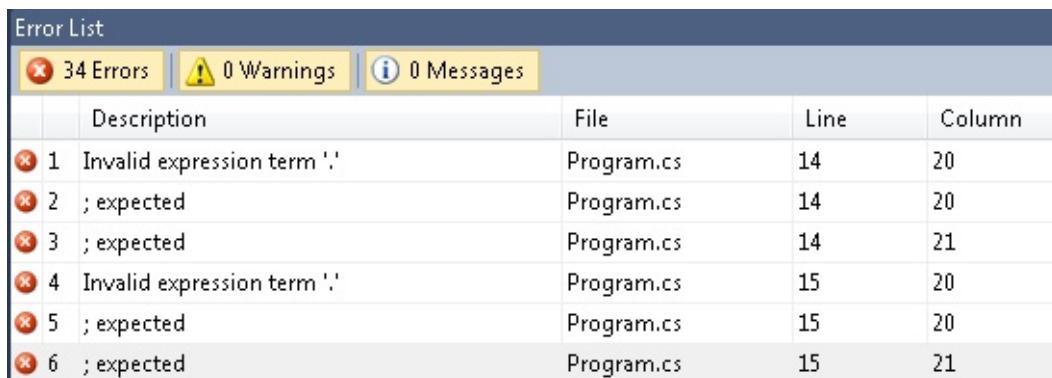
## Don't Save Any Changes

This runs the code but does not save any of the changes you have made. Usually this is not a good idea, but it might be useful in some situations where you are testing many different code scenarios quickly.

## 07.10 Navigate Errors in the Error List

<b>DEFAULT</b>	F8 (next); Shift+F8 (previous)
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	F8 (next); Shift+F8 (previous)
<b>VISUAL C++ 2</b>	F4 (next); Shift+F4 (previous)
<b>VISUAL C++ 6</b>	F8 (next); F4 (next)
<b>VISUAL STUDIO 6</b>	F8 (next); F12 (next); Shift+F8 (previous); Shift+F12 (previous)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.GoToNextLocation; Edit.GoToPrevLocation
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0019

When checking out errors in the Errors window, you can easily navigate to the next error by pressing F8 or to the previous error by pressing Shift+F8. These actions rotate through all the errors in the direction of choice:



Additionally, as you cycle through, the location of each error in your code is selected:

The screenshot shows a code editor with a vertical green bar on the left indicating the current line of code. The code consists of five lines of C# code, each containing a call to `Console.WriteLine`. The fifth line, which contains the error, has its entire line underlined in red, indicating it is the currently selected error in the list.

```
13 Console.WriteLine
14 Console.WriteLine
15 Console.WriteLine
16 Console.WriteLine
17 Console.WriteLine
```

## 07.11 Ordering and Multicolumn Sorting in Tool Windows

VERSIONS	2005, 2008, 2010
CODE	vstipTool0021

When working with the various tool windows, you often need the ability to rearrange the information in different ways. With column ordering and column sorting, you have this ability.

### Column Ordering

This technique can be used in a variety of tool windows, most notably the Task List window. Let's begin with reordering the columns. In tool windows where this is supported, you can drag the columns around to put them in the order you want:

!	Description	File	Line	Column
	todo one	Program.cs	12	16
	todo two	Program.cs	17	15
	todo three	Program.cs	25	15

Additionally, you can sort by clicking on a column to make it sort ascending or descending, as shown in the following illustrations:

Line
12
17
25

Line
25
17
12

### Multicolumn Sorting

But the best part is that you can have multicolumn sorting. So you can sort first by one column:

Task List - 3 tasks			
Comments			
!	Description	Column	File
!	todo one	16	Program.cs

Then just press Shift, and click on the next column you would like to sort by:

Task List - 3 tasks			
Comments			
!	Description	Column	File
!	todo one	16	Program.cs

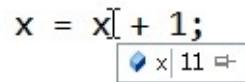
If you want, you can continue to hold Shift and sort by more columns:

Task List - 3 tasks			
Comments			
!	Description	Column	File
!	todo one	16	Program.cs

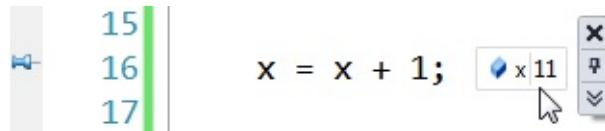
## 07.12 Pin a DataTip to Source Code

<b>COMMAND</b>	EditorContextMenus.CodeWindow.PinToSource
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipDebug0005

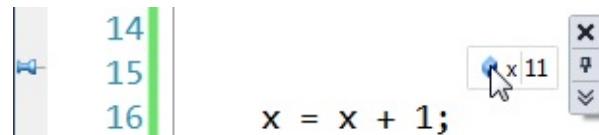
In Visual Studio 2010, you can now pin DataTips to source code. The purpose of pinned DataTips is to have the information stay with the line of code at all times, even after you scroll away from that line. To create one, just put your mouse pointer over any variable while debugging:



Notice the little pin at the end of the DataTip shown in the preceding illustration? If you click the pin, you have a pinned DataTip. The most obvious indicator of this is the pushpin that now shows up in the far left margin:



It is now, quite literally, pinned to a certain line of code. If you click the DataTip and drag it, you can change the line that the DataTip is pinned to:



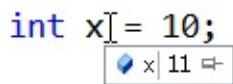
You can also drill down into objects and pin properties as well:



## 07.13 Create a Floating DataTip

VERSIONS	2010
CODE	vstipDebug0006

Floating DataTips are great for having information available where you want it. This tip shows how you use them. First, enter Break Mode, and pause your mouse pointer over a variable in the current scope; you should see something like the following:



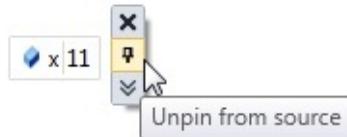
Click the pin to create a pinned DataTip (see vstipDebug0005, [07.12 Pin a DataTip to Source Code](#), on page 305). Now, to make it a floating tip, put your mouse over the pinned tip until you see the control panel shown in the following illustration.

### NOTE

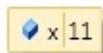
The control panel might not come up exactly where the pinned tip is, so you might have to look around a bit to find it.



For this tip, our focus is on the pin in the middle:



Click it, and you should get a floating DataTip. (Notice the yellow color for floating DataTips.)



OK, so why should you care? Well, unlike pinned DataTips, floating DataTips

don't follow the source code as you step through it. This is useful if you want to step through code and have one or more pieces of information. The control panel might not come up exactly where the pinned tip is, so you might have to look around a bit to find it.



## 07.14 Adding Comments to a DataTip

VERSIONS	2010
CODE	vstipDebug0007

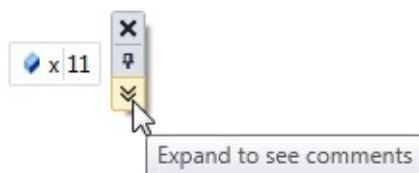
You might want to make a comment to remind yourself about something in a DataTip (pinned or floating), and now you can. First, enter Debug Mode, and then pause your mouse over a pinned (see vstipDebug0005, [07.12 Pin a DataTip to Source Code](#), on page 305) or floating tip (see vstipDebug0006, [07.13 Create a Floating DataTip](#), page 306) until you see the control panel shown in the following illustration.

### NOTE

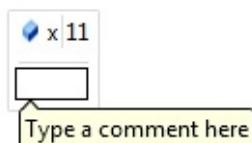
The control panel might not come up exactly where the pinned tip is, so you might have to look around a bit to find it.



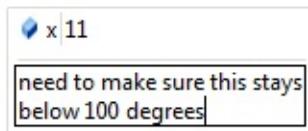
Click the chevron at the bottom:



You should see something like the following illustration on your DataTip:



Now you can put in comments that travel with the DataTip at all times:



At the time of this writing, no upper limit is assigned to the amount of text you can put into this area. I was able to successfully paste the entire text of “War and Peace” in here. While I don’t suggest you do the same, the point is that you can be quite verbose with your comments if you need to be.

## 07.15 Use a DataTip to Edit a Value

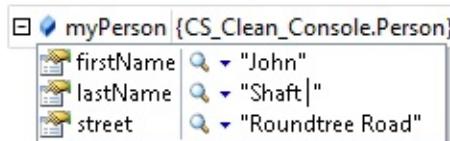
VERSIONS	2008, 2010
CODE	vstipDebug0026

You can change a variable value on the fly in a variety of ways. One of them is through the DataTip. Just click the value for a simple variable to change it:

```
x = x + 1;
```



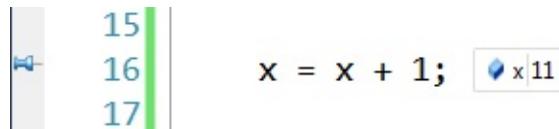
For more complex types, you might need to expand the variables and edit individual items:



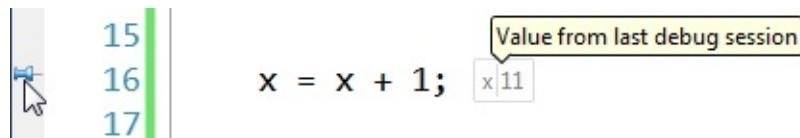
## 07.16 DataTip Value from the Last Debug Session

VERSIONS	2010
CODE	vstipDebug0012

Ever forget the values of variables you were just debugging? This is one of the coolest features of the DataTips in Visual Studio 2010. Let's assume that you have a pinned DataTip in your code (see vstipDebug0005, [07.12 Pin a DataTip to Source Code](#), on page 305):



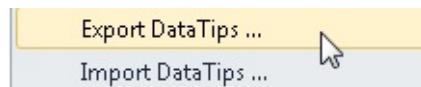
Now you stop debugging. Even though you are not debugging, you can still view the value from the last debug session by simply resting your mouse pointer over the pin in the margin:



## 07.17 Import and Export DataTips

<b>WINDOWS</b>	Alt,D, X,X, Enter; Alt,D, P, P, Enter
<b>MENU</b>	Debug   Export DataTips; Debug   Import DataTips
<b>COMMAND</b>	Debug.ExportDataTips; Debug.ImportDataTips
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0013

Just as with breakpoints (see vstipDebug0003, [07.26 Import and Export Breakpoints](#), on page 329), you now have the ability to share your DataTips with team members by using the Export / Import features. Just go to Debug | Import (or Export) DataTips:

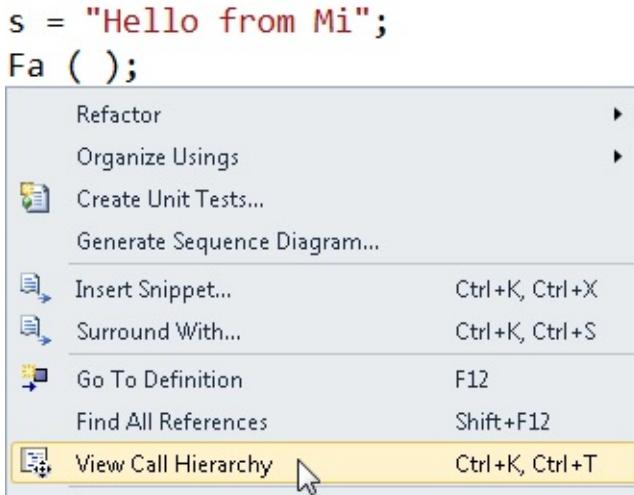


When you export the DataTips, they are exported as XML. You can version these files along with your source code so that other team members can have a copy of your DataTips.

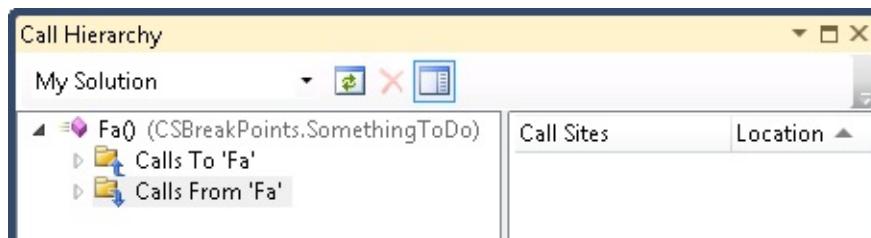
## 07.18 Using the Call Hierarchy

<b>DEFAULT</b>	Ctrl+K, Ctrl+T; Ctrl+K, T
<b>VISUAL C++ 2</b>	[no shortcut]
<b>MENU</b>	[Context Menu]   View Call Hierarchy
<b>COMMAND</b>	EditorContextMenus.CodeWindow.ViewCallHierarchy
<b>VERSIONS</b>	2010
<b>LANGUAGES</b>	C++, C#
<b>CODE</b>	vstipTool0005

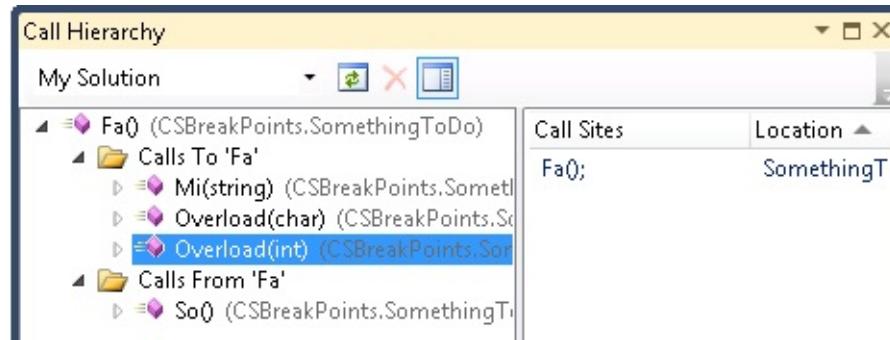
The Call Hierarchy window allows you to visualize calls to and from a selected method, property, or constructor. To see how it works, just right-click any method, property, or constructor in the editor and select View Call Hierarchy:



You should get a window similar to the following:



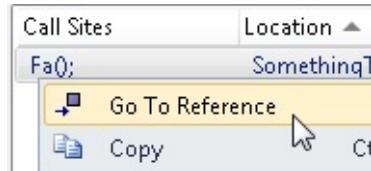
Notice the Calls To and Calls From areas related to your selection. You can expand them:



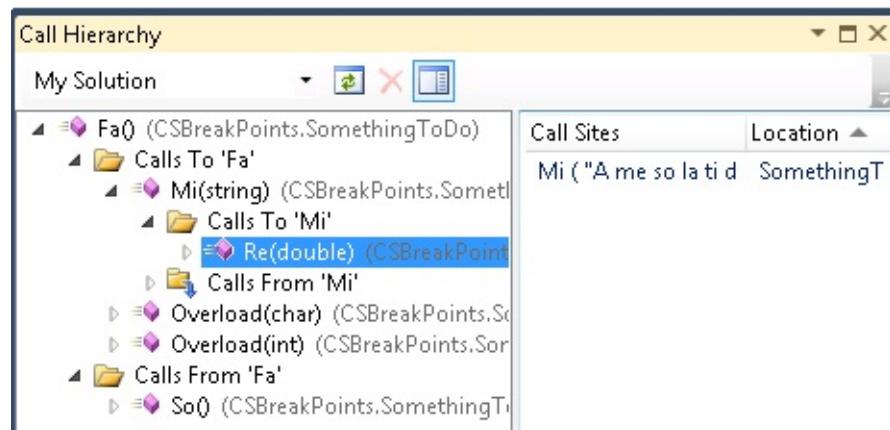
When you click on a node in the tree, the Call Sites pane updates so that you can visit the call if you want to:

#### NOTE

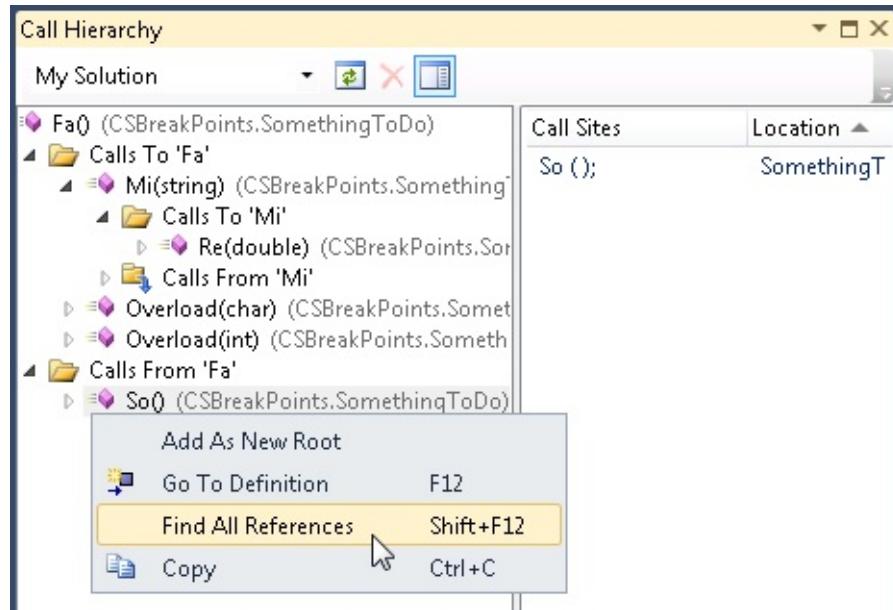
You can double-click on the call site to have it automatically take you to the reference.



You can continue expanding the hierarchy to see further Calls To and Calls From information:



The best part is that you can right-click a symbol and get several options:



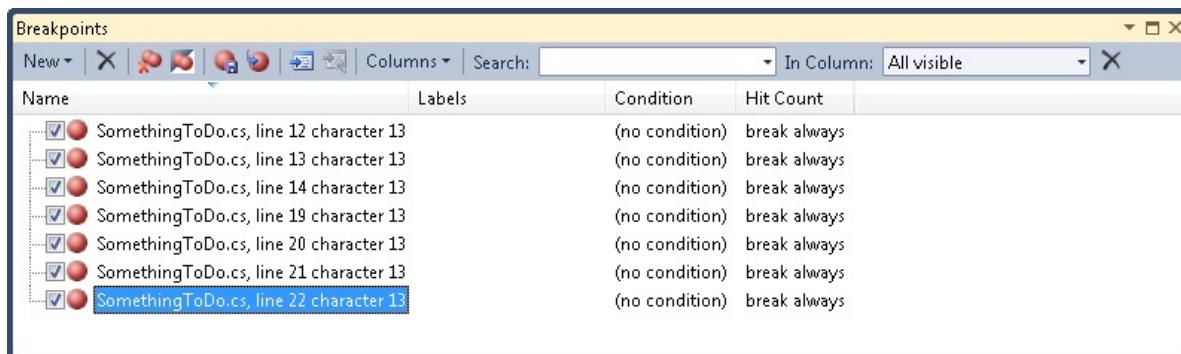
The following table describes some options you will come across as you use this feature:

CONTEXT MENU ITEM	DESCRIPTION
<b>Add As New Root</b>	Adds the selected node to the tree view pane as a new root node.
<b>Remove Root</b>	Removes the selected root node from the tree view pane. This option is available only from a root node.  You can also use the Remove Root toolbar button to remove the selected root node.
<b>Go To Definition</b>	Runs the Go To Definition command on the selected node. This navigates to the original definition for a method call or variable definition.  You can also press F12 to run the Go To Definition command on the selected node.
<b>Find All References</b>	Runs the Find All References command on the selected node. This finds all the lines of code in your project that reference a class or member.  You can also use Shift+F12 to run the Find All References command on the selected node.
<b>Copy</b>	Copies the contents of the selected node (but not its subnodes).
<b>Refresh</b>	Collapses the selected node so that re-expanding it displays current information.

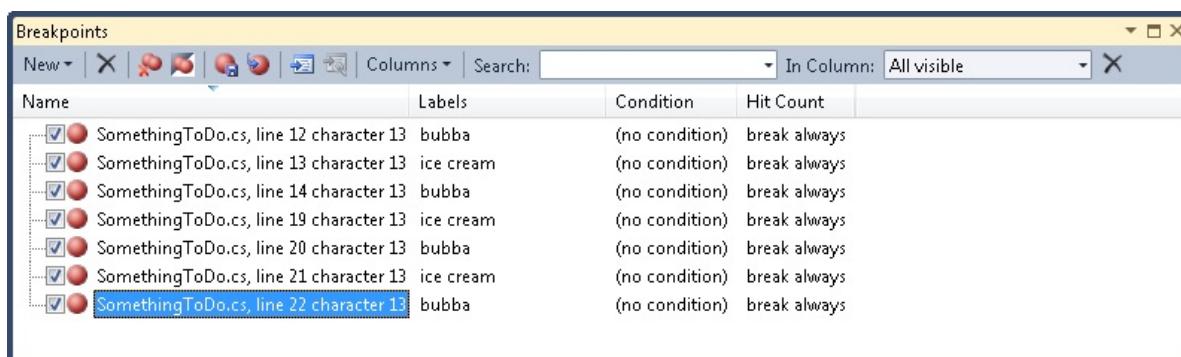
## 07.19 Searching Breakpoints

VERSIONS	2010
CODE	vstipDebug0002

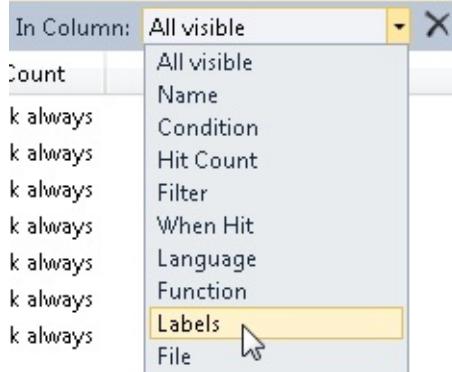
When working with larger sets of breakpoints, it is useful to be able to search and filter based on criteria you choose. In Visual Studio 2010, you finally have the ability to search breakpoints the way you want. The ability to search your breakpoints is critical to being able to take actions on groups of breakpoints because the Breakpoints window commands now act on breakpoints that match the current search criteria. First, set some sample breakpoints in your code, and then open the Breakpoints window (Ctrl+Alt+B):



For this example, set some labels for your breakpoints (see [vstipDebug0001, 07.03 Adding Labels to Breakpoints, on page 293](#)) that have values you would like to search on:



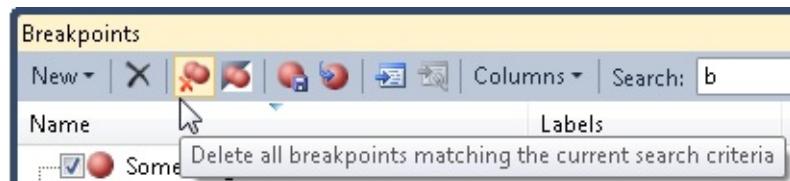
Select the column you want to search on (in this example, the Labels column):



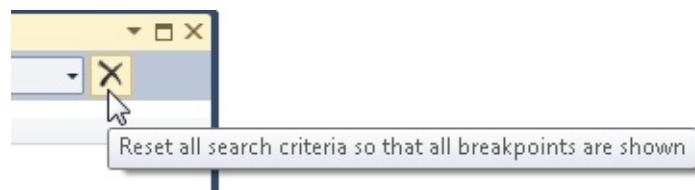
Type the string you are looking for in the search box inside the Breakpoints window, and press Enter. In this example, I'll search for any label with the letter *b* anywhere in it. This yields the following result:

Name	Labels	Condition	Hit Count
[checkbox] SomethingToDo.cs, line 12 character 13	bubba	(no condition)	break always
[checkbox] SomethingToDo.cs, line 14 character 13	bubba	(no condition)	break always
[checkbox] SomethingToDo.cs, line 20 character 13	bubba	(no condition)	break always
[checkbox] SomethingToDo.cs, line 22 character 13	bubba	(no condition)	break always

Now you can take actions on the breakpoints you can currently see. Most actions in the Breakpoints window now act on the results of the current search criteria:



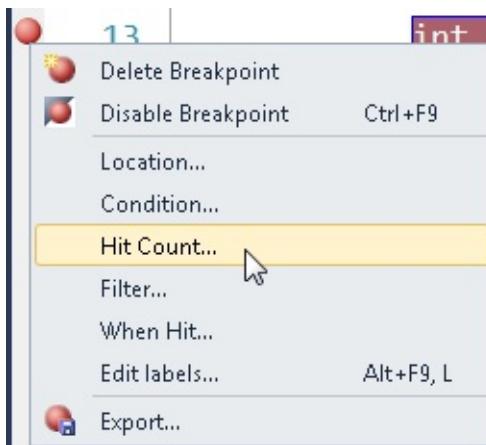
To clear the search so that you can see all the breakpoints, just click the Reset All Search Criteria So That All Breakpoints Are Shown button:



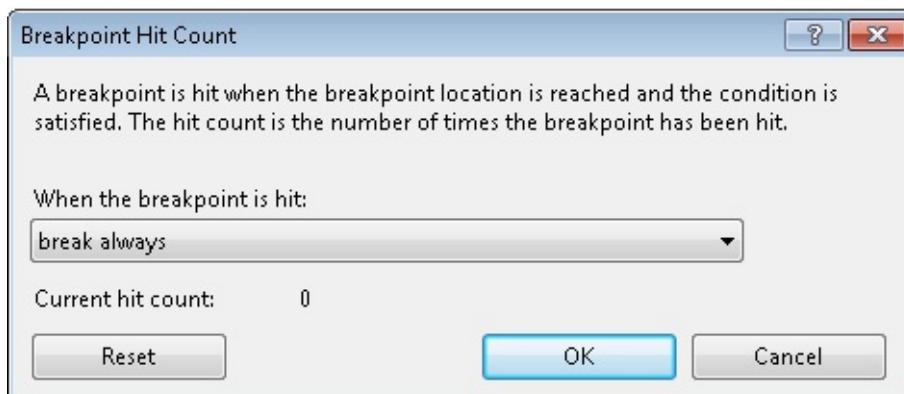
## 07.20 Breakpoint Hit Count

<b>MENU</b>	[Context Menu]   Hit Count
<b>COMMAND</b>	EditorContextMenus.CodeWindow.Breakpoint.BreakpointHitCount
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0019

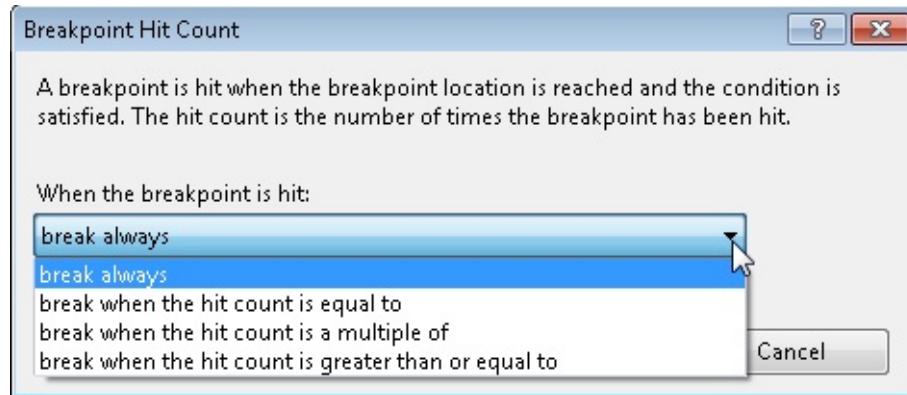
Sometimes you might not want a breakpoint to stop the first time it is encountered. This is particularly true when you are working with loops and want to stop after a number of iterations. You can set a breakpoint so that it does not break every time but only when it is hit a certain number of times. Just right-click any breakpoint, and click Hit Count:



The Breakpoint Hit Count dialog box appears:

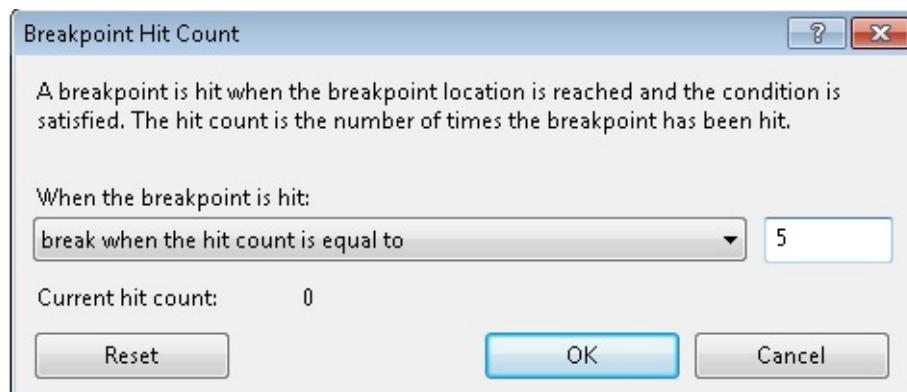


The default is to “break always,” but you can change that to one of the following options:



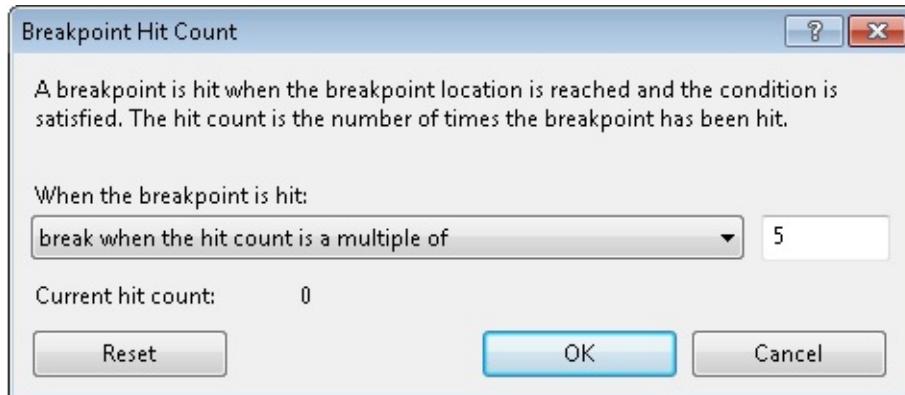
## Break When The Hit Count Is Equal To

Choose this option if you want to break when the hit count reaches an exact value. So, in this example, if you put in a 5, the break does not occur until the breakpoint has been hit 5 times. This is useful when you know the number of hits you want before stopping.

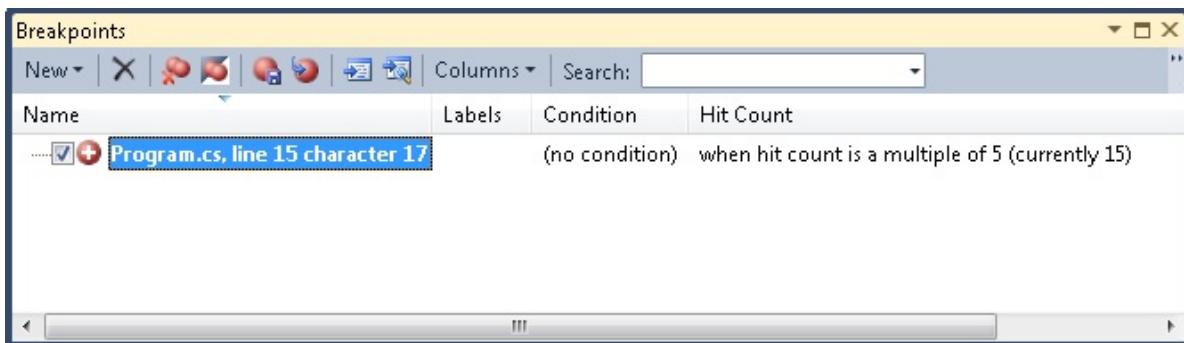


## Break When The Hit Count Is A Multiple Of

This breaks every x number of times it is hit. So if you put in a 5, it breaks every 5th time (5, 10, 15, and so on). This helps when you aren't sure exactly where the problem is, so you want to skip in predefined increments as you look:

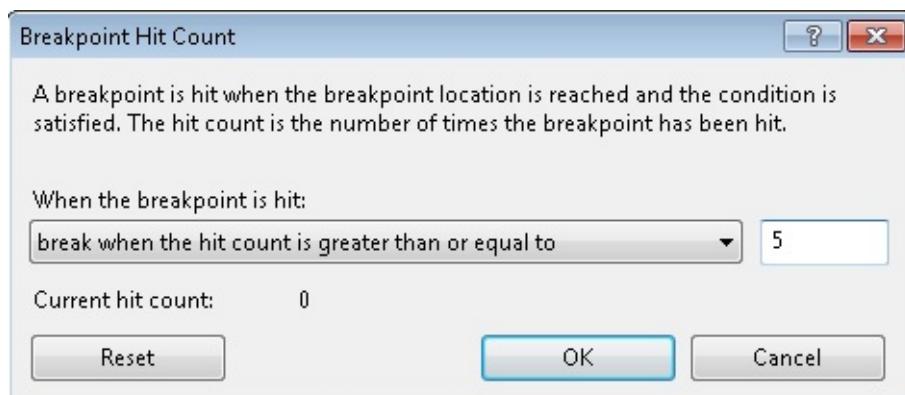


At any time, you can use the Breakpoints window to tell you the current hit count while you are debugging. In this example, the hit count is currently 15:



## Break When The Hit Count Is Greater Than Or Equal To

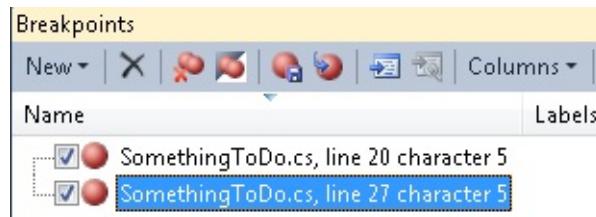
This option takes any number and stops when the hit count has reached that number or higher. Use this option when you aren't exactly sure what the value should be but want to stop when it reaches an upper bound:



## 07.21 Set a Breakpoint on a Function

<b>DEFAULT</b>	Ctrl+B
<b>VISUAL BASIC 6</b>	Ctrl+B
<b>VISUAL C# 2005</b>	Ctrl+B; Ctrl+D, Ctrl+N; Ctrl+D, N
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+B
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt,D, B, F
<b>MENU</b>	Debug   New Breakpoint   Break at Function
<b>COMMAND</b>	Debug.BreakatFunction
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0020

By default, breakpoints are based on line and character position:

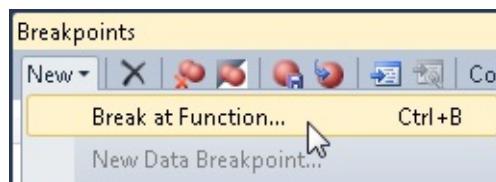


```
17 public void Re ( double d )
18 {
19     d = 0.667;
20     double eLocal = 0.002;
21     eLocal = eLocal * d;
22     Mi ( "A me so la ti do" );
23 }
24
25 public void Mi ( String s )
26 {
27     s = "Hello from Mi";
28     Fa ( );
29 }
```

But what if you don't want to break on a specific line but instead want to break when you hit a particular function? There are two ways to do this.

## Breakpoints Window

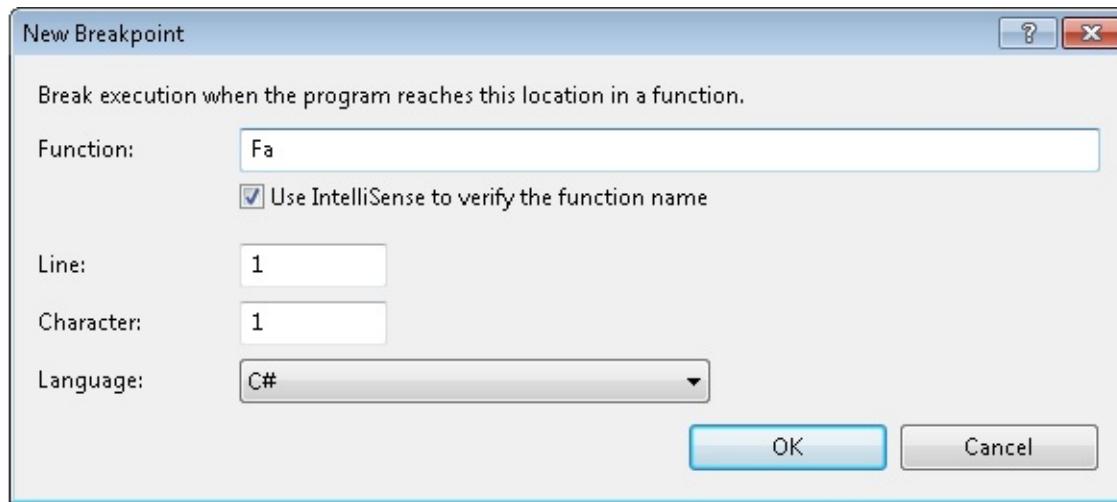
In the Breakpoints window, click New and choose Break At Function (or press Ctrl+B) to bring up the New Breakpoint dialog box.



In the New Breakpoint dialog box, type in the name of the function you want to break at, and then click OK.

### WARNING

To ensure that the correct function name is being used, always select Use IntelliSense To Verify The Function Name when you perform this action. Otherwise, the breakpoint might not work.



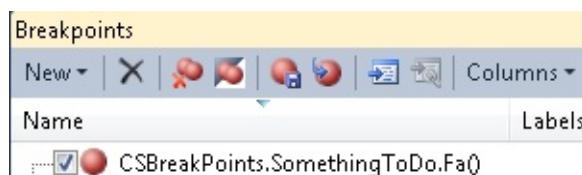
After you click OK in the New Breakpoint dialog box, you should see something similar to the following:

```
31 | public void Fa ( )  
32 | {  
33 |     So ( );  
34 | }
```

A code editor window showing a C# method definition. A red circular breakpoint icon is positioned to the left of the first line of code (line 31). The code is as follows:

```
public void Fa ( )  
{  
    So ( );  
}
```

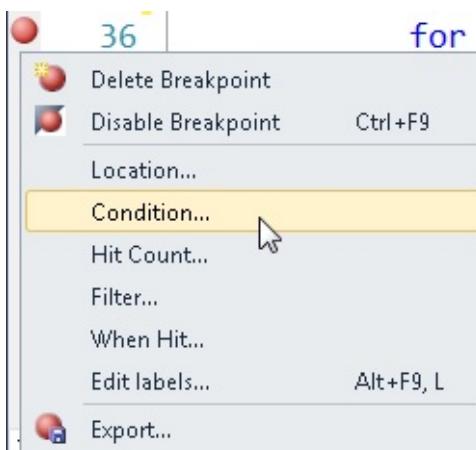
The next illustration shows what a function breakpoint looks like in the Breakpoints window:



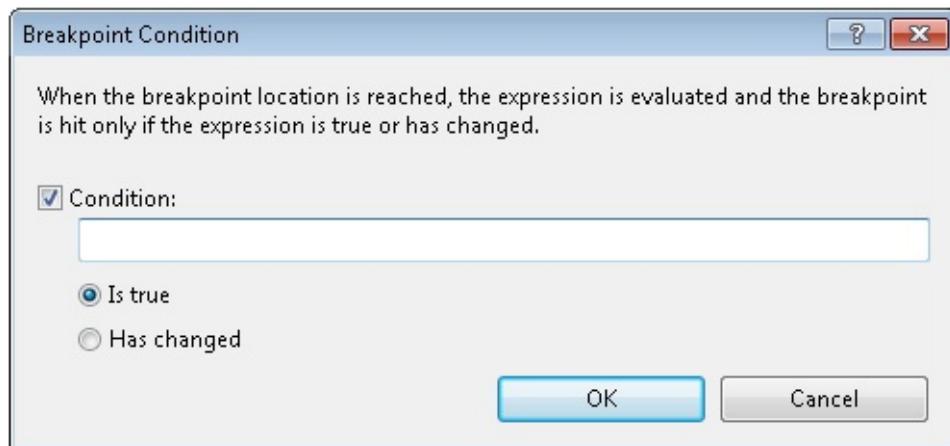
## 07.22 Set a Simple Breakpoint Condition

<b>MENU</b>	[Context Menu]   Condition
<b>COMMAND</b>	EditorContextMenus.CodeWindow.Breakpoint.BreakpointCondition
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0021

Conditional breakpoints are arguably the most powerful types of breakpoints you can set. Many steps are involved in using them correctly, and knowing what steps you need to take is half the battle. Start by right-clicking on any breakpoint and choosing Condition.



This gives us the Breakpoint Condition dialog box. Notice that the condition can be turned off by clearing the Condition check box. Additionally, two options are available for the condition that you set:



- **Is true**

Used for Boolean expressions that evaluate to true or false.

- **Has changed**

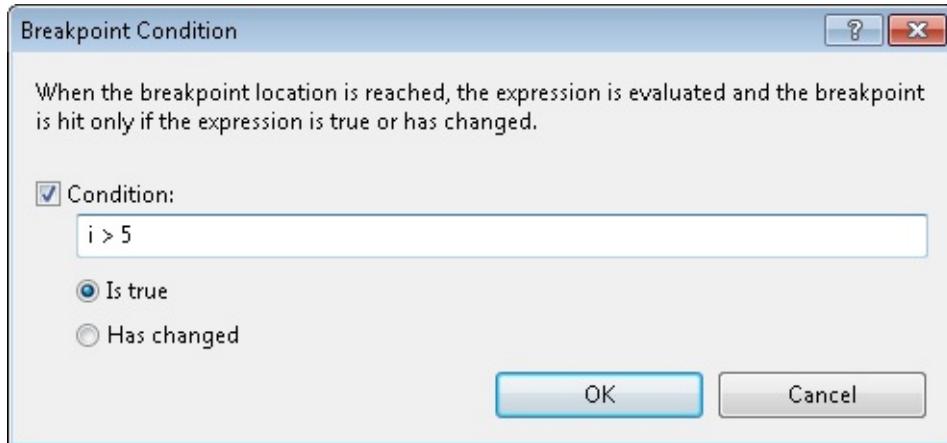
Used for detecting whether the value of an expression has changed at the breakpoint location.

Let's take a look at a couple of examples. Suppose we have the following For loop:

```
for ( i = 0 , m = 0 ; i < 10 ; i++ , m-- )  
{  
    sb.Length = 0;  
    sb.AppendFormat ( "i = {0} m = {1}" , i , m );  
    Trace.WriteLine ( sb.ToString ( ) );  
}
```

## Is True

We can set a simple “Is true” condition that says when the variable “i” is greater than 5, the code should stop:

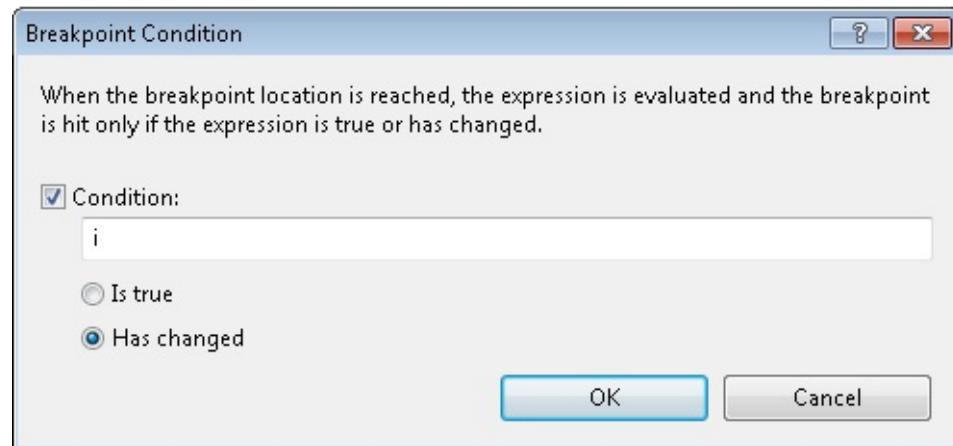


When we run the code and the breakpoint is hit, sure enough, it stops when the value of “i” is greater than 5:

```
i[ < 10 ;  
    i| 6 □
```

## Has Changed

This one is more interesting. Basically we set up something to watch. In this case, let's just have it watch the “i” variable:



When the breakpoint is hit and the value of "i" has changed in any way, the code stops:

i] < 10 ;  
i | 0 ←

## Special Notes

Anytime you set an advanced breakpoint, you get a new glyph (red sphere with a plus sign in it):



38

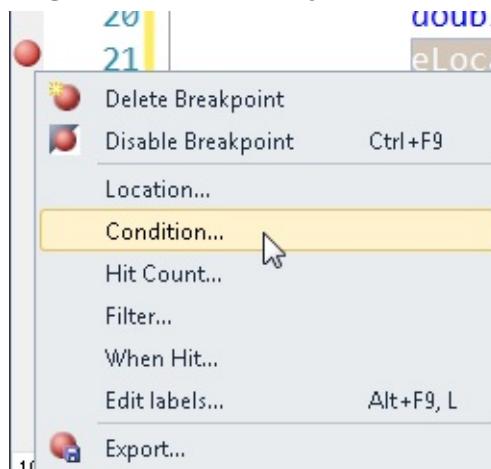
You can always tell what kind of breakpoint you have by pausing your mouse over the glyph and looking at the tooltip or by looking in the Breakpoints window:

Breakpoints		
Name	Labels	Condition
... MainForm.cs, line 38 character 17		when 'i' has changed

## 07.23 Set a Complex Breakpoint Condition

<b>MENU</b>	[Context Menu]   Condition
<b>COMMAND</b>	EditorContextMenus.CodeWindow.Breakpoint.BreakpointCondition
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0022

We previously discussed how to set simple conditions. (See vstipDebug0021, [07.22 Set a Simple Breakpoint Condition](#), on page 318.) The real power of the Breakpoint Condition dialog box is the ability to execute any line of code from it:



For example, you can use it to call external methods. The following example illustrates this, but bear in mind that it's a contrived example, designed to show how this works. The following illustration shows the code we want to execute:

```
public void Re(double d)
{
    d = 0.667;

    int x = 20;

    double eLocal = x + d;
    eLocal = eLocal * d;
    Mi("A me so la ti do");
}
```

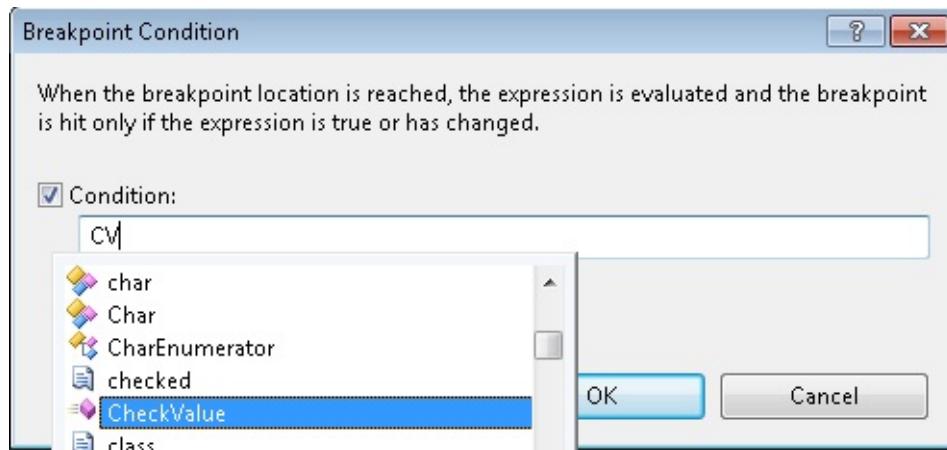
Notice that a variable, x, is given a value. In this case, I hard-coded a value, but

the value would presumably come as the result of some method execution. I want the breakpoint to stop only if the value of x is 20. I have a method that returns a Boolean value to test for the condition I want:

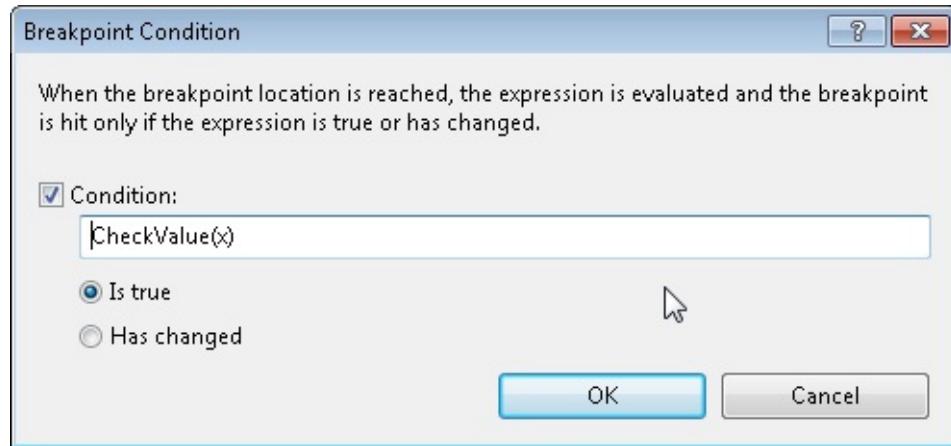
```
public bool CheckValue(int someValue)
{
    if (someValue == 20)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

I won't get into the obvious debate about whether this is a good idea or not, how the method should be constructed, or if the average rainfall in the Amazon Basin is a factor here. I'll just call the method from the Breakpoint Condition dialog box.

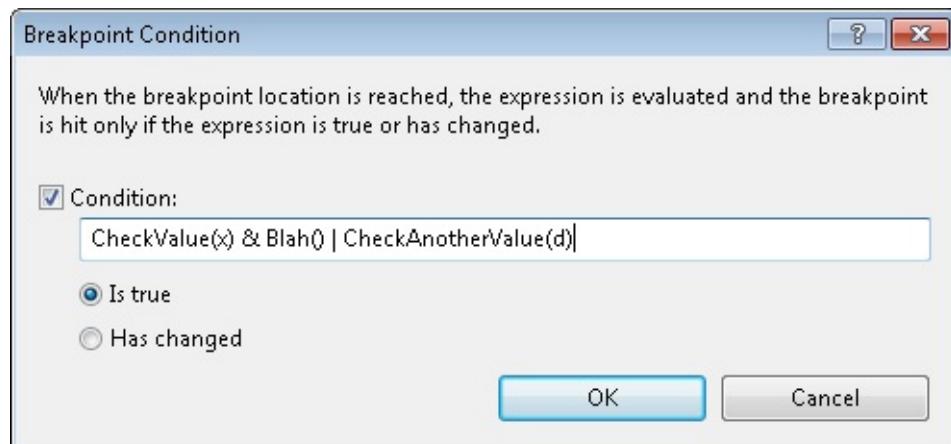
Notice that after typing a few characters, I use Ctrl+Space to get IntelliSense here if needed (see vstipEdit0017, [06.07 Using the New IntelliSense: Pascal Case](#), on page 216):



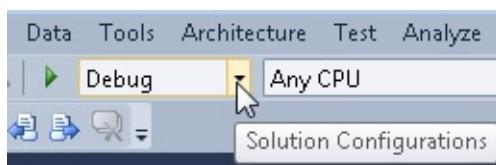
Make sure the Is True option is selected:



Now I have a conditional breakpoint that uses an external method to determine whether or not the code should stop. Naturally, you can add more Boolean logic here as well and include And / Or / Not / Xor:



Now the really interesting part is that you can have your checking code be available only for Debug builds:



By surrounding your code with conditional compilation directives, you can easily have checking code that is around only while debugging. Following is what the checking method would look like in C#:

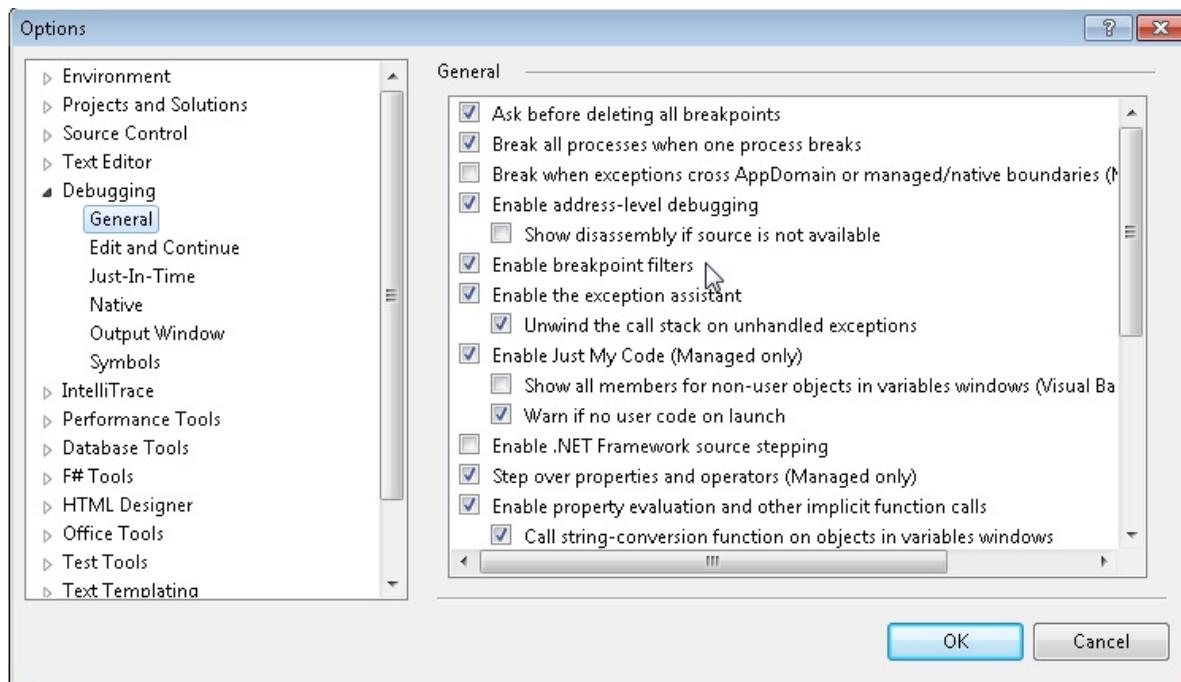
```
#if DEBUG
    public bool CheckValue(int someValue)
    {
        if (someValue == 20)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
#endif
```

Play with this feature some, and you can see why most people (including me) think that this is one of the most powerful breakpoint features around.

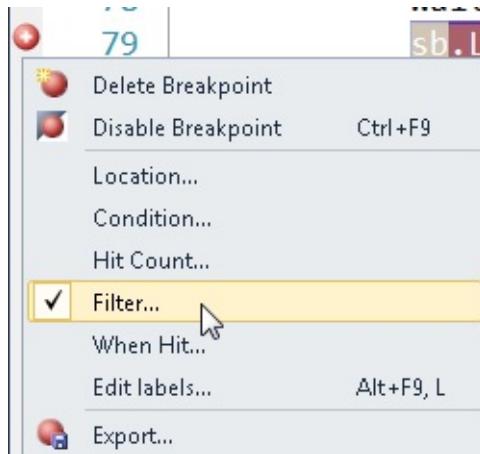
## 07.24 Setting a Breakpoint Filter

<b>WINDOWS</b>	Alt,T, O (options)
<b>MENU</b>	Tools   Options; [Context Menu]   Filter
<b>COMMAND</b>	Tools.Options; EditorContextMenus.CodeWindow.Breakpoint.BreakpointFilter
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0024

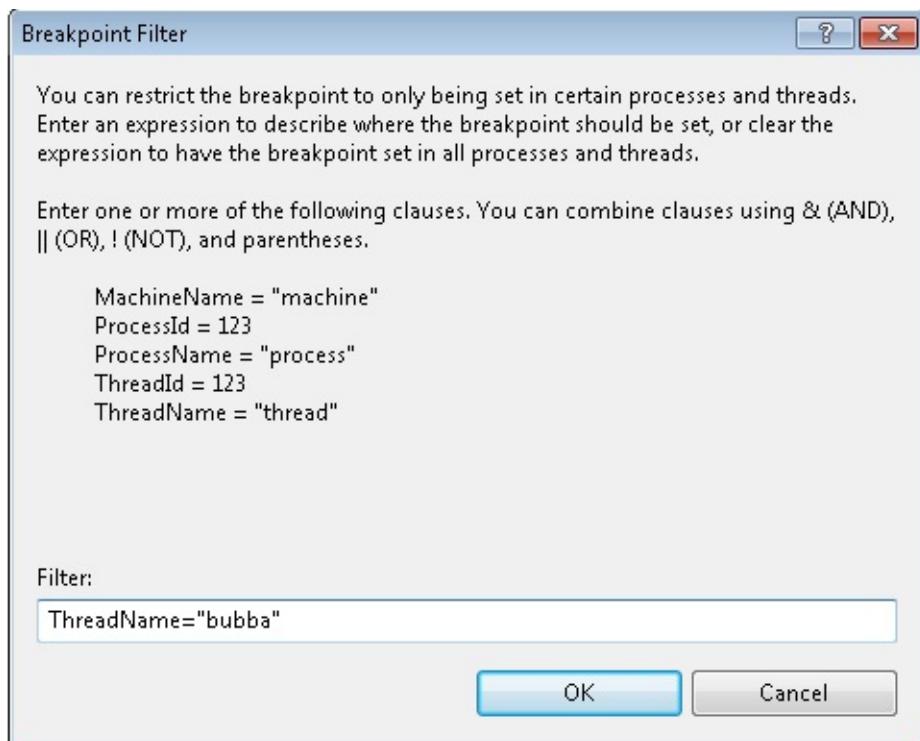
Breakpoint filters are used when you want to break based on thread, process, or machine information. This is particularly useful for debugging multithreaded applications. To make sure you can set them, go to Tools | Options | Debugging | General and select Enable Breakpoint Filters:



To make a breakpoint filter, just right-click any regular breakpoint and choose Filter:



You get the Breakpoint Filter dialog box:



As shown in the preceding graphic, I've decided to break whenever the breakpoint is hit and the thread name is “bubba.” The following illustration shows what it looks like in the Threads window when I actually run my code and the breakpoint is hit:

▼	1928	11	Worker Thread	Thread 2
▼ ➔	7912	12	Worker Thread	bubba
▼	7712	13	Worker Thread	Thread 4
▼	8648	14	Worker Thread	Thread 5

## 07.25 Setting a Tracepoint in Source Code

MENU	[Context Menu]   When Hit
COMMAND	EditorContextMenus.CodeWindow.Breakpoint.BreakpointWhenHit
VERSIONS	2005, 2008, 2010
CODE	vstipDebug0010

Tracepoints give you the opportunity to unobtrusively print out information during application execution. This tip shows you how to use the IDE to create tracepoints, but for detailed information about how to do this in code, see “Tracing and Instrumenting Applications” at [http://msdn.microsoft.com/en-us/library/zs6s4h68\(VS.100\).aspx](http://msdn.microsoft.com/en-us/library/zs6s4h68(VS.100).aspx).

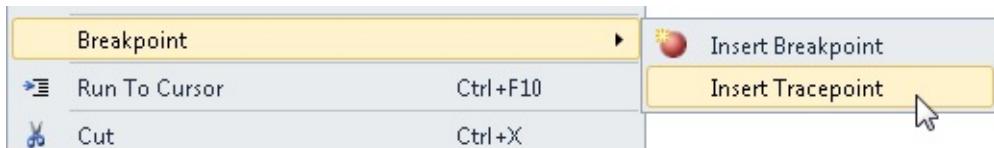
The best way to illustrate simple tracepoints is with a basic loop. I would suggest creating a new project and making a simple for loop to play with. Here is my sample code:

```
static void Main(string[] args)
{
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine("I'm doing something!");
    }
}
```

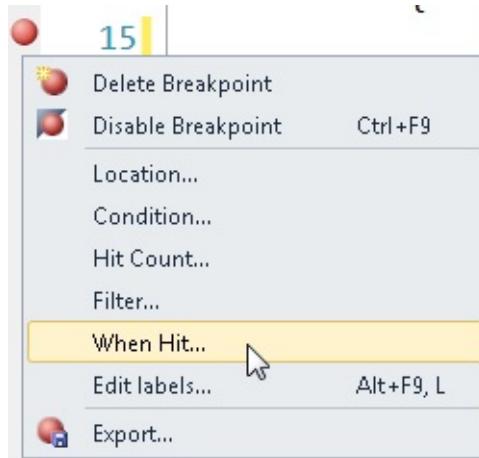
## Setting Tracepoints

You have a couple of ways to set a tracepoint on a line of code.

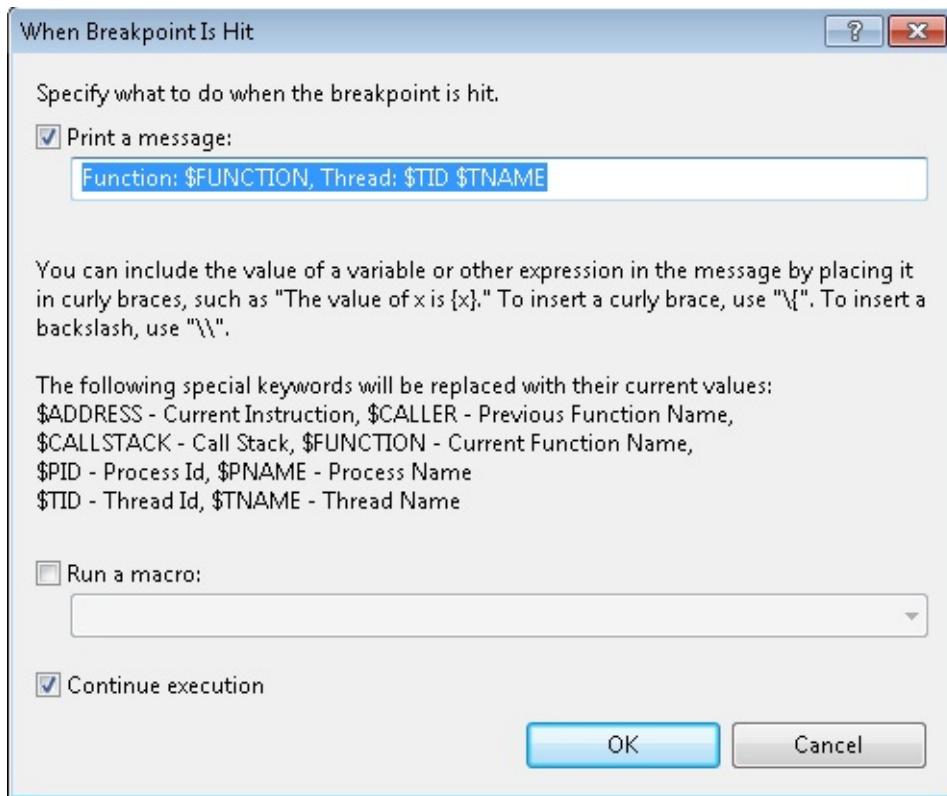
One way is to right-click on the line and choose Breakpoint | Insert Tracepoint.



Alternatively, you can set a breakpoint (F9) and then right-click on the breakpoint in the indicator margin and choose When Hit:



Whichever technique you use, the When Breakpoint Is Hit dialog box appears:



The When Breakpoint Is Hit dialog box provides three options.

## Print a message

Used to print out any special variables (that begin with a \$), evaluated expressions (inside curly braces), or literal text.

## Run a macro

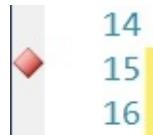
Used to actually run a macro when this tracepoint is hit and can do extended

processing or kick off some other task.

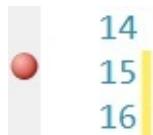
## Continue execution

Makes the tracepoint unobtrusive and lets the application run. If you clear the Continue Execution check box, the tracepoint becomes a breakpoint.

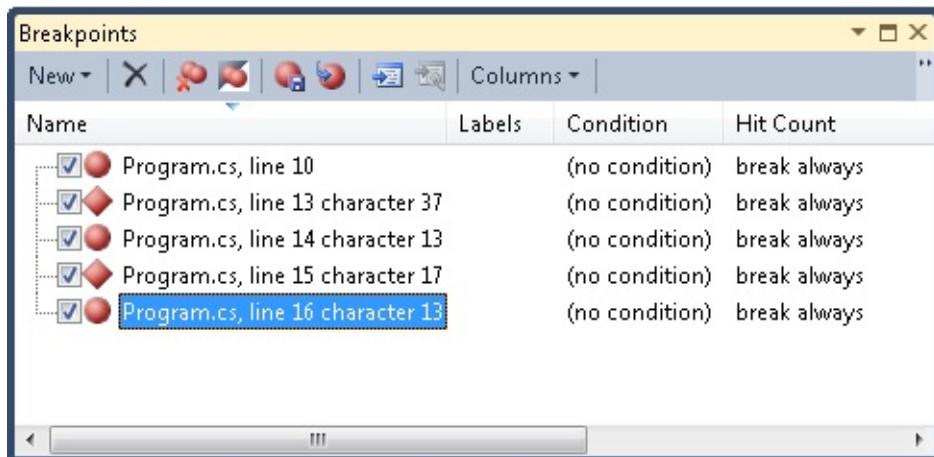
Click OK and notice something interesting. The normal round breakpoint indicator is now a diamond:



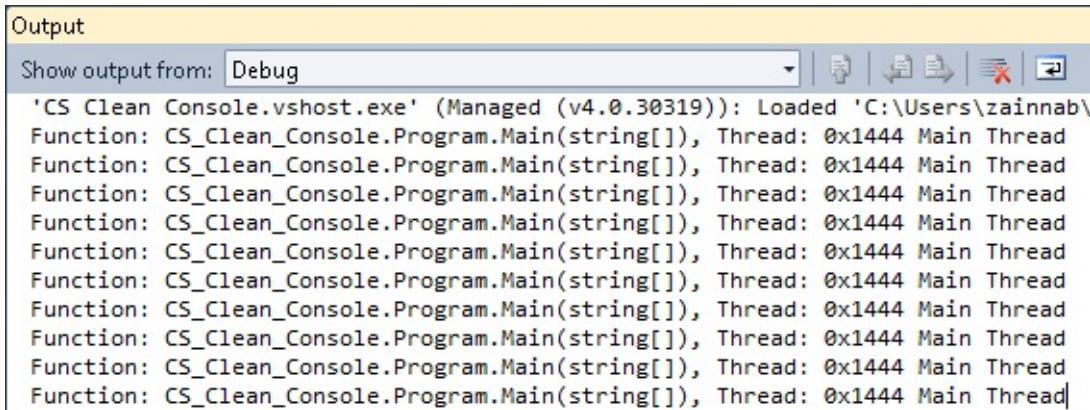
This is how we distinguish breakpoints (stop execution) from tracepoints (don't stop execution). For example, if I were to clear the Continue Execution check box in the When Breakpoint Is Hit dialog box, the breakpoint symbol would become round again:



Tracepoints show up along with breakpoints in the Breakpoints window:



At this point, you have done enough to see tracepoints in action, so run your application. It should execute and then end. Open the Output window (Debug | Windows | Output), and notice the entries from our tracepoint:

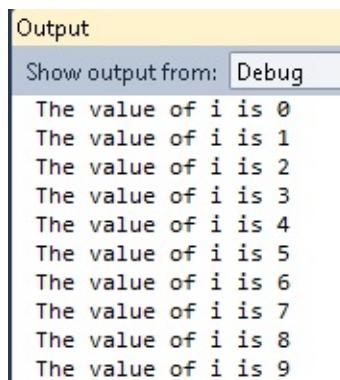


```
Output
Show output from: Debug
'CS Clean Console.vshost.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\zainnab\Function: CS_Clean_Console.Program.Main(string[]), Thread: 0x1444 Main Thread
Function: CS_Clean_Console.Program.Main(string[]), Thread: 0x1444 Main Thread
```

The default message isn't very helpful in this case, so let's change it to something else. In this example, let's put **The value of i is {i}**. Notice that I put the expression to be evaluated (the variable i in this case) inside curly braces:



Now you should see the following in the Output window:



```
Output
Show output from: Debug
The value of i is 0
The value of i is 1
The value of i is 2
The value of i is 3
The value of i is 4
The value of i is 5
The value of i is 6
The value of i is 7
The value of i is 8
The value of i is 9
```

## Change Default Message

There is a lot more to learn here, but you have a good start. One thing you might want to do is permanently change the default output message (currently “Function: \$FUNCTION, Thread: \$TID \$TNAME”) that you get when you set a new tracepoint. This is useful if you find that you use a certain message often across projects. You can do this by going into the registry:

HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\<version>\Debugger

**WARNING**

Editing the registry can cause issues with Visual Studio, so perform these steps at your own risk.

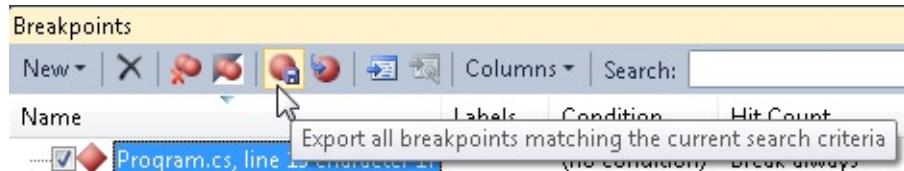
Then modify the string value called DefaultTracepointMessage to the new default you would like to have.

## 07.26 Import and Export Breakpoints

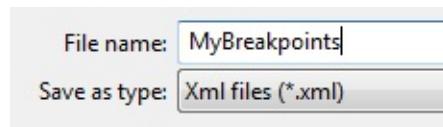
<b>COMMAND</b>	EditorContextMenus.CodeWindow.Breakpoint.BreakpointExport; DebuggerContextMenus.BreakpointsWindow.Exportselected
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipDebug0003

These next features are specifically designed so that you can share breakpoints with other team members. When you do this, make sure to version your exported breakpoints with your source code for sharing.

Set one or more breakpoints in your code, and open the Breakpoints window (Ctrl+Alt+B). Notice the new Export button. It's important to understand that it exports breakpoints *matching the current search criteria*. In other words, if you don't see the breakpoint in the Breakpoints window, it will not be exported:

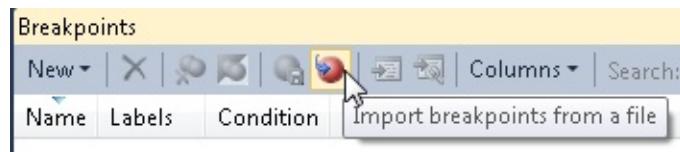


When you click the Export button, you get the classic Save As dialog box; notice that your breakpoints are saved as an XML file:



Put in some name for the file, and click Save. Give the file you created to your teammate or, if you are just practicing, you can delete your breakpoints.

Click on the new Import button:



Choose the XML file that was exported, and click Open to import your breakpoints.

## 07.27 Run to Cursor

<b>DEFAULT</b>	Ctrl+F10
<b>VISUAL BASIC 6</b>	Ctrl+F10; Ctrl+F8
<b>VISUAL C# 2005</b>	Ctrl+F10
<b>VISUAL C++ 2</b>	Ctrl+F10; F7
<b>VISUAL C++ 6</b>	Ctrl+F10
<b>VISUAL STUDIO 6</b>	Ctrl+F10
<b>WINDOWS</b>	[no shortcut]
<b>MENU</b>	[Context Menu]   Run To Cursor
<b>COMMAND</b>	Debug.RunToCursor
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0023

This is one that has been around a while but seems to get lost among all the other features that are out there. Basically, if you have some code and want to quickly run it and set a temporary breakpoint at the same time, this tip is for you.

Just put the cursor on the line you want to break on:

```
static void Main(string[] args)
{
    Console.WriteLine("Go!");
}
```

Press Ctrl+F10, or right-click and choose Run To Cursor:



The application starts and a temporary Breakpoint is set on the line where you were, *but you do not see any breakpoint indicator*. The next time the code hits that line, it enters break mode:

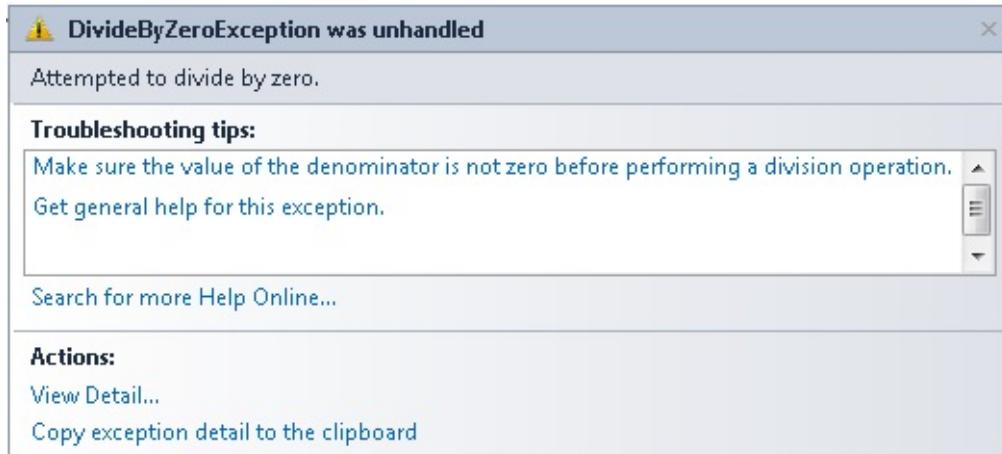
```
static void Main(string[] args)
{
    Console.WriteLine("Go!");
}
```

Keep in mind that the application does not break until the line the temporary breakpoint is on is hit. If the line is never executed, the application will not break for debugging.

## 07.28 Using the Exception Assistant

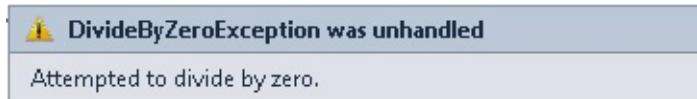
VERSIONS	2005, 2008, 2010
CODE	vstipDebug0030

The Exception Assistant appears whenever a runtime exception occurs. It shows the type of exception, troubleshooting tips, and corrective actions as applicable:



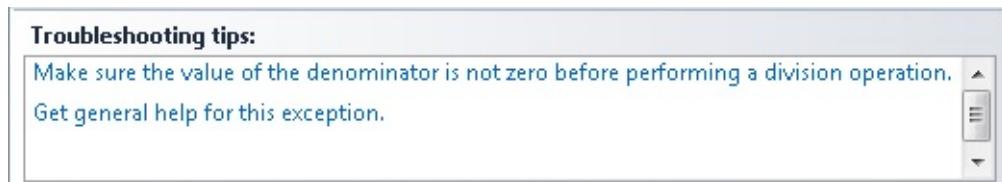
## Exception Object and Description

The layout is pretty straightforward. First you have the exception object type and description:



## Troubleshooting Tips

Then you have the Troubleshooting Tips area that provides advice about how to resolve the problem in a user-friendly format. The items are typically links to more information that can be found in the online or offline Help:

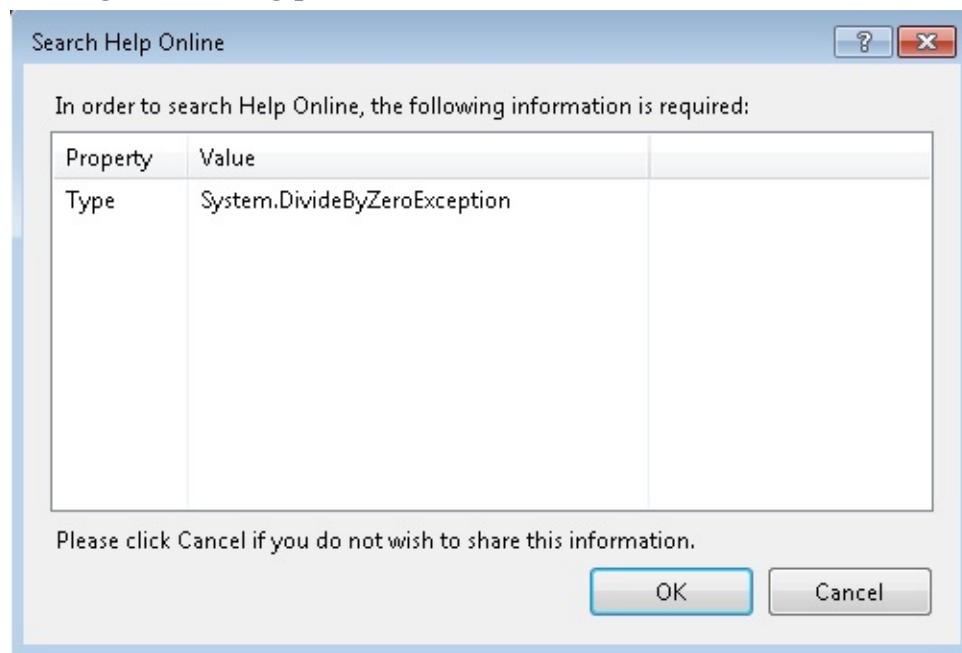


## Help Online

Search For More Help Online is pretty interesting when you click the link:

Search for more Help Online...

You get a dialog box asking permission to send information online:



Clicking OK results in sending the information to MSDN online and performing a search:

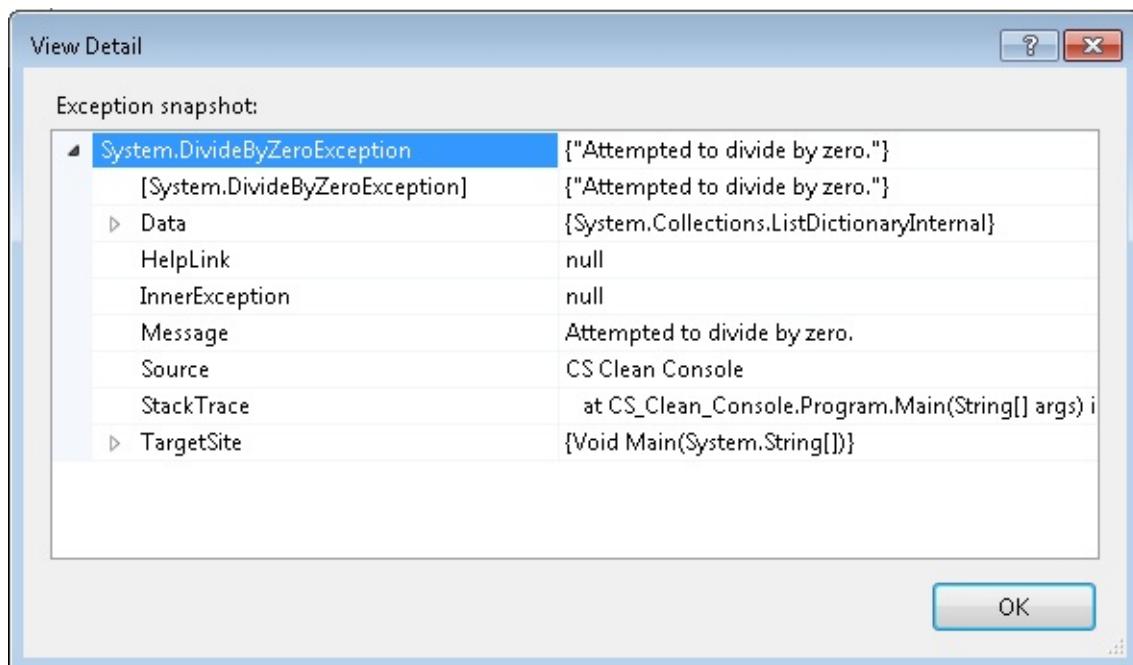
The screenshot shows a browser window with the title 'System.DivideByZeroException - MSDN Search'. The URL is 'http://social.msdn.microsoft.com/search/en-US?query=System.DivideByZeroException'. The page is from the MSDN website. The search bar contains 'System.DivideByZeroException'. The results section shows 'Results 1-20 of about 1,350 for: System.DivideByZeroException (0.968 seconds)'. The first result is 'DivideByZeroException Class (System)' with a link to 'msdn.microsoft.com/en-us/library/system.dividebyzeroexception'. On the left, there is a sidebar with 'Refine search' sections for 'By Source' (Documentation & Articles, CodePlex, Forums, MSDN Magazine) and 'By Topic' (Windows CE, .NET Development, .NET Framework 1.1 Library).

## Actions

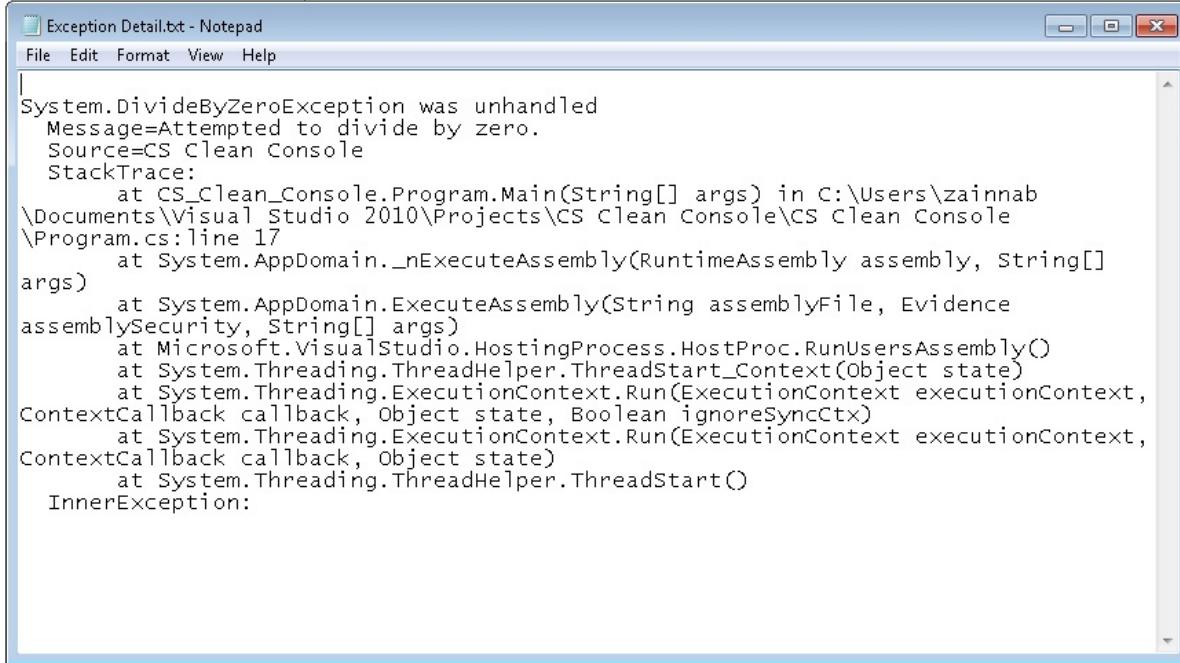
The Actions area lets you get information about the Exception Object:

**Actions:**  
[View Detail...](#)  
[Copy exception detail to the clipboard](#)

The View Detail link opens a dialog box that exposes the details of the object for you to review:



The Copy Exception Detail To The Clipboard link captures textual information you can put into any editor for analysis. It's not overly detailed but does provide a starting point for resolving the issue:

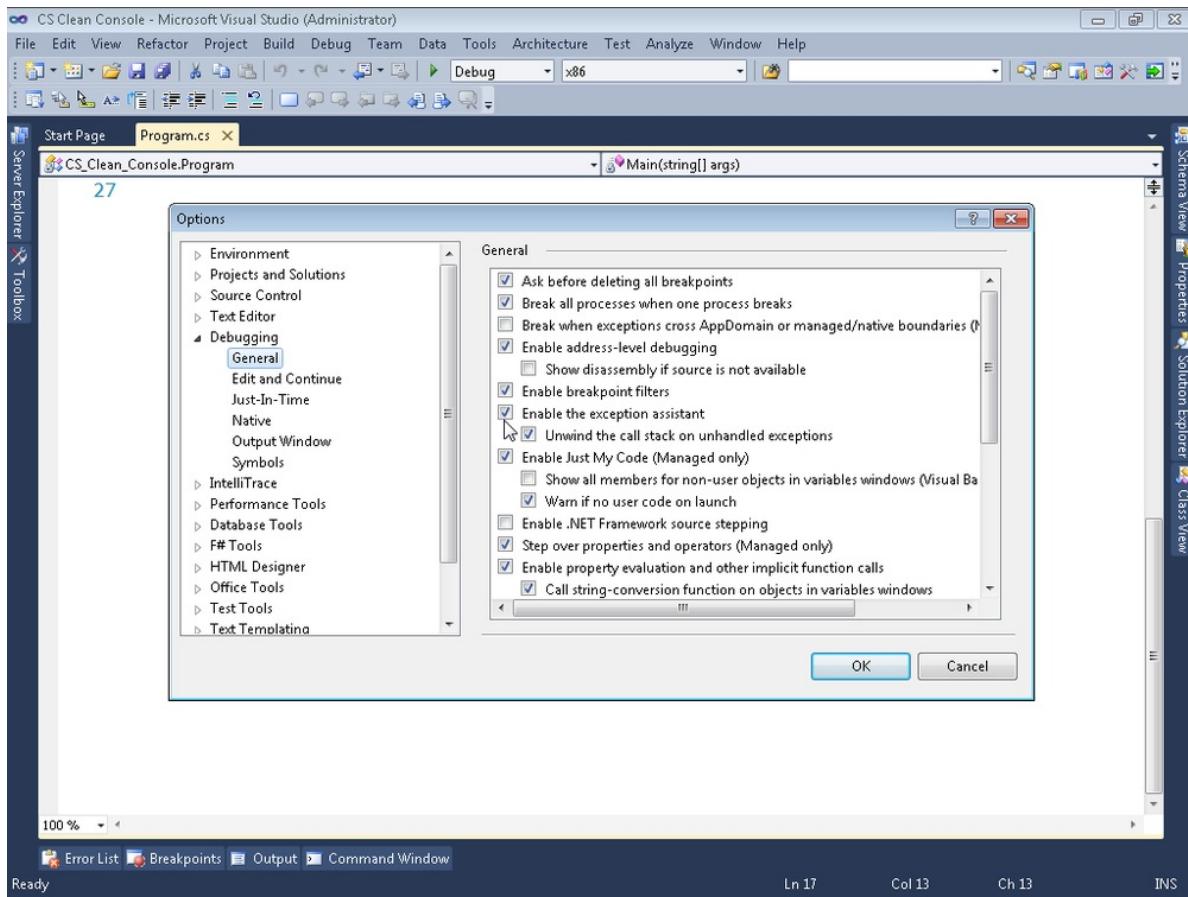


```
Exception Detail.txt - Notepad
File Edit Format View Help

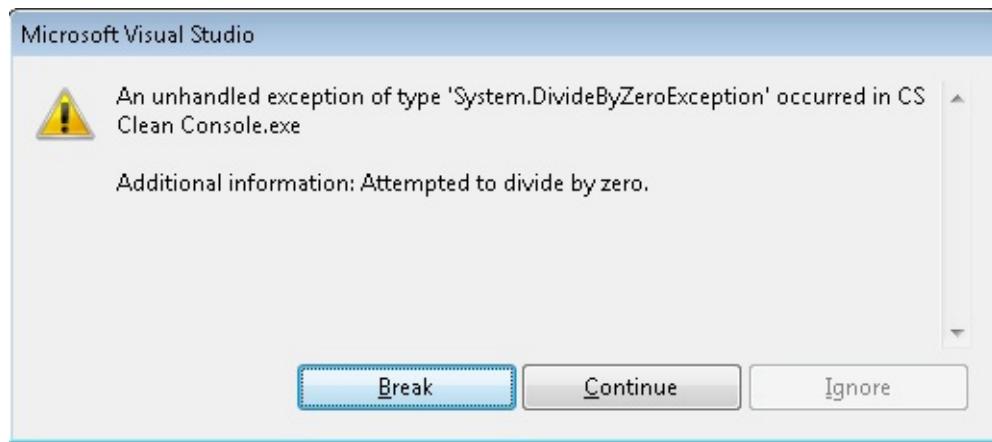
System.DivideByZeroException was unhandled
  Message=Attempted to divide by zero.
  Source=CS Clean Console
  StackTrace:
    at CS_clean_Console.Program.Main(String[] args) in C:\Users\zainnab
\Documents\Visual Studio 2010\Projects\CS Clean Console\CS Clean Console
\Program.cs:line 17
    at System.AppDomain._nExecuteAssembly(RuntimeAssembly assembly, String[]
args)
    at System.AppDomain.ExecuteAssembly(String assemblyFile, Evidence
assemblySecurity, String[] args)
    at Microsoft.VisualStudio.HostingProcess.HostProc.RunUsersAssembly()
    at System.Threading.ThreadHelper.ThreadStart_Context(Object state)
    at System.Threading.ExecutionContext.Run(ExecutionContext executionContext,
ContextCallback callback, Object state, Boolean ignoreSyncCtx)
    at System.Threading.ExecutionContext.Run(ExecutionContext executionContext,
ContextCallback callback, Object state)
    at System.Threading.ThreadHelper.ThreadStart()
InnerException:
```

## Turning Off the Exception Assistant

Although not suggested, you can actually turn this feature off. Just go to Tools | Options | Debugging | General, and clear the Enable The Exception Assistant check box:



The following illustration shows what the same error looks like with the Exception Assistant off:



## Unwind The Call Stack On Unhandled Exceptions

If you want more information about the details of this option, check out the great article by Bill Horst, at <http://blogs.msdn.com/b/vbteam/archive/2008/12/08/did->

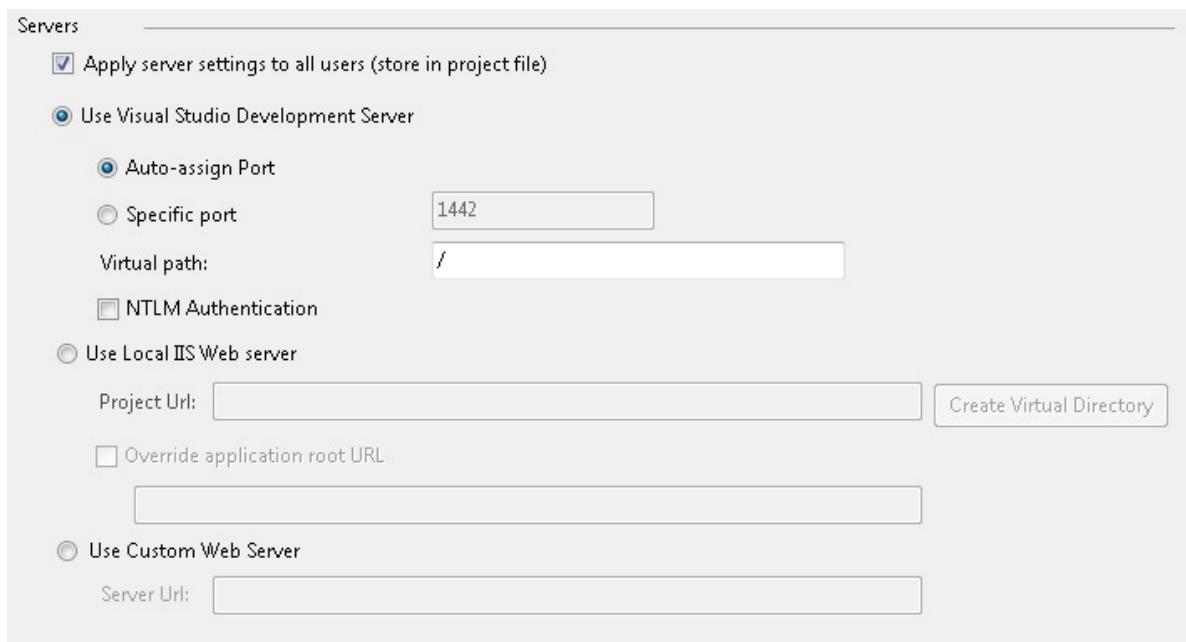
[you-know-you-can-unwind-the-call-stack-from-exceptions-bill-horst.aspx](#).

## 07.29 Use a Specific Port for the Development Server (Web Applications)

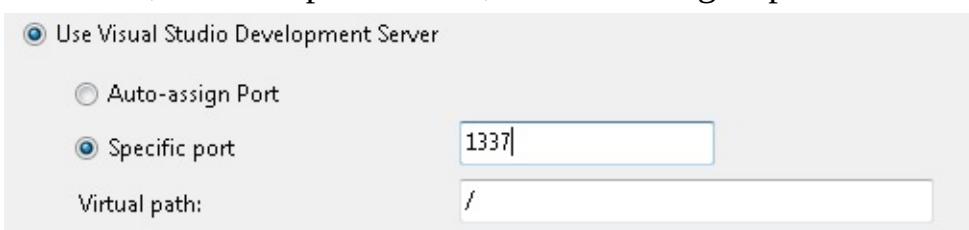
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0029

If you are using a firewall and want to use the same port while you do web development to accommodate a firewall rule, you can configure Visual Studio to use a fixed port. To do this, go to the project properties for any web application and click Web:

On the Web tab, go to the Servers area:



In the Servers area, choose Specific Port, and then assign a port number to use:



The project now continues to use the port number assigned instead of automatically assigning one.

## 07.30 Application and Page Level Tracing

VERSIONS	2005, 2008, 2010
CODE	vstipProj0030

When debugging any project, it's good to have as much information as possible to help deal with any issues. Working with web projects is always a challenge no matter what IDE you use. Fortunately, the folks on the web team have provided a useful tool just for web developers: tracing. Tracing can be enabled at two levels: application and page.

### WARNING

Generally, you should not enable tracing in an production website, because this can display sensitive configuration information to anyone who views pages in the website. Tracing is intended for debugging purposes only. If the localOnly attribute is true, trace information is displayed only for localhost requests. Additionally, if <deployment retail=true> is set in the Web.config file, tracing is disabled.

## Application Level Tracing

You can enable application tracing by going to Web.config and adding the <trace> element inside <system.web>:

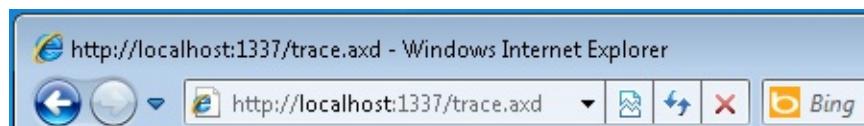
```
<configuration>
  <system.web>

    <trace enabled="true" requestLimit="50" localOnly="true" />

  </system.web>

</configuration>
```

To access the application trace information, simply append “trace.axd” to the root of your website:



The following is a part of what you should see:

Requests to this Application					Remaining: 43
No.	Time of Request	File	Status Code	Verb	
1	6/23/2011 7:02:50 AM	Default.aspx	200	GET	<a href="#">View Details</a>
2	6/23/2011 7:02:51 AM	Styles/Site.css	200	GET	<a href="#">View Details</a>
3	6/23/2011 7:02:51 AM	WebResource.axd	200	GET	<a href="#">View Details</a>
4	6/23/2011 7:02:51 AM	WebResource.axd	200	GET	<a href="#">View Details</a>
5	6/23/2011 7:02:51 AM	favicon.ico	404	GET	<a href="#">View Details</a>
6	6/23/2011 7:02:57 AM	Default.aspx	200	GET	<a href="#">View Details</a>
7	6/23/2011 7:02:57 AM	favicon.ico	404	GET	<a href="#">View Details</a>

## Attributes

Several attributes can be applied when using trace, all of which are optional:

- **enabled** (Boolean) Specifies whether tracing is enabled for an application. The default is false.
- **localOnly** (Boolean) Indicates whether the trace information is available only on the host web server. If false, the trace is visible from any computer, which can be a security issue. The default is true.
- **mostRecent** (Boolean) Shows whether the most recent application-level tracing output is displayed. If beyond the limits that are indicated by the requestLimit attribute, older trace data is discarded. If false, trace data is displayed for requests until the requestLimit attribute is reached. The default is false.
- **pageOutput** (Boolean) Specifies whether trace output is rendered at the end of each page. If false, trace output is accessible through the trace utility only. The default is false.
- **requestLimit** (Int32) Indicates the number of trace requests to store on the server. If the limit is reached and the mostRecent attribute is false, trace is automatically disabled. The maximum request limit is 10,000. If a value that is greater than 10,000 is specified, it is silently rounded down to 10,000 by ASP.NET. The default is 10.
- **traceMode** The order in which to display trace information. The traceMode

attribute can be one of two possible values:

- **SortByCategory** Trace information is displayed alphabetically by user-defined category.
- **SortByTime** Trace information is displayed in the order that the trace information is processed. The default is SortByTime.
- **writeToDiagnosticsTrace** (Boolean) Indicates whether ASP.NET trace messages are forwarded to the System.Diagnostics tracing infrastructure, for any listeners that are registered to display trace messages. The default is false.

You can find more information about trace element settings at <http://msdn.microsoft.com/en-us/library/6915t83k.aspx>.

## Trace Details

Additionally, you can click on the View Details link of each item and see a great deal of information:

The screenshot shows a web browser window with the URL <http://localhost:1337/Trace.axd?id=0>. The page title is "Request Details". Below it is a table titled "Request Details" with the following data:

Session Id:	dow40bst4gddakbbtsz3mkgp	Request Type:	GET
Time of Request:	6/23/2011 7:02:50 AM	Status Code:	200
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

Below this is another table titled "Trace Information" with the following data:

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.006050857562307	0.006051
aspx.page	Begin Init	0.00609448914746816	0.000044
aspx.page	End Init	0.00615022024784207	0.000056
aspx.page	Begin InitComplete	0.00615938654724567	0.000009
aspx.page	End InitComplete	0.00616965280257771	0.000010
aspx.page	Begin PreLoad	0.00617735249407674	0.000008
aspx.page	End PreLoad	0.00619128526917022	0.000014
aspx.page	Begin Load	0.00619971826462153	0.000008
aspx.page	End Load	0.00632767980429584	0.000128
aspx.page	Begin LoadComplete	0.00634014597148474	0.000012
aspx.page	End LoadComplete	0.00634931227088835	0.000009
aspx.page	Begin PreRender	0.00635664531041123	0.000007

## Page Level Tracing

You might not want to have tracing enabled for the entire application for a variety of reasons, such as performance, wanting to focus in on just one page, and so on. Turning on tracing for a page is very simple. Just turn on tracing in your @Page

directive, as in the following example:

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master"
AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="WebApplication61._Default"
Trace="true" %>
```

Now when you view that page, it shows the trace information on the page:

The screenshot shows a Windows Internet Explorer window with the URL <http://localhost:1567/WebForm1.aspx>. The page contains a single button labeled "Click Me". Below the page content, there are two panels: "Request Details" and "Trace Information".

**Request Details**

Session Id:	l3nbsapqinutj1mr3egzcilp	Request Type:	GET
Time of Request:	6/23/2011 7:12:37 AM	Status Code:	200
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

**Trace Information**

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit	1.35661231173338E-05	0.000014
aspx.page	End PreInit	2.49323343778026E-05	0.000011
aspx.page	Begin Init	3.66651976144156E-05	0.000012
aspx.page	End Init	4.54648450418753E-05	0.000009
aspx.page	Begin InitComplete	5.42644924693351E-05	0.000009
aspx.page	End InitComplete	6.23308359445065E-05	0.000008
aspx.page	Begin PreLoad	7.03971794196779E-05	0.000008
aspx.page	End PreLoad	7.88301748709935E-05	0.000008
aspx.page	Begin Load	8.83631262507416E-05	0.000010
aspx.page	End Load	9.71627736782013E-05	0.000009
aspx.page	Begin LoadComplete		

This is equivalent to setting `pageOutput` to true at the application level, but it affects only those pages where you have turned tracing on.

## Combined Tracing

Tracing can get confusing, but the one thing to remember is that the page always wins when tracing is explicitly set. You can use the following table to help keep the tracing results in mind if you set both application and page level tracing.

APPLICATION	PAGE	RESULT FOR A PAGE
true	false	false
false	true	true

## Finally

To say there is a lot going on here would be an understatement. I leave it to you to explore more of the details, which can be found at <http://msdn.microsoft.com/en-us/library/bb386420.aspx>.

## 07.31 The Watch Window: Watching and Changing Values

<b>DEFAULT</b>	Shift+F9 (QuickWatch); Ctrl+Alt+Q (QuickWatch)
<b>VISUAL BASIC 6</b>	Shift+F9 (QuickWatch); Ctrl+Alt+Q (QuickWatch)
<b>VISUAL C# 2005</b>	Shift+F9 (QuickWatch); Ctrl+Alt+Q (QuickWatch); Ctrl+D, Ctrl+Q (QuickWatch); Ctrl+D, Q (QuickWatch)
<b>VISUAL C++ 2</b>	Shift+F9 (QuickWatch); Ctrl+Alt+Q (QuickWatch)
<b>VISUAL C++ 6</b>	Shift+F9 (QuickWatch); Ctrl+Alt+Q (QuickWatch)
<b>VISUAL STUDIO 6</b>	Shift+F9 (QuickWatch); Ctrl+Alt+Q (QuickWatch)
<b>WINDOWS</b>	Alt,D, Q (QuickWatch)
<b>MENU</b>	Debug   QuickWatch; [Context Menu]   Add Watch
<b>COMMAND</b>	Debug.QuickWatch; Debug.AddWatch
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0104

By definition, a watch expression does exactly what it sounds like it should do: It watches something and shows you the result so that you can monitor data as you are debugging. You can use any of the following techniques to create a watch expression:

- Type it in
- QuickWatch
- Add Watch

Let's take a closer look.

### Watch Expressions

First we need to define what you can watch. Essentially, you can watch any valid expression. Some examples of valid watch expressions are shown in the

following illustration:

Name	Value	Type
a	10	Integer
a + c	40	Integer
ShowString() & " added text"	"Cool added text"	String
a+b+c+d+e+f+g+h+i+j+k	550	Integer
AddStuff(a,b,c)	60 [Integer]	Object

## Watch Window

The easiest way to open the main Watch window is to press **Ctrl+Alt+W,1** or go to **Debug | Windows | Watch | Watch 1**. You can have up to four Watch windows, so you can organize your watch expressions into groups if you want:

Name	Value	Type

## Creating a Watch Expression

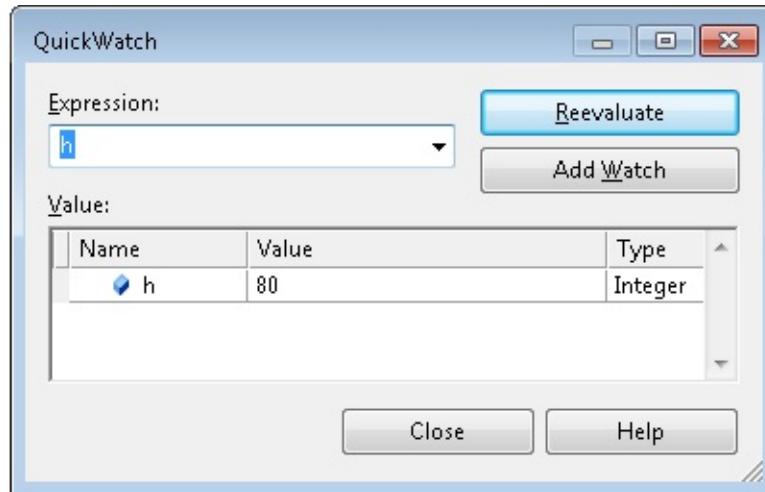
### Type it in

One of the quickest ways to put in most watch expressions is just to type in the variable or expression you want to watch in the Watch window:

Name	Value	Type
c	30	Integer
e		

## QuickWatch (Shift+F9)

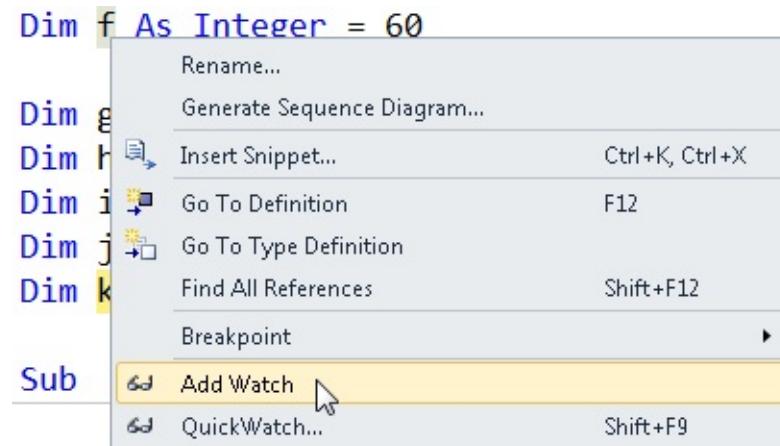
Despite the name, this is actually the longest way to set a watch expression. When you press Shift+F9 or go to Debug | QuickWatch, you get the following dialog box:



Type any expression in the Expression area, and then click Reevaluate to see the value appear in the Value area. If you are happy with what you see and want to add a watch expression to the QuickWatch window, just click Add Watch.

## Add Watch

This is another pretty quick watch expression to set. Just right-click a variable or selected expression and instantly have it added to Watch 1:



## Changing Values

With any watch expression, you can change the value to suit your needs by just selecting the value you want to change:

Name	Value	Type
c	30	Integer
a	10	Integer
b	20	Integer

Then type in a new value, and press Enter:

Name	Value	Type
c	50	Integer
a	10	Integer
b	20	Integer

The value turns red to indicate it has changed. This makes it easy to track your changes.

## 07.32 Understanding QuickWatch

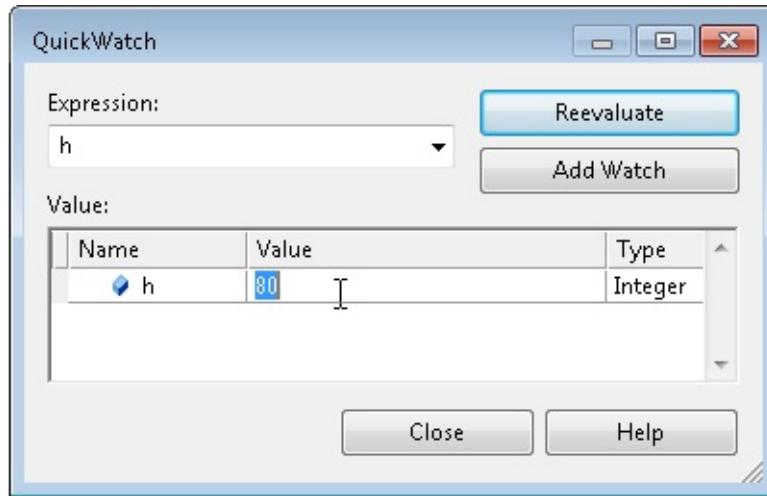
<b>DEFAULT</b>	Shift+F9; Ctrl+Alt+Q
<b>VISUAL BASIC 6</b>	Shift+F9; Ctrl+Alt+Q
<b>VISUAL C# 2005</b>	Shift+F9; Ctrl+Alt+Q; Ctrl+D, Ctrl+Q; Ctrl+D, Q
<b>VISUAL C++ 2</b>	Shift+F9; Ctrl+Alt+Q
<b>VISUAL C++ 6</b>	Shift+F9; Ctrl+Alt+Q
<b>VISUAL STUDIO 6</b>	Shift+F9; Ctrl+Alt+Q
<b>WINDOWS</b>	Alt, D, Q
<b>MENU</b>	Debug   QuickWatch
<b>COMMAND</b>	Debug.QuickWatch
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0108

In vstipTool0104 ([07.31 The Watch Window: Watching and Changing Values](#), page 340), I showed you how to use QuickWatch to get data into a Watch window. From that perspective, using QuickWatch is rather slow and cumbersome. However, QuickWatch doesn't exist for only that reason, so it is worth another look.

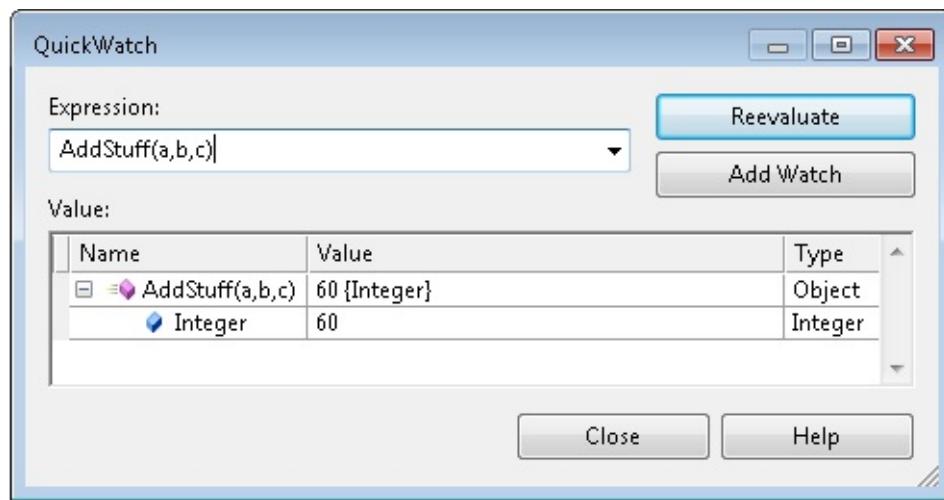
So, why does QuickWatch exist? It was actually created to be a dialog box that is a one-stop shop for quickly working with expressions. Think of it as a dialog box dedicated to a single watch expression. It's a modal dialog box, so you need to close it before moving on with any other debugging. Bring it up by pressing Shift+F9 or going to Debug | QuickWatch on the Menu Bar.

### What Does It Do?

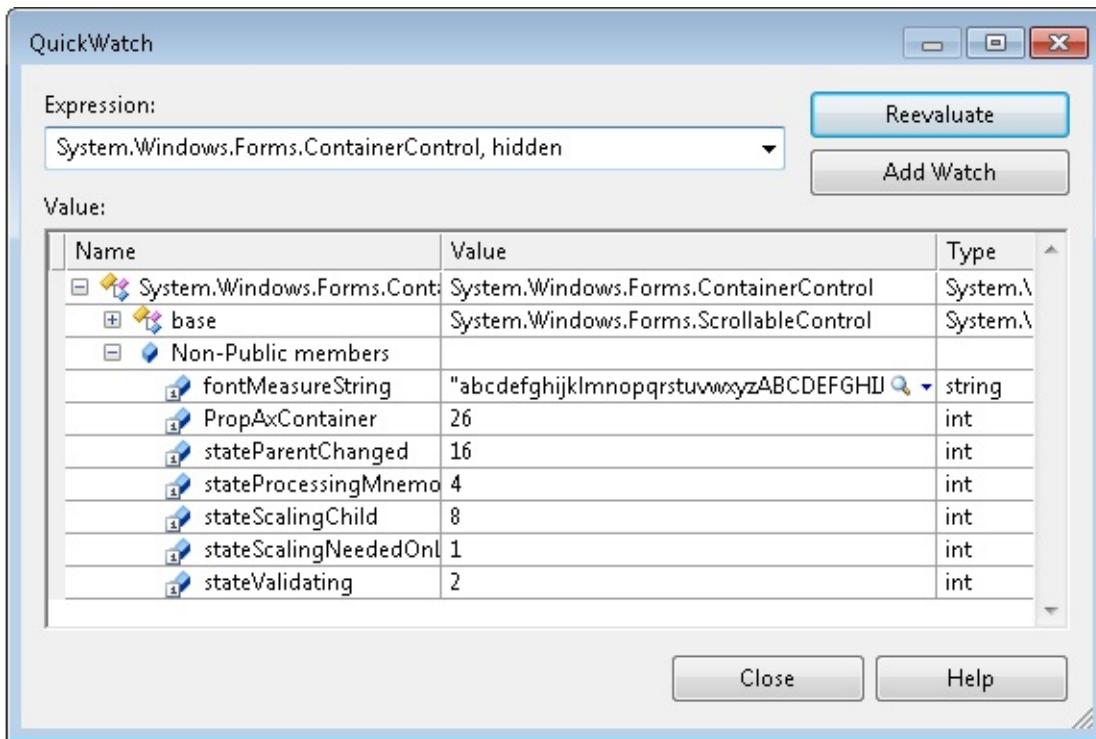
Like a normal watch expression, you can edit the value in it:



Also, you can change the expression and click Reevaluate to see a new value:



One big advantage to the QuickWatch window is the fact that it can be resized. It can be very useful when digging deep:



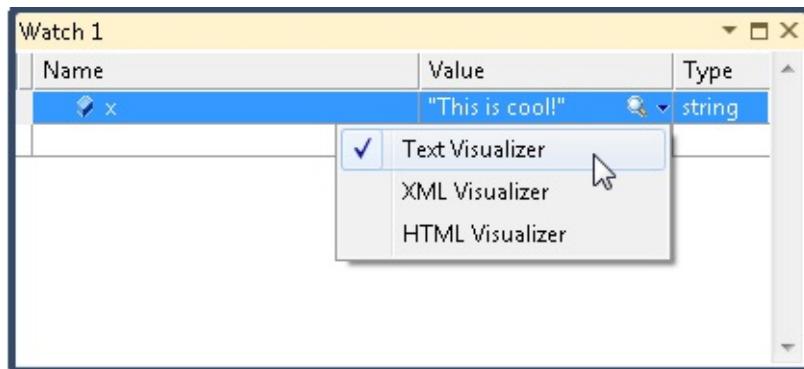
## Other Options

While it is useful for specific scenarios, with the advent of DataTips (see vstipDebug0005, [07.12 Pin a DataTip to Source Code](#), on page 305), the usefulness of the QuickWatch dialog box has diminished somewhat, so you might choose to use DataTips instead. It all comes down to personal preference.

## 07.33 The Watch Window: Visualizers

<b>DEFAULT</b>	Ctrl+Alt+W,1; Ctrl+Alt+W,[2-4]
<b>VISUAL BASIC 6</b>	Ctrl+Alt+W,1; Ctrl+Alt+W,[2-4]
<b>VISUAL C# 2005</b>	Ctrl+Alt+W,1; Ctrl+D, Ctrl+W; Ctrl+D, W; Ctrl+Alt+W,[2-4]
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+Alt+W,1; Ctrl+Alt+W,[2-4]
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+W,1; Ctrl+Alt+W,[2-4]
<b>WINDOWS</b>	Alt+D, W, W, [1-4]
<b>MENU</b>	Debug   Windows   Watch   Watch [1,2,3,4]
<b>COMMAND</b>	Debug.Watch[1,2,3,4]
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0106

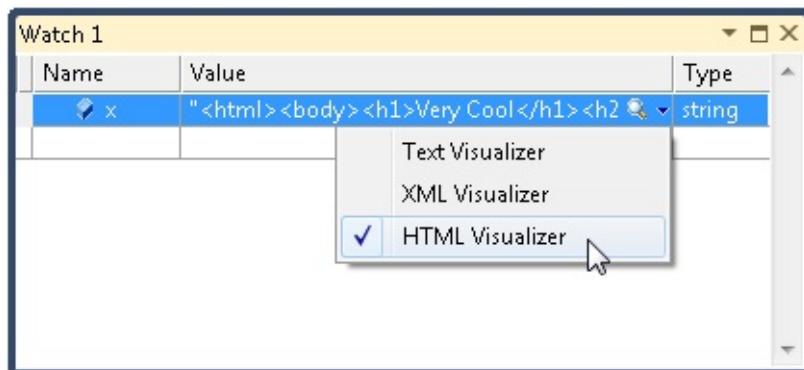
When using the Watch windows, you might come across visualizers. A visualizer is a debugger component that enables the debugger to display (visualize) the contents of a data object in a meaningful, understandable form. Visualizers are pretty easy to spot because they have a magnifying glass icon:



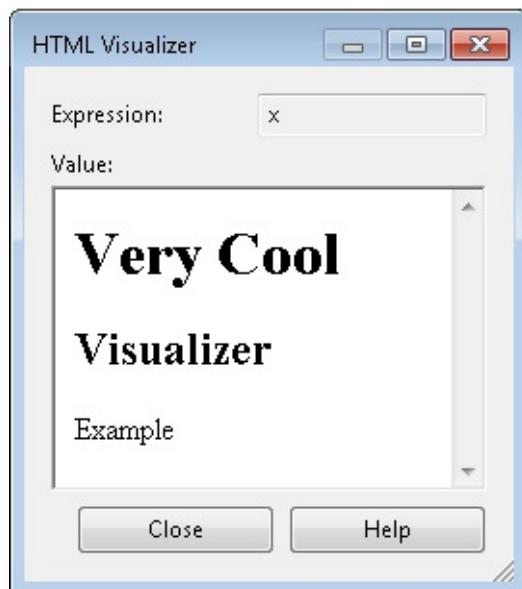
Notice that when you click on the magnifying glass, it gives you options (based on the data you are looking at) for a visualizer. If you select Text Visualizer, you get the following dialog box:



You might find it useful to leverage the power of visualizers when looking at different types of data. If this output were HTML, you would use the HTML Visualizer:



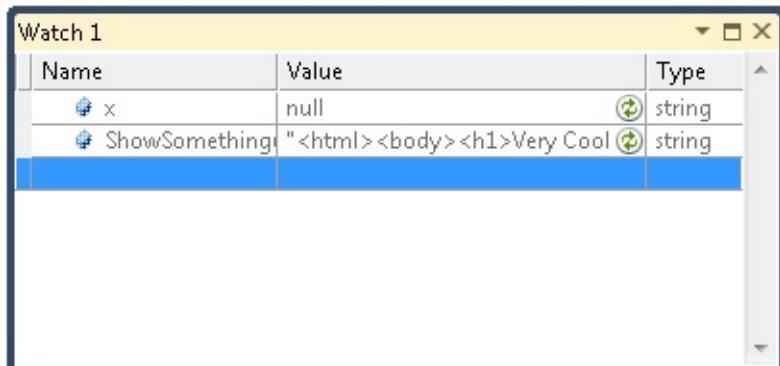
And you would see the following dialog box:



## 07.34 The Watch Window: Refreshing Data

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Debugging   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0107

When you evaluate an expression in one of the Watch windows, one of two refresh icons might appear in the Value column. One refresh icon is a circle that contains two wavy lines that resemble threads. The other icon is two circling arrows, as shown in the following example:



## Refresh Icons

Following is what the documentation (<http://msdn.microsoft.com/en-us/library/z4ecfxd9.aspx>) has to say about these icons.

### Circling arrows

If the circling arrows appear, the expression was not evaluated for one of the following reasons:

- An error occurred as the expression was being evaluated. For example, a timeout might have occurred, or a variable might have been out of scope.
- Evaluating the expression would have required evaluating a property or making an implicit function code. Evaluation of properties and implicit function calls can have side effects that affect the state of your program. Because these effects can make debugging more difficult, automatic evaluation

of properties and implicit function calls by the debugger is often turned off. Occasionally, a programmer might unintentionally turn off automatic evaluation.

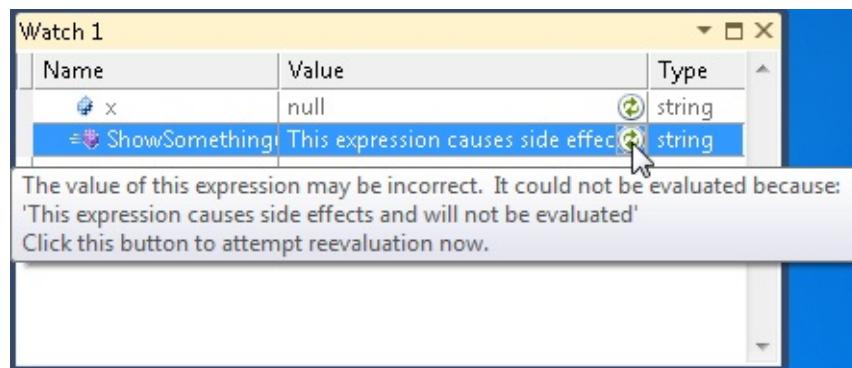
If you want to learn more about side effects, see the topic “Side Effects and Expressions” at <http://msdn.microsoft.com/en-us/library/a7a250bs.aspx>.

## Two threads

If the two threads appear, the expression was not evaluated because of a potential cross-thread dependency. A cross-thread dependency means that evaluating the code requires other threads in your application to run temporarily. When you are in break mode, all threads in your application are typically stopped. Allowing other threads to run temporarily can have unexpected effects on the state of your program and causes the debugger to ignore events such as breakpoints.

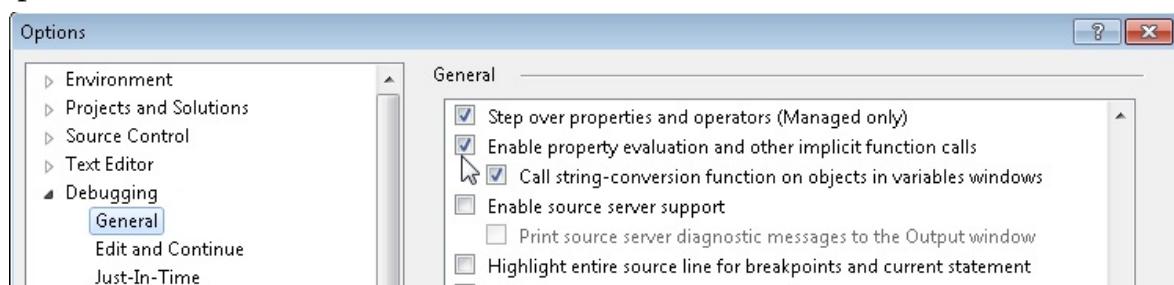
## Refreshing the data

To refresh the data, just click the icon or press the Spacebar:



## Turning It Off

Although not suggested, you can turn this feature off by going to Tools | Options | Debugging | General and clearing the Enable Property Evaluation And Other Implicit Function Calls check box:

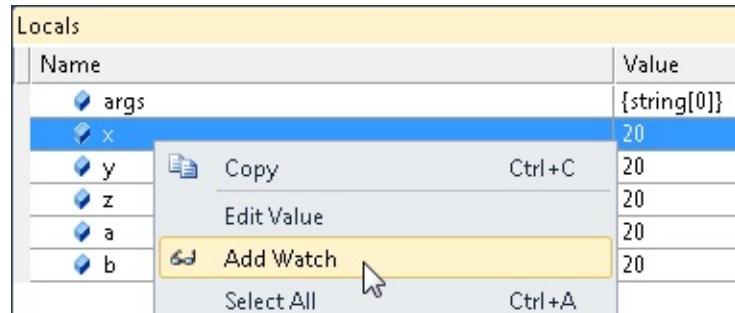


## 07.35 The Watch Window: Adding Watches from Variable Windows

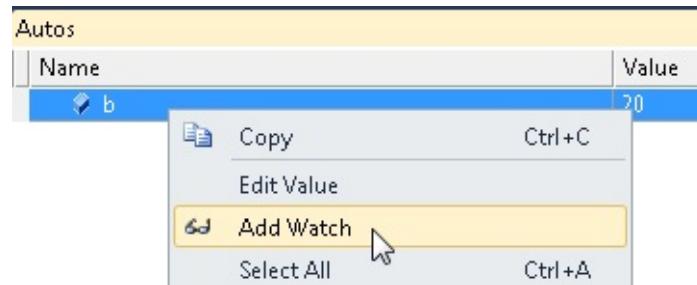
<b>MENU</b>	[Context Menu]   Add Watch
<b>COMMAND</b>	Debug.AddWatch
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0109

The Locals, Autos, Watch, and QuickWatch windows are all known, collectively, as the Variable windows. Did you know that every Variable window supports the Debug.AddWatch command? This tip provides some examples.

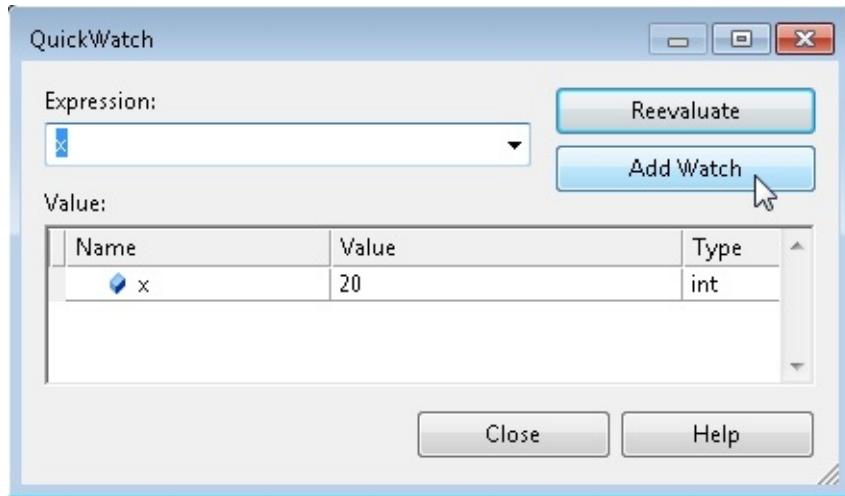
### Locals Window



### Autos Window

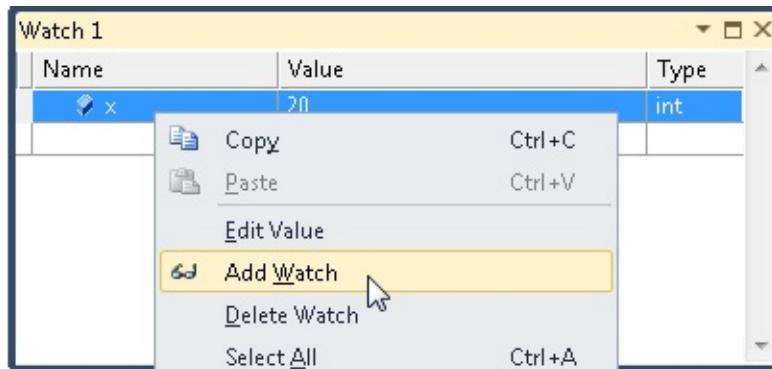


### QuickWatch



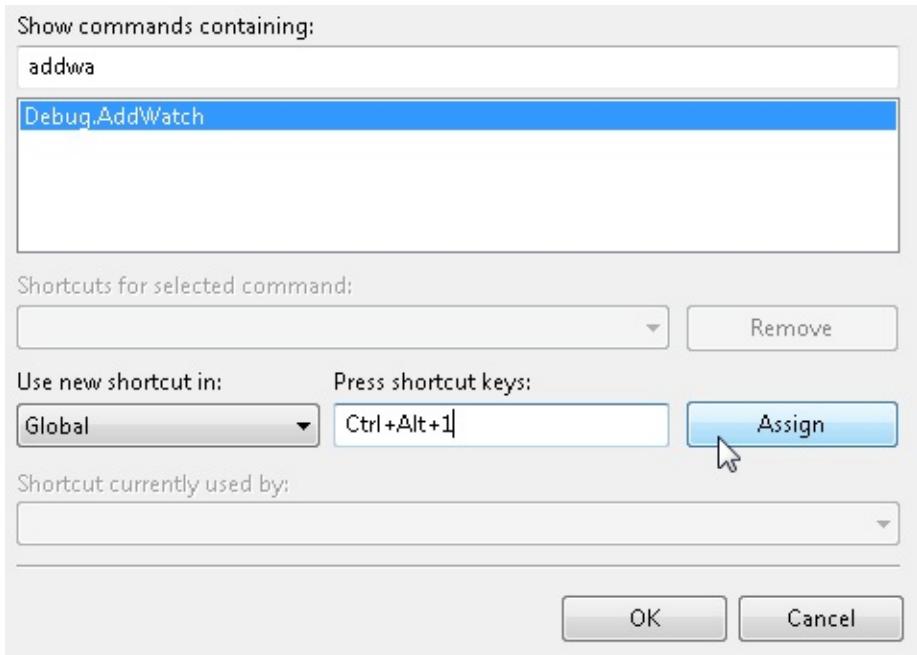
## Watch [1, 2, 3, 4] Window

This is one you might not expect. The Watch windows actually have the ability to add watch expressions. Very useful if you have a complex expression and want to copy it to do a modification:



## Keyboard Mapping

If you find yourself adding watch expressions frequently in break mode, you should consider mapping a keyboard shortcut. Go to Tools | Options | Environment | Keyboard, and assign a shortcut to the Debug.AddWatch command:



Use new shortcut in:      Press shortcut keys:

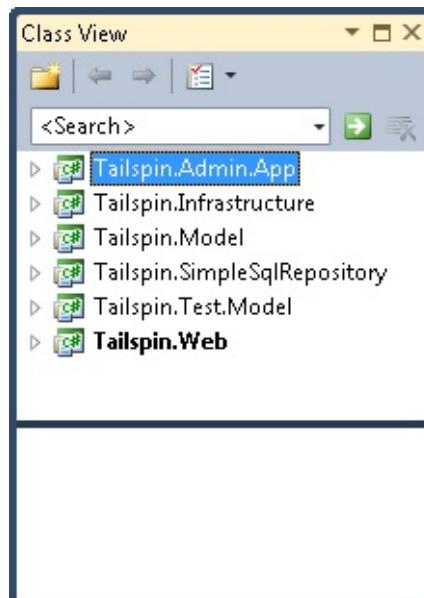


Shortcut currently used by:

## 07.36 Create Folders in Class View

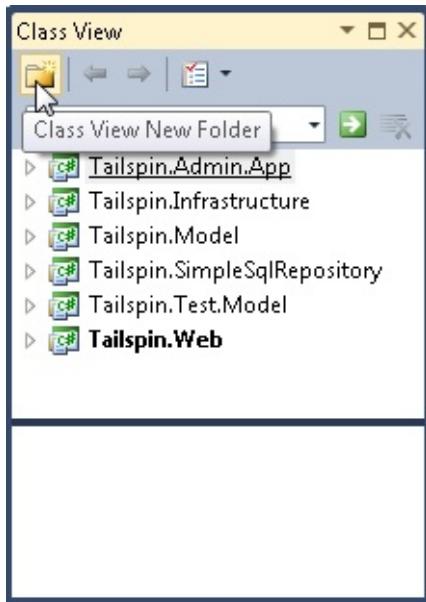
<b>MENU</b>	[Context Menu]   New Folder
<b>COMMAND</b>	View.ClassViewNewFolder
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0072

Did you know that you can create folders to easily organize items in Class View? It's pretty easy to do. Just open the Class View dialog box (Ctrl+Shift+C):

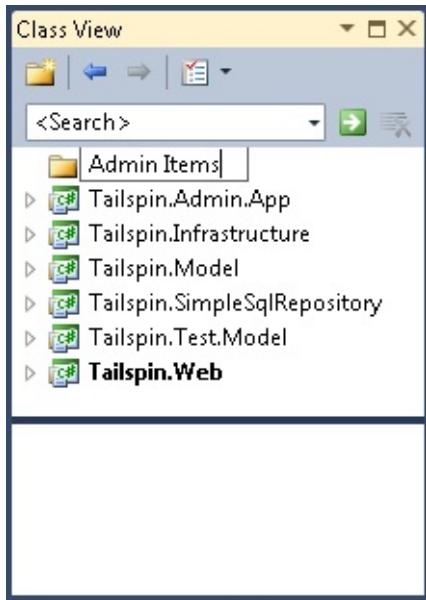


### Create a New Folder

Click the Class View New Folder button:



Give your new folder some logical name:

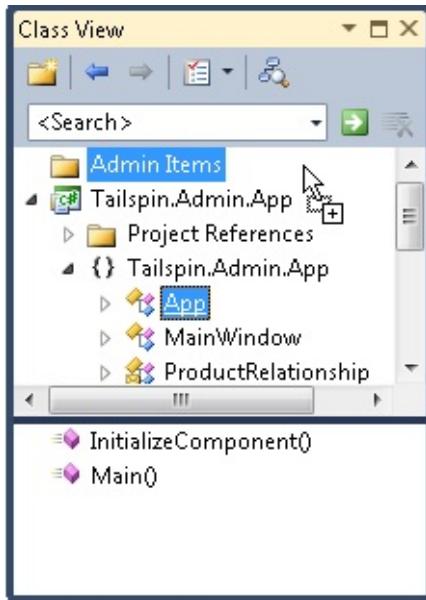


#### NOTE

These folders are not created on the file system but are stored in your .suo file.

## Putting Items into Your Folder

You can simply click and drag items into your new folder to organize them:



You can also copy and paste items into the folders as well. These operations are strictly organizational and don't actually make an extra copy of the item in your code.

## Removing Items from Folders

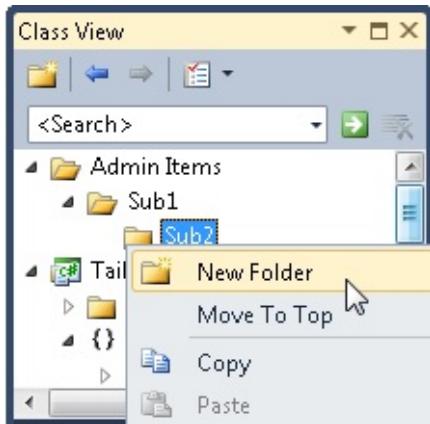
You can remove items from your folder at any time by deleting them (right-click or press the Delete key).

### NOTE

This deletes only the item from the Class View and doesn't actually delete your code.

## Creating Subfolders

You can even nest the folder structure by creating a new folder inside an existing one:



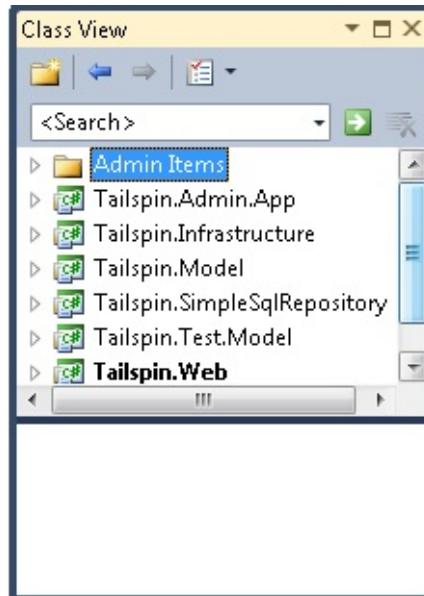
## Deleting Folders

Naturally you can delete any folder by right-clicking it and choosing Delete.

## 07.37 Search in Class View

<b>DEFAULT</b>	Ctrl+K, Ctrl+V
<b>VISUAL C++ 2</b>	[no shortcut]
<b>COMMAND</b>	View.ClassViewGoToSearchCombo; View.ClassViewSearch
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0073

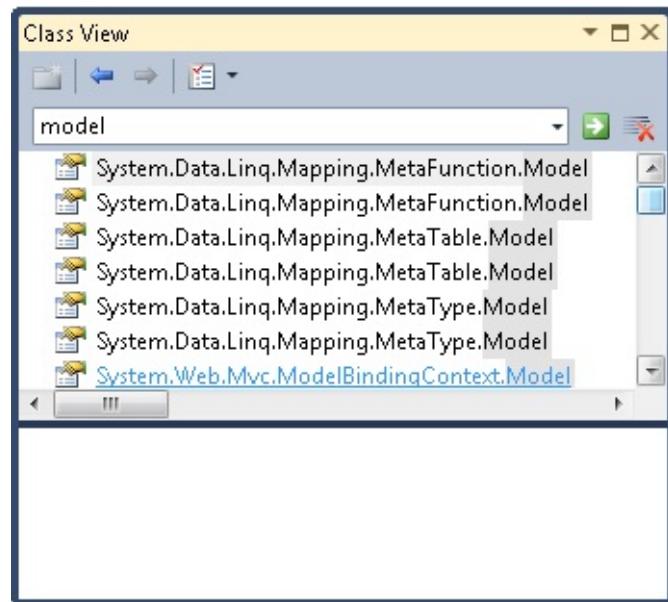
The Class View window has a search capability that finds items quickly. Just go to the Class View Search Combo box (Ctrl+K, Ctrl+V):



### WARNING

I've actually had the search combo box disappear on me before. Restarting Visual Studio should bring it back if this happens to you, too.

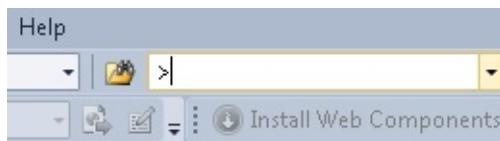
Type in the keyword to search for, and press Enter:



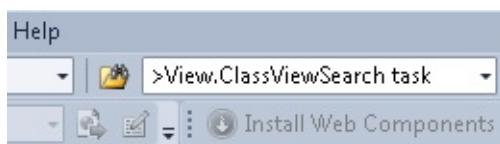
It searches for the keyword in items and shows the results while highlighting the last instance where the keyword was found in an item. The search is a contains operation, so it looks for anything that has the keyword anywhere in the result.

## View.ClassViewSearch Command

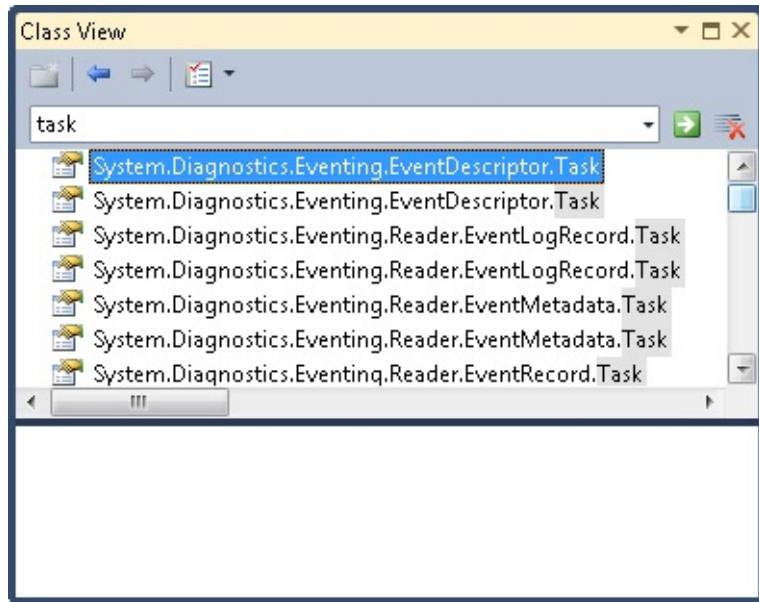
If you want a quick way to do this by running a command, you can use `View.ClassViewSearch` with any keyword(s). First, go someplace you can run a command. I prefer to use the Find Combo box with the command character (`Ctrl+U`) to run my commands (see vtipTool0070, [03.28 Understanding Commands: Logging Commands](#) on page 121, for more options):



Type in **View.ClassViewSearch [keyword(s)]**. In this case, let's look for the word "task":

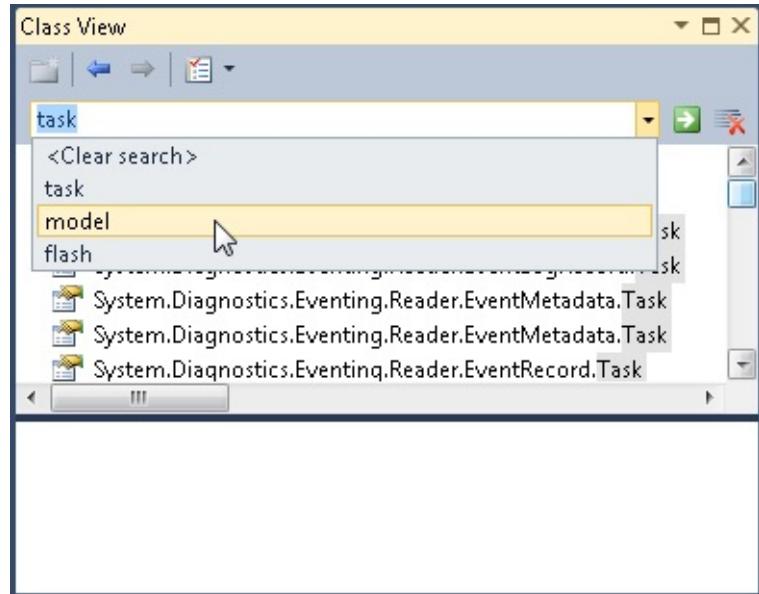


Just press Enter, and we get our results in the Class View:



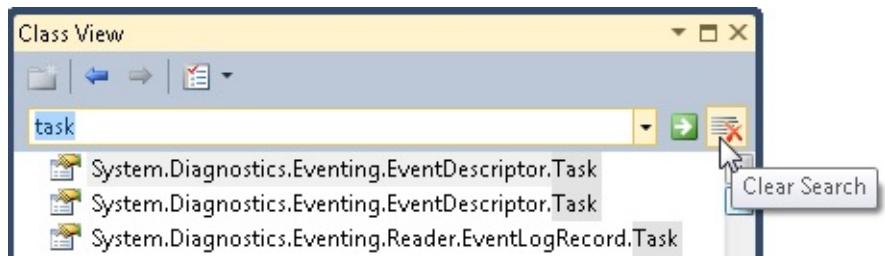
## Use a Previous Search

You can repeat any previous search in Class View by using the drop-down list from the Search Combo box:



## Clear Your Search

Whichever method you use, you can always clear your search by clicking the Clear Search button:

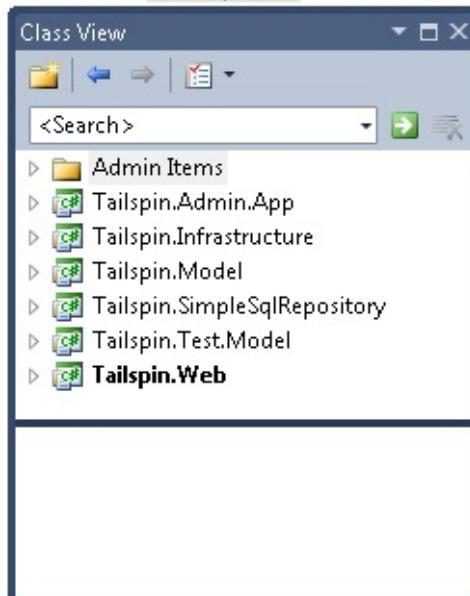


## 07.38 Synchronize Your Class View

<b>COMMAND</b>	View.SynchronizeClassView
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0074

When you are in your code, sometimes you would like to have Class View synchronize with your code. By default, it doesn't do this.

`public Customer(string u:`



You can run the `View.SynchronizeClassView` command from the Find combo (Ctrl+/) to see how this works:

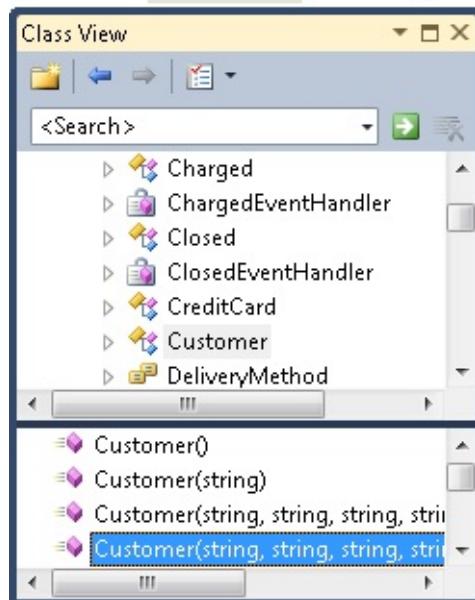


### NOTE

This command works only if the editor has the focus, so you need to run it from the Find drop-down box as shown in this example or assign a shortcut key to it. It does not work if you try to run it from the Command window.

It finds your current location in Class View:

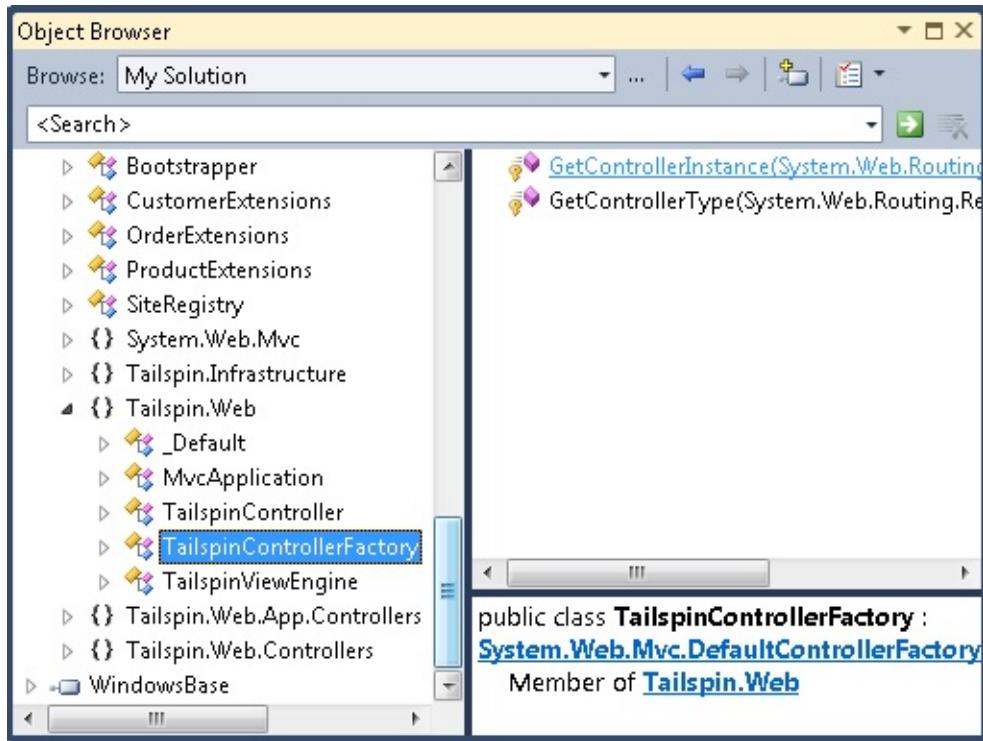
```
public Customer(string u:
```



## 07.39 The Misnamed and Misunderstood Object Browser

<b>DEFAULT</b>	Ctrl+Alt+J
<b>VISUAL BASIC 6</b>	Ctrl+Alt+J; F2
<b>VISUAL C# 2005</b>	Ctrl+Alt+J; Ctrl+W, Ctrl+J; Ctrl+W, J
<b>VISUAL C++ 2</b>	Ctrl+Alt+J; Shift+Alt+F1
<b>VISUAL C++ 6</b>	Ctrl+Alt+J
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+B; F2
<b>WINDOWS</b>	Alt,V, J
<b>MENU</b>	View   Object Browser
<b>COMMAND</b>	View.ObjectBrowser
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0077

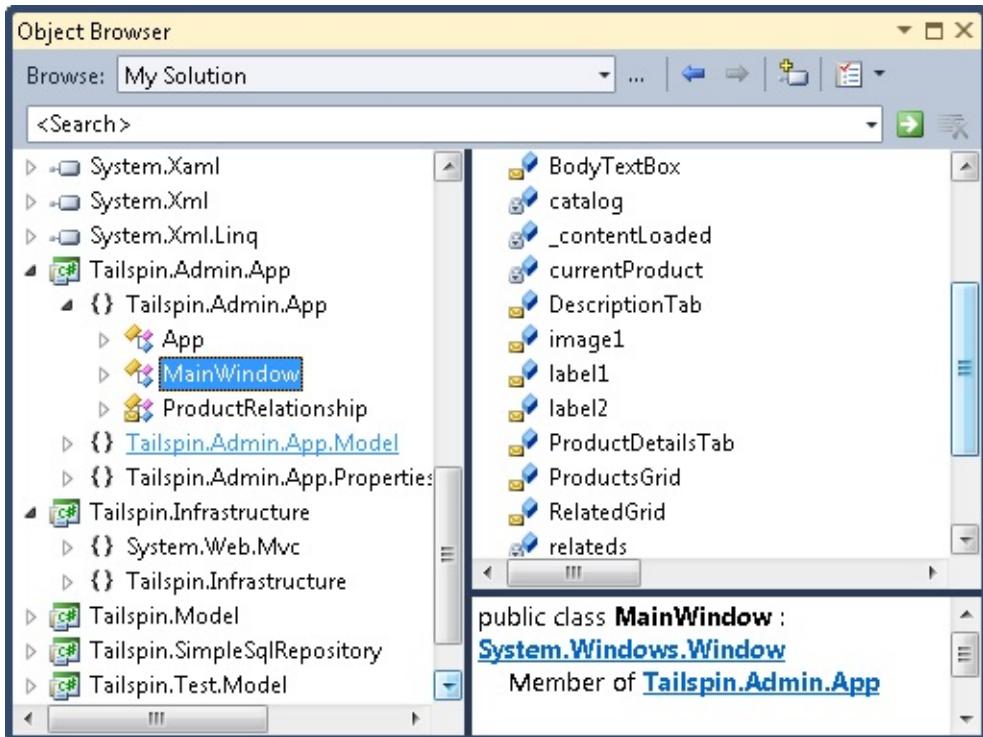
Many people often ask why they should use the Object Browser. First and foremost, it is a way to browse to find the correct class, object, property, method, interface, and so on that is useful for your needs as you code. But the Object Browser does much, much more than its name implies.



As you can see from the high-level view shown in the preceding illustration, the Object Browser is composed of several parts:

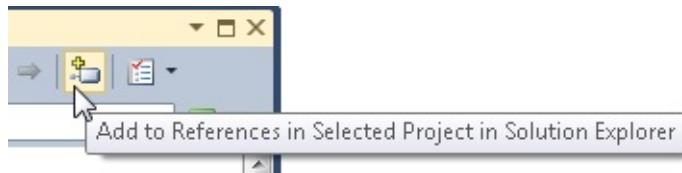
- **Toolbar** (very top) Contains various controls for manipulating the various functions available.
- **Objects pane** (left, top and bottom) Displays an expandable list of symbols whose top-level nodes represent components or namespaces available in the current browsing scope.
- **Members pane** (right, top) Displays the available members, if available, of any symbol selected in the Objects pane.
- **Description pane** (right, bottom) Displays detailed information about the currently selected object or member.

First, the name would have you think that it shows only objects. This is simply untrue. In fact, it shows many different types of symbols, which is better than just showing objects:



In this example, as shown in the preceding illustration, we see a couple of projects with several namespaces in them, in addition to some interfaces and classes. The classes with envelopes represent internal classes. The icons are explained in vstipTool0076, [AX.135 Creating a Class Diagram from Class View](#), in Appendix B (<http://go.microsoft.com/fwlink/?LinkId=223758>).

Second, you might think that all you can do is browse items. Again, not true. For example, you can add references to your project from the Object Browser when you find something that you want to include:

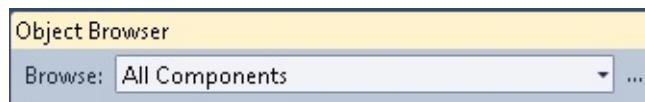


The next series of tips explore, in detail, the use of the Object Browser in your work.

## 07.40 The Object Browser: Setting the Browsing Scope

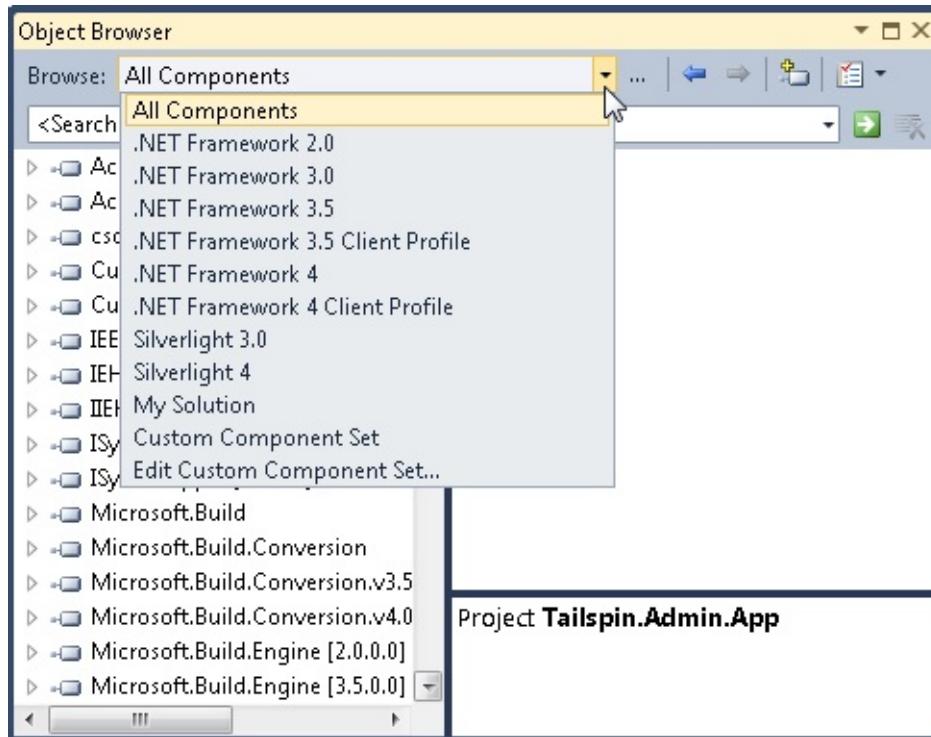
<b>DEFAULT</b>	Ctrl+Alt+J
<b>VISUAL BASIC 6</b>	Ctrl+Alt+J; F2
<b>VISUAL C# 2005</b>	Ctrl+Alt+J; Ctrl+W, Ctrl+J; Ctrl+W, J
<b>VISUAL C++ 2</b>	Ctrl+Alt+J; Shift+Alt+F1
<b>VISUAL C++ 6</b>	Ctrl+Alt+J
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+B; F2
<b>WINDOWS</b>	Alt, V, J
<b>MENU</b>	View   Object Browser
<b>COMMAND</b>	View.ObjectBrowser
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0078

When you use the Object Browser, it's a good idea to know what components you are browsing. So let's start our examination at the top, with the Browse area:



### Browse

The Browse drop-down box lets you specify the browsing scope (the list of items you see in the Objects pane):



## All Components

The All Components drop-down quite literally lets you browse all components, including all of the .NET Framework, the current solution and its referenced components, and any other components that you have added by selecting Edit Custom Component Set. It's a massive dose of information overload, so be prepared.

## .NET Framework X / Silverlight X

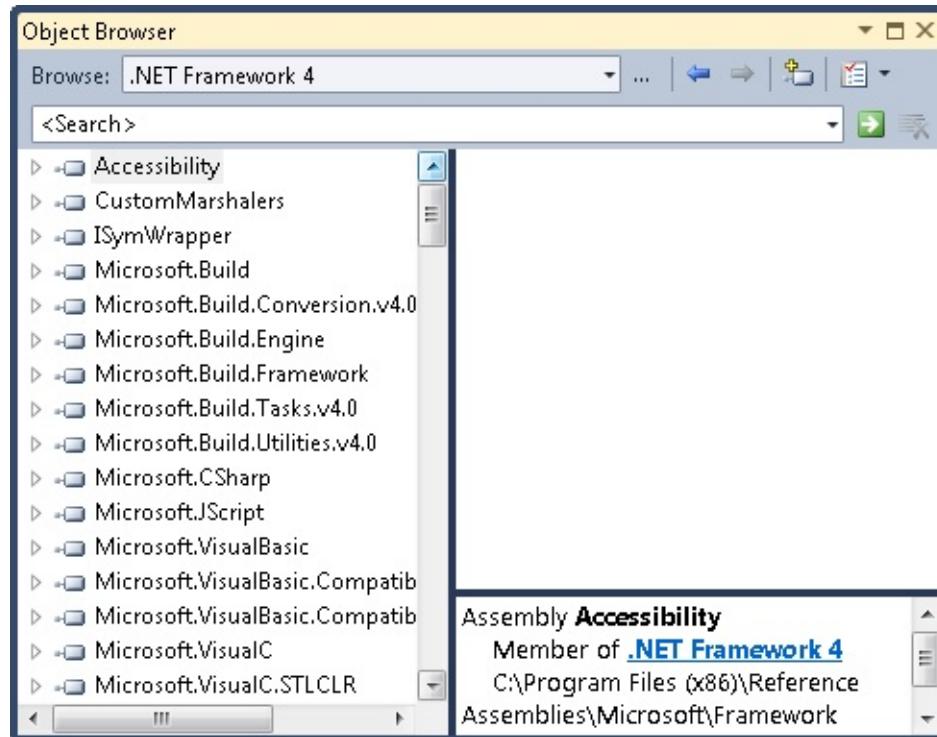
Sets the browsing scope to a specific version of the Framework.

## My Solution

Shows items in the current solution and referenced components.

## Custom Component Set

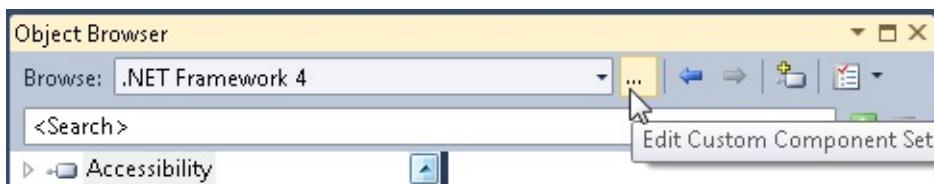
Sets browsing scope to the list of components identified by editing the Custom Component Set. For example, if I choose NET Framework 4, I see the following:



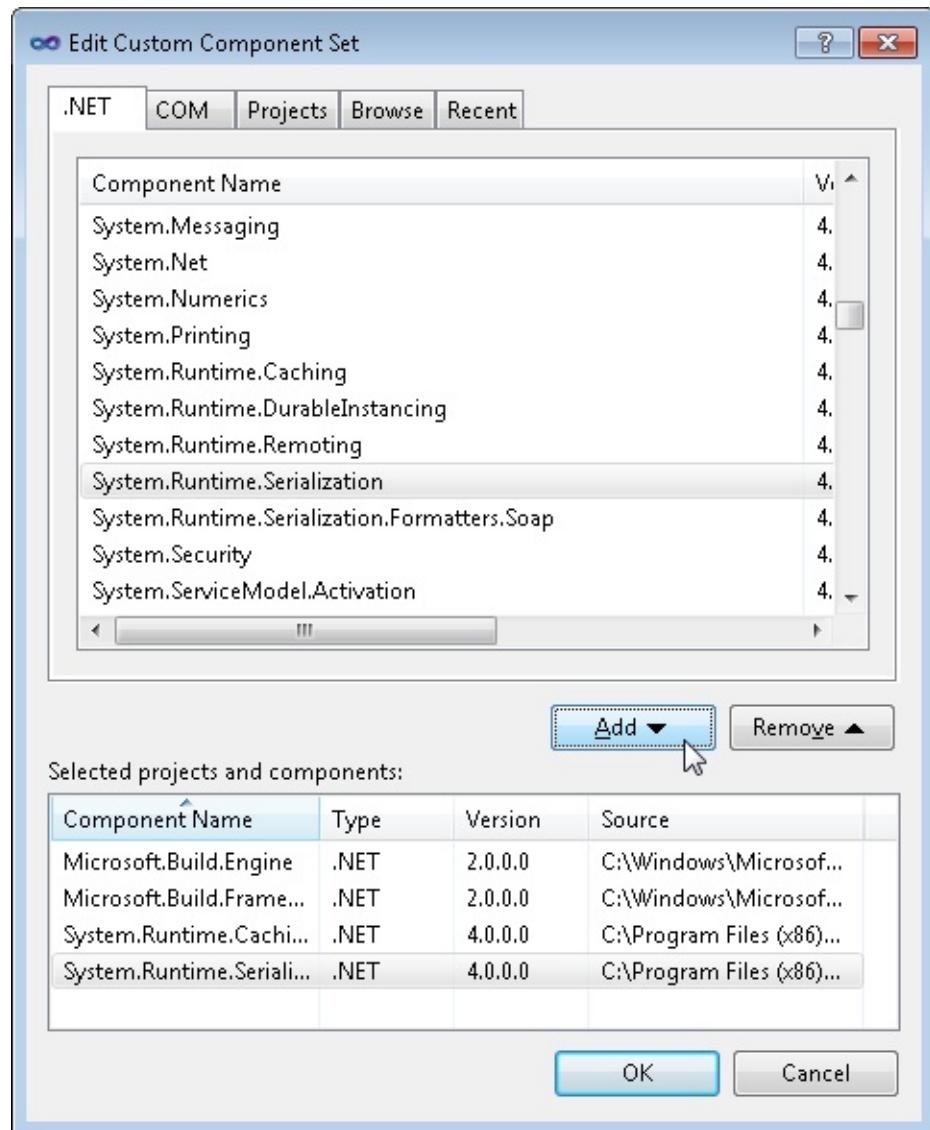
## Edit Custom Component Set

You can edit the Custom Component Set to identify the components that create the browsing scope in two ways:

- Choose Edit Custom Component Set in the browse drop-down box.
- Click the **Edit Custom Component Set** button on the toolbar:

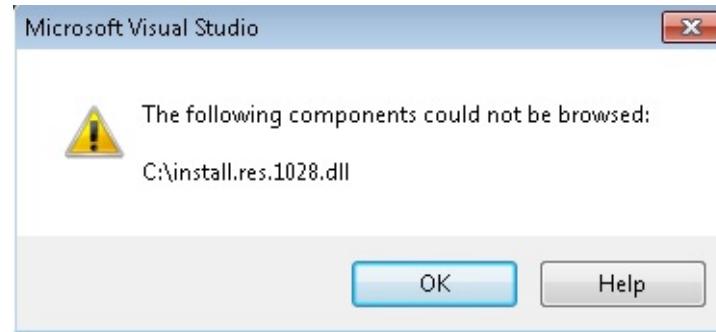


Regardless of the method used, you get the Edit Custom Component Set dialog box:

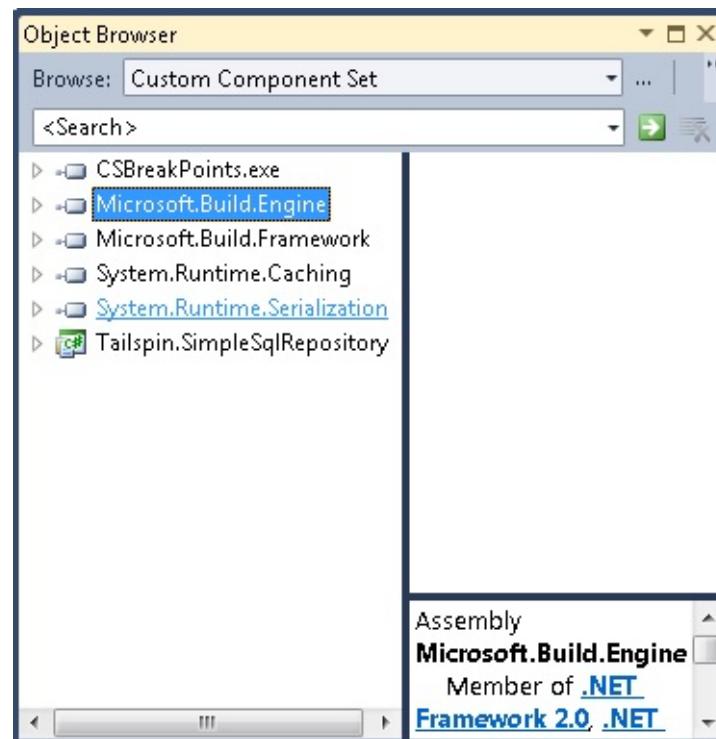


In the Edit Custom Component Set dialog box, you can add or remove components from a variety of sources. The browsing scope of the Custom Component Set is determined by the items listed in the Selected Projects And Components section of this dialog box. As you can see, information about the type, version, and source (location) is available here.

If you pick something that can't be browsed, you get the following message:



After you have successfully edited the Custom Component Set, it is automatically selected in the Browse drop-down box and becomes the current browsing scope:



## 07.41 The Object Browser: Navigation and References

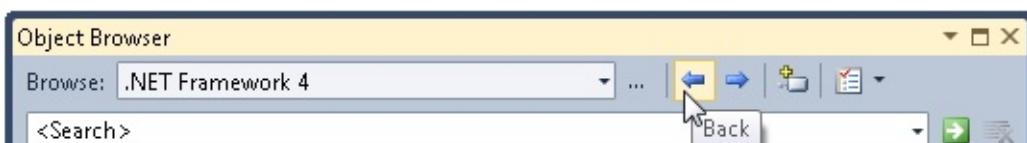
<b>DEFAULT</b>	Ctrl+Alt+J (view object browser); Alt+Right Arrow (forward); Alt+Left Arrow (back)
<b>VISUAL BASIC 6</b>	Ctrl+Alt+J (view object browser); F2 (view object browser); Alt+Right Arrow (forward); Alt+Left Arrow (back)
<b>VISUAL C# 2005</b>	Ctrl+Alt+J (view object browser); Ctrl+W, Ctrl+J (view object browser); Ctrl+W, J (view object browser); Alt+Right Arrow (forward); Alt+Left Arrow (back)
<b>VISUAL C++ 2</b>	Ctrl+Alt+J (view object browser); Shift+Alt+F1 (view object browser); Alt+Right Arrow (forward); Alt+Left Arrow (back)
<b>VISUAL C++ 6</b>	Ctrl+Alt+J (view object browser); Alt+Right Arrow (forward); Alt+Left Arrow (back)
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+B (view object browser); F2 (view object browser); Alt+Right Arrow (forward); Alt+Left Arrow (back)
<b>WINDOWS</b>	Alt,V, J (view object browser)
<b>MENU</b>	View   Object Browser
<b>COMMAND</b>	View.ObjectBrowser; View.Forward; View.Backward; View.ObjectBrowserAddReference
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0079

Continuing our look at the Object Browser, we now turn our attention to the toolbar buttons to the right of the Browse area:



## Navigation

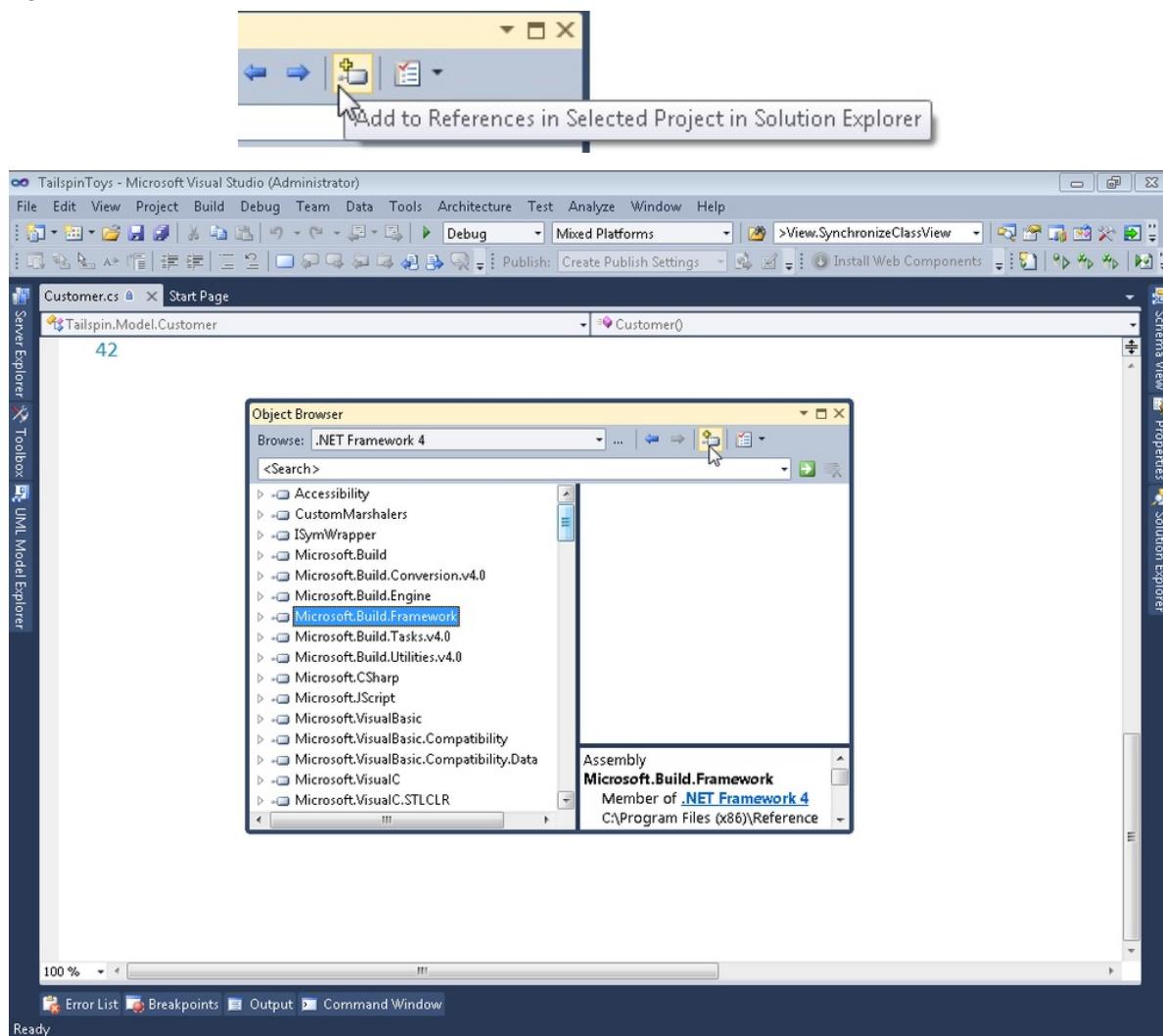
The Forward (Alt+Right Arrow) and Back (Alt+Left Arrow) buttons can be used to easily navigate among items in the Objects pane as you browse:



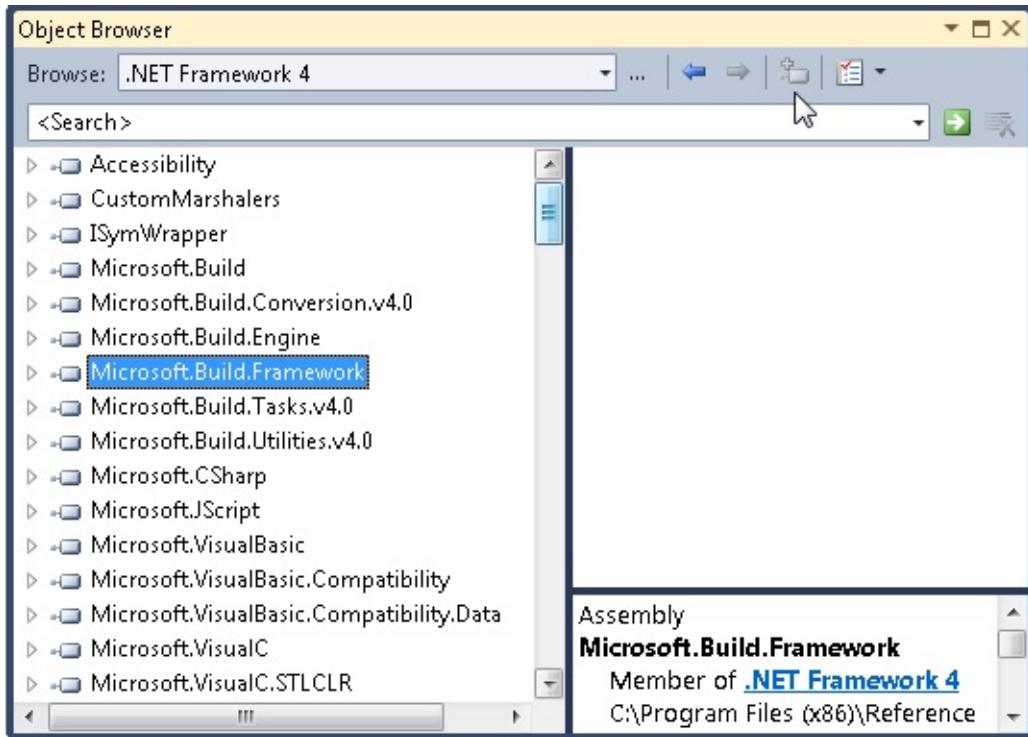
I haven't discovered any obvious upper limit to how many times you can go back. I have personally tested it to go back over 75 items, with no end in sight. The places are remembered (and more added) as long as Visual Studio remains open, regardless of which project or solution you are using.

## References

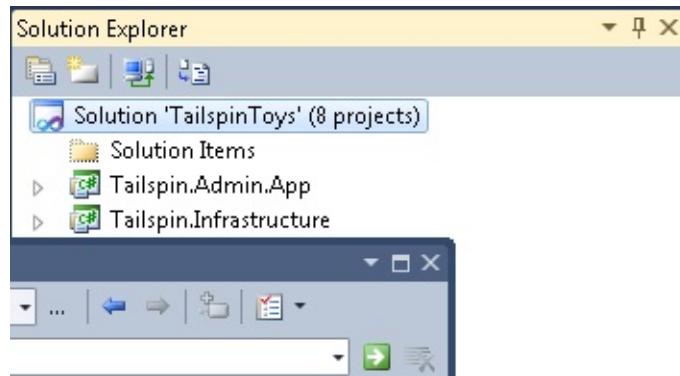
You can add a reference to the currently selected item in the Objects pane to your project:



Read the tooltip carefully. Notice that it says *Add To References In Selected Project In Solution Explorer*. This might confuse you when you try to use it for the first time because you can select a reference that you want to add and then find the button disabled (top right in the following illustration):



This is because you most likely don't have a project selected in Solution Explorer, as we do in the following example:



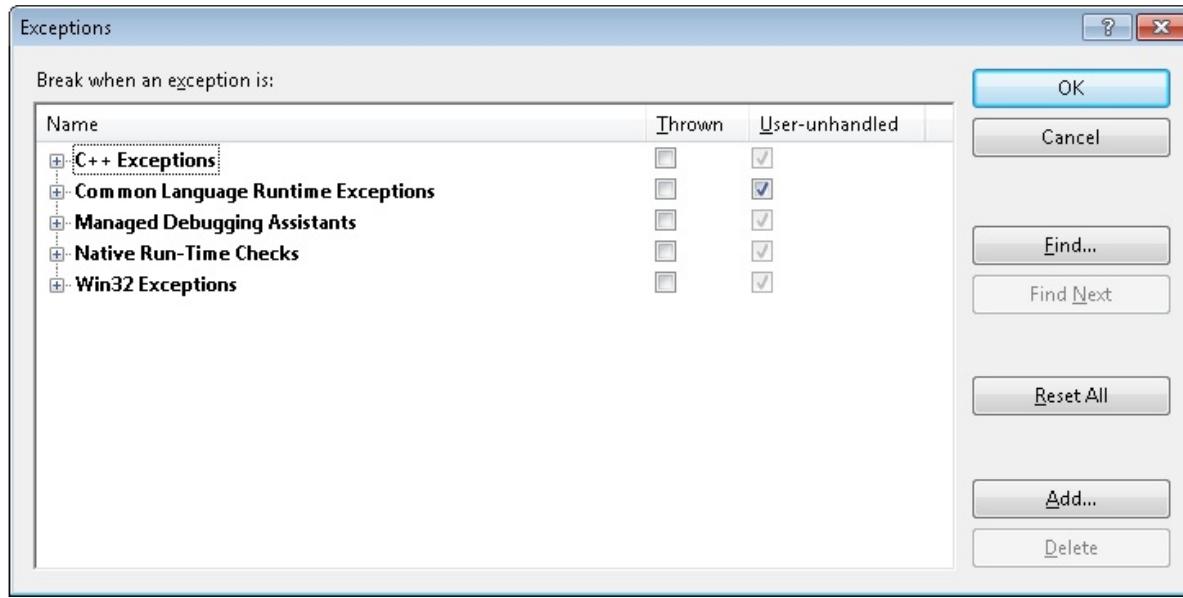
Simply click on (or in) any project, and the button becomes enabled so that you can add the reference to your project.

## 07.42 The Exceptions Dialog Box

<b>DEFAULT</b>	Ctrl+Alt+E
<b>VISUAL BASIC 6</b>	Ctrl+Alt+E
<b>VISUAL C# 2005</b>	Ctrl+Alt+E; Ctrl+D, Ctrl+E; Ctrl+D, E
<b>VISUAL C++ 2</b>	Ctrl+Alt+E
<b>VISUAL C++ 6</b>	Ctrl+Alt+E
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+E
<b>WINDOWS</b>	Alt,D, X
<b>MENU</b>	Debug   Exceptions
<b>COMMAND</b>	Debug.Exceptions
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0039

By default, Visual Studio breaks only when an exception is not handled by user code. This can sometimes be problematic because the exception can bubble up through several calls far away from where the actual exception originally occurred, making it harder to find the problem.

You can use the Exceptions dialog box (Ctrl+Alt+E) to configure exceptions to break when they happen rather than checking to see whether they are unhandled:



The exceptions are divided up into five broad categories. Also, as shown in the preceding illustration, two options are provided: Thrown and User-Unhandled. When you check Thrown for any category or individual exception, it breaks when the exception occurs instead of waiting to see whether the user handles the exception.

Additionally, you can use the following buttons.

## Find

### Find

Helps you search for a specific exception.

### Reset All

Puts exceptions back to their default settings.

### Add

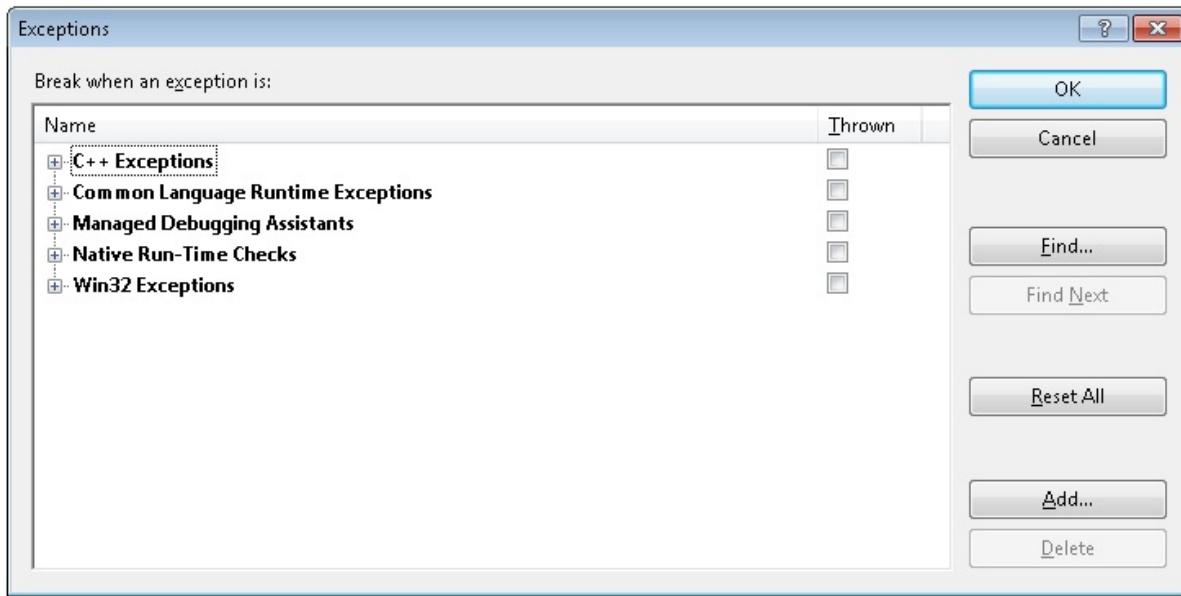
Allows you to add exceptions not currently in the list.

### Delete

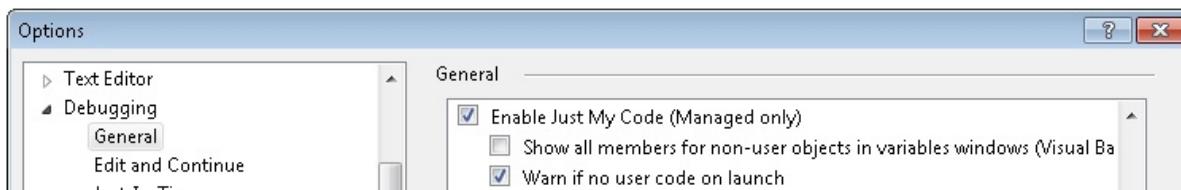
Lets you delete any added exceptions.

## Special Case

You might find the User-Unhandled option missing from the Exceptions dialog box:



To get it back, go to Tools | Options | Debugging | General and select Enable Just My Code (Managed Only):



## 07.43 Setting a Breakpoint in the Call Stack Window

<b>DEFAULT</b>	Ctrl+Alt+C
<b>VISUAL BASIC 6</b>	Ctrl+Alt+C; Ctrl+L
<b>VISUAL C# 2005</b>	Ctrl+Alt+C; Ctrl+D, Ctrl+C; Ctrl+D, C
<b>VISUAL C++ 2</b>	Ctrl+Alt+C; Ctrl+K; Alt+6
<b>VISUAL C++ 6</b>	Ctrl+Alt+C; Alt+7
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+C
<b>WINDOWS</b>	Alt,D, W, C
<b>MENU</b>	Debug   Toggle Breakpoint
<b>COMMAND</b>	Debug.CallStack
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0008

Did you know that breakpoints can be set inside the Call Stack window? This tip shows how to do it.

First, set a breakpoint deep in series of calls to get a nice call stack.

### NOTE

If you don't have a call stack handy, just make several methods called One, Two, Three, and so forth and have them call each other, as I have done in these examples.

Run your code, and let it stop at the breakpoint. Bring up your Call Stack window (Ctrl+Alt+C or Debug | Windows | Call Stack):

Call Stack	
Name	Lang
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Doh() Line 56	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Ti() Line 51 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.La() Line 46 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.So() Line 41 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Fa() Line 36 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Mi(string s = "Hello from C#") Line 31 + 0x10 bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Re(double d = 0.667) Line 28 + 0x10 bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Do(int iParam = 10, int iA) Line 25 + 0x10 bytes	C#
CSBreakPoints.exe!CSBreakPoints.MainForm.buttonDeepStack_Click(object s)	C#
[External Code]	
CSBreakPoints.exe!CSBreakPoints.Program.Main() Line 17 + 0x28 bytes	C#
[External Code]	

Click somewhere in the call stack you would like to stop at as it unwinds, and press F9:

Call Stack	
Name	Lang
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Doh() Line 56	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Ti() Line 51 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.La() Line 46 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.So() Line 41 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Fa() Line 36 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Mi(string s = "Hello from C#") Line 31 + 0x10 bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Re(double d = 0.667) Line 28 + 0x10 bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Do(int iParam = 10, int iA) Line 25 + 0x10 bytes	C#
CSBreakPoints.exe!CSBreakPoints.MainForm.buttonDeepStack_Click(object s)	C#
[External Code]	
CSBreakPoints.exe!CSBreakPoints.Program.Main() Line 17 + 0x28 bytes	C#
[External Code]	

It sets a breakpoint. You can verify this by looking in your Breakpoints window:

Breakpoints	
New	X
<input checked="" type="checkbox"/>	CSBreakPoints.SomethingToDo.So() + 0x0000002a
<input checked="" type="checkbox"/>	SomethingToDo.cs, line 56 character 13

Now just press F5 to continue, and watch as the debugger stops at the place you told it to:

Call Stack	
Name	Lang
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.So() Line 41 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Fa() Line 36 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Mi(string s = "Hello from	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Re(double d = 0.667) Line	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Do(int iParam = 10, int iA	C#
CSBreakPoints.exe!CSBreakPoints.MainForm.buttonDeepStack_Click(object s	C#
[External Code]	
CSBreakPoints.exe!CSBreakPoints.Program.Main() Line 17 + 0x28 bytes	C#
[External Code]	

## 07.44 Setting a Tracepoint in the Call Stack Window

<b>DEFAULT</b>	Ctrl+Alt+C
<b>VISUAL BASIC 6</b>	Ctrl+Alt+C; Ctrl+L
<b>VISUAL C# 2005</b>	Ctrl+Alt+C; Ctrl+D, Ctrl+C; Ctrl+D, C
<b>VISUAL C++ 2</b>	Ctrl+Alt+C; Ctrl+K; Alt+6
<b>VISUAL C++ 6</b>	Ctrl+Alt+C; Alt+7
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+C
<b>WINDOWS</b>	Alt,D, W, C
<b>COMMAND</b>	Debug.CallStack
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0009

In vstipDebug0008 ([07.43 Setting a Breakpoint in the Call Stack Window](#), page 367), I showed you a classic technique of setting a breakpoint in the Call Stack window. The only problem is that breakpoints tend to be somewhat intrusive if all you want is information. So I thought it would be a good idea to also show how to set tracepoints. For more information about tracepoints, see vstipDebug0010 ([07.25 Setting a Tracepoint in Source Code](#), page 325).

1. Set a breakpoint deep in series of calls to get a nice call stack.

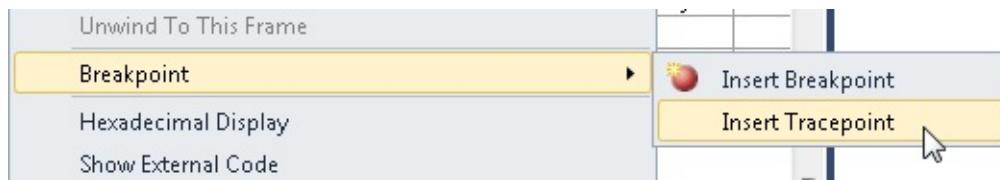
**NOTE**

If you don't have a call series handy, just make several methods called One, Two, Three, and so forth, and have them call each other, as I have done in these examples.

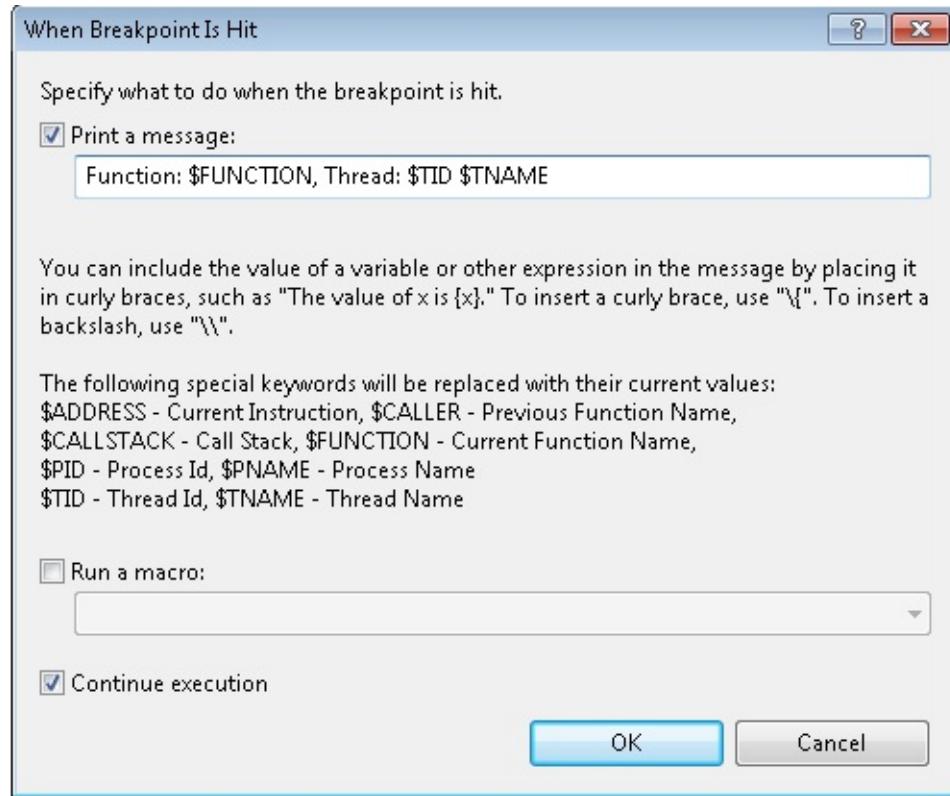
2. Run your code, and let it stop at the breakpoint.
3. Bring up your Call Stack window (Ctrl+Alt+C or Debug | Windows | Call Stack):

Name	Lang
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Doh() Line 56	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Ti() Line 51 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.La() Line 46 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.So() Line 41 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Fa() Line 36 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Mi(string s = "Hello from	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Re(double d = 0.667) Line	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Do(int iParam = 10, int iA	C#
CSBreakPoints.exe!CSBreakPoints.MainForm.buttonDeepStack_Click(object s	C#
[External Code]	
CSBreakPoints.exe!CSBreakPoints.Program.Main() Line 17 + 0x28 bytes	C#
[External Code]	

4. Right-click some place in the stack where you would like to set a tracepoint, and go to Breakpoint | Insert Tracepoint:



5. You get the When Breakpoint Is Hit dialog box:

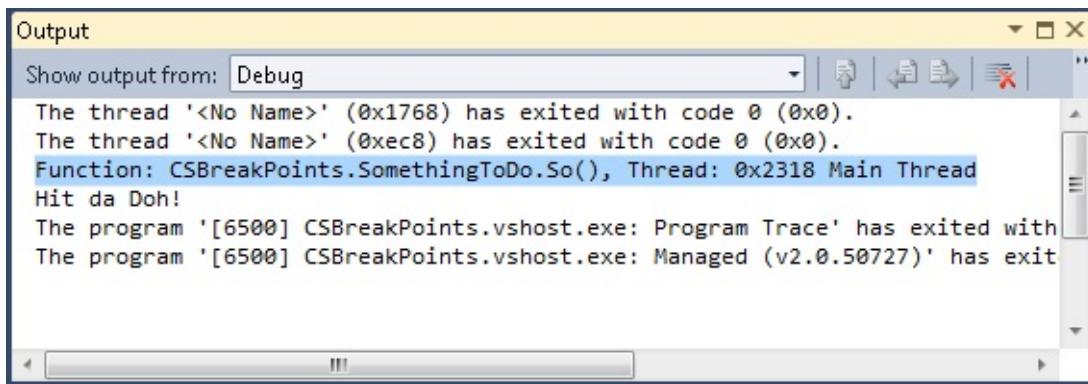


6. You definitely want to get up to speed on the details of tracepoints if you aren't familiar with them (see `vstipDebug0010`, [07.25 Setting a Tracepoint in Source Code](#) on page 325), but for now, just click OK.
7. Now we get our tracepoint. (Notice the diamond glyph.)

Call Stack	
Name	Lang
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Doh() Line 56	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Ti() Line 51 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.La() Line 46 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.So() Line 41 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Fa() Line 36 + 0xa bytes	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Mi(string s = "Hello from	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Re(double d = 0.667) Line	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Do(int iParam = 10, int iA	C#
CSBreakPoints.exe!CSBreakPoints.MainForm.buttonDeepStack_Click(object s	C#
[External Code]	
CSBreakPoints.exe!CSBreakPoints.Program.Main() Line 17 + 0x28 bytes	C#
[External Code]	

8. Press F5 to continue the program and let the call stack unwind.
9. Now bring up your Output window (Debug | Windows | Output), and notice

that the trace information is in there:



The screenshot shows the 'Output' window from Microsoft Visual Studio. The title bar says 'Output'. A dropdown menu 'Show output from:' is set to 'Debug'. The main pane displays the following text:

```
The thread '<No Name>' (0x1768) has exited with code 0 (0x0).
The thread '<No Name>' (0xec8) has exited with code 0 (0x0).
Function: CSBreakPoints.SomethingToDo.So(), Thread: 0x2318 Main Thread
Hit da Doh!
The program '[6500] CSBreakPoints.vshost.exe: Program Trace' has exited with
The program '[6500] CSBreakPoints.vshost.exe: Managed (v2.0.50727)' has exit
```

## 07.45 Using the WPF Tree Visualizer

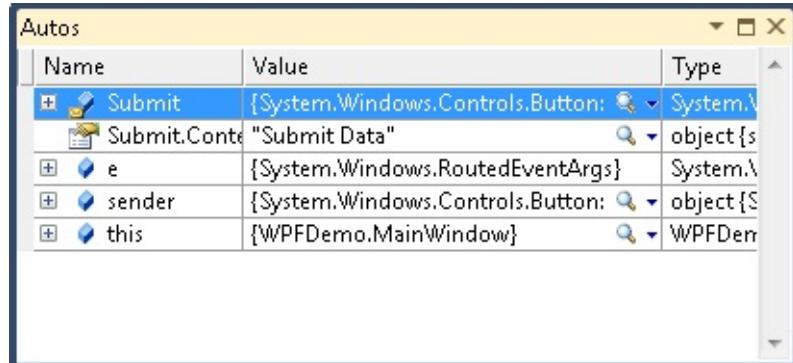
VERSIONS	2010
CODE	vstipDebug0004

The WPF Tree Visualizer started out as a CodePlex project and ended up in the product itself as a visualizer. This tip shows how you can use it.

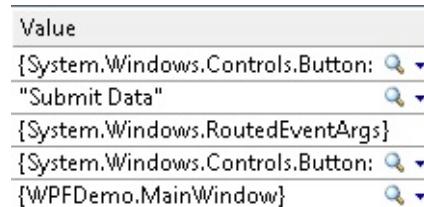
### NOTE

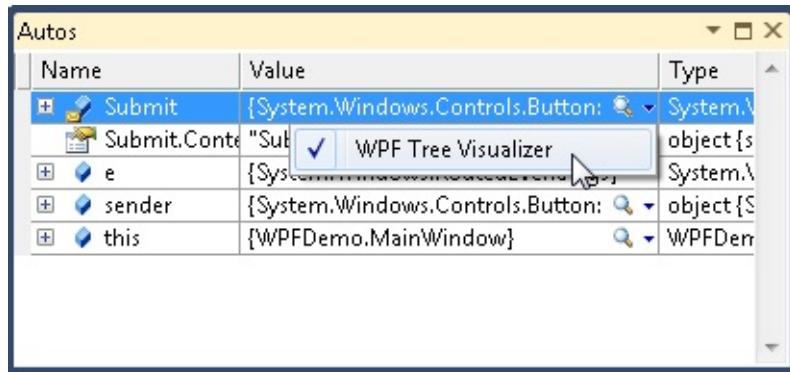
For those unfamiliar with WPF trees, you might want to review the article “Trees in WPF,” at [http://msdn.microsoft.com/en-us/library/ms753391\(VS.100\).aspx](http://msdn.microsoft.com/en-us/library/ms753391(VS.100).aspx).

Enter Debug mode in any WPF project you have. Then take a look at a DataTip, the Watch window, the Autos window, or the Locals window. For this example, let’s use the Autos window:

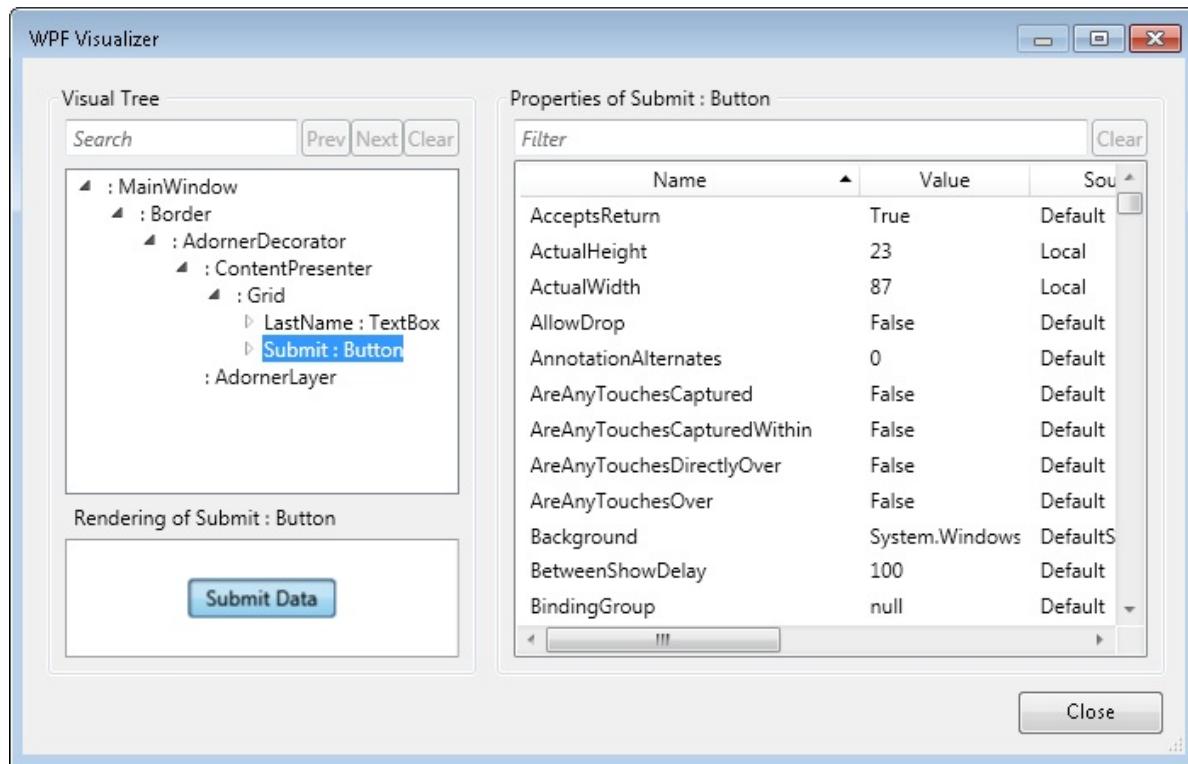


Choose any control in the Autos window, and then click the magnifying glass way over to the right of the control name:

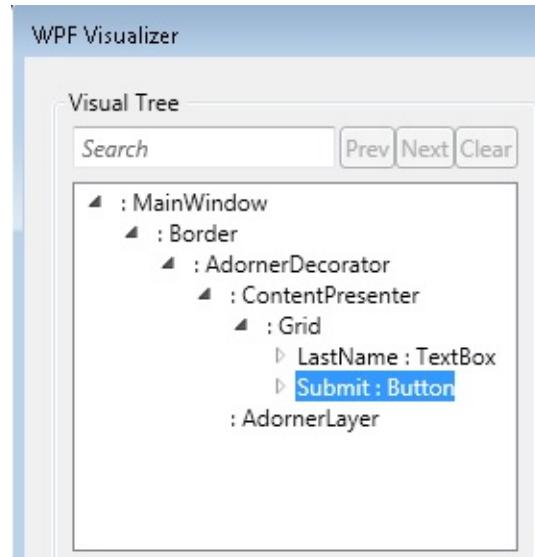




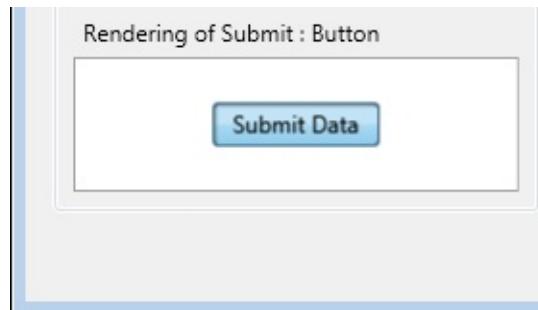
On the drop-down list, choose WPF Tree Visualizer to get the following dialog box:



This dialog box has a lot of moving parts, so let's take a look at each one. First, the Visual Tree section shows you the hierarchy of the controls:



Clicking on any particular node of the tree shows you the rendering at the bottom:



Also, the selected control has its properties displayed in the large area to the right:

Properties of Submit : Button		
Filter		
Name	Value	Source
AcceptsReturn	True	Default
ActualHeight	23	Local
ActualWidth	87	Local
AllowDrop	False	Default
AnnotationAlternates	0	Default
AreAnyTouchesCaptured	False	Default
AreAnyTouchesCapturedWithin	False	Default
AreAnyTouchesDirectlyOver	False	Default
AreAnyTouchesOver	False	Default
Background	System.Windows	DefaultS
BetweenShowDelay	100	Default
BindingGroup	null	Default

In both the Visual Tree and the Properties area, you can search or filter the results by typing into the Search or Filter text boxes respectively:

Properties of : MainWindow		
Keyboard		
Name	Value	Source
AcceptsReturn	False	Default
ControlTabNavigation	Cycle	Default
DirectionalNavigation	Cycle	Default
IsKeyboardFocused	False	Local
IsKeyboardFocusWithin	True	Local
IsTabStop	False	Default
TabIndex	2147483647	Default
TabNavigation	Cycle	Default

### WARNING

Watch out for the results because they might not be what you expect. See the extra items in the list that don't have the word "keyboard" in them? How did they get there? Well, if I scroll to the right and look at

other properties, you can see how it happened. Currently, there is no way that I am aware of to change this behavior.

Properties of : MainWindow	
Style Value	Declared Type
	System.Windows.Input.KeyboardNavigation
	System.Windows.Input.KeyboardNavigation
	System.Windows.Input.KeyboardNavigation
<not set>	System.Windows.UIElement
<not set>	System.Windows.UIElement
	System.Windows.Input.KeyboardNavigation
	System.Windows.Input.KeyboardNavigation
	System.Windows.Input.KeyboardNavigation

## 07.46 Understanding Break All Processes When One Process Breaks

<b>MENU</b>	Tools   Options   Debugging   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0029

In vstipEnv0015 ([02.05 Multiple Startup Projects](#), page 48), we examined how to run multiple projects at the same time. Now let's explore what happens when you go into break mode on the applications. By default, when you break one application, all the other applications break too. Let's take a look at an example.

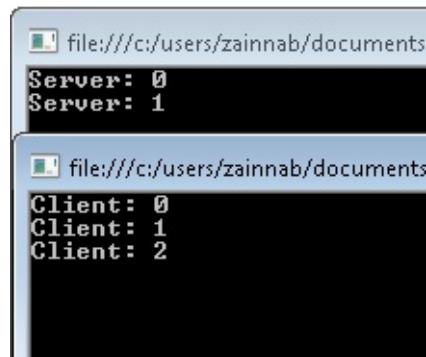
You set a breakpoint in your first application that has a hit counter on it (see [vstipDebug0019, 07.20 Breakpoint Hit Count](#), on page 314):

```
static void Main(string[] args)
{
    for (int counter = 0; counter <= 20; counter++)
    {
        Console.WriteLine("Server: " + counter);
        System.Threading.Thread.Sleep(1000);
    }
}
```

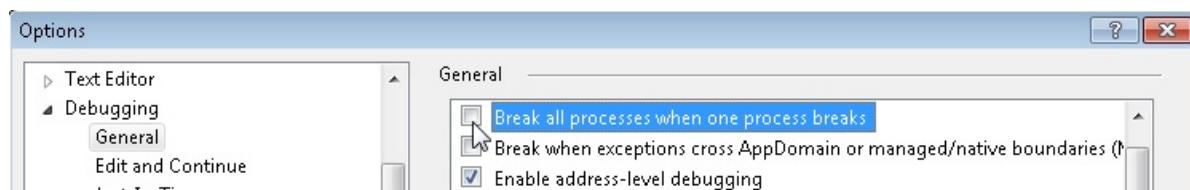
But you don't have any breakpoints in your second application:

```
static void Main(string[] args)
{
    for (int counter2 = 0; counter2 <= 20; counter2++)
    {
        Console.WriteLine("Client: " + counter2);
        System.Threading.Thread.Sleep(1000);
    }
}
```

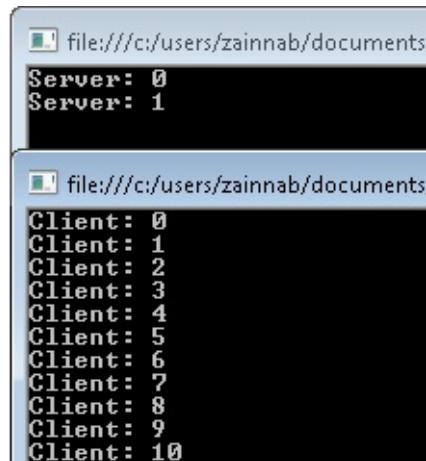
When you run the applications, you get the following:



Both applications stop when one of them stops. This behavior is set in Tools | Options | Debugging | General by selecting the Break All Processes When One Process Breaks check box:



If you turn this feature off and run your applications again, you see the following (predictable) result:



## 07.47 Changing Context in the Locals Window

<b>DEFAULT</b>	Ctrl+Alt+V, L (locals); Ctrl+5 then Alt+Down Arrow (process combo); Ctrl+6 then Alt+Down Arrow (thread combo); Ctrl+7 then Alt+Down Arrow (stack frame combo)
<b>VISUAL BASIC 6</b>	Ctrl+Alt+V, L (locals); Ctrl+5 then Alt+Down Arrow (process combo); Ctrl+6 then Alt+Down Arrow (thread combo); Ctrl+7 then Alt+Down Arrow (stack frame combo)
<b>VISUAL C# 2005</b>	Ctrl+Alt+V, L (locals); Ctrl+D, Ctrl+L (locals); Ctrl+D, L (locals); Ctrl+5 then Alt+Down Arrow (process combo); Ctrl+6 then Alt+Down Arrow (thread combo); Ctrl+7 then Alt+Down Arrow (stack frame combo)
<b>VISUAL C++ 2</b>	Ctrl+Alt+V, L (locals); Alt+3 (locals); Ctrl+5 then Alt+Down Arrow (process combo); Ctrl+6 then Alt+Down Arrow (thread combo); Ctrl+7 then Alt+Down Arrow (stack frame combo)
<b>VISUAL C++ 6</b>	Ctrl+Alt+V, L (locals); Alt+4 (locals); Ctrl+5 then Alt+Down Arrow (process combo); Ctrl+6 then Alt+Down Arrow (thread combo); Ctrl+7 then Alt+Down Arrow (stack frame combo)
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+V, L (locals); Ctrl+Alt+L (locals); Ctrl+5 then Alt+Down Arrow (process combo); Ctrl+6 then Alt+Down Arrow (thread combo); Ctrl+7 then Alt+Down Arrow (stack frame combo)
<b>WINDOWS</b>	Alt,D, W, L (locals)
<b>MENU</b>	Debug   Windows   Locals
<b>COMMAND</b>	Debug.Locals; Debug.LocationToolbar.ProcessCombo;Debug.LocationToolbar.ThreadCombo; Debug.LocationToolbar.StackFrameCombo
<b>VERSIONS</b>	2005, 2008, 2010

Most people are familiar with the Locals window. For those who aren't, it displays variables local to the current context or scope. Information often appears in the Locals window, but the information will not be current until the next time the program breaks.

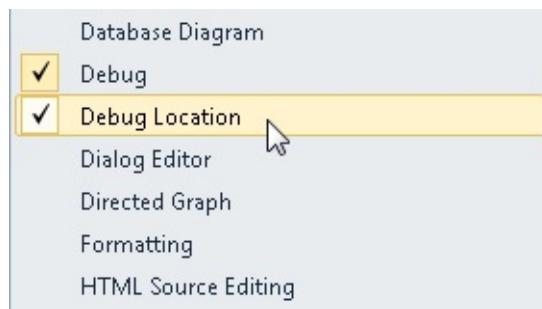
Even those who are familiar with the Locals window might not know that you can change the context. When you are in break mode, bring up the Locals window (Ctrl+Alt+V, L):

```
double eLocal = x + d;
eLocal = eLocal * d;
Mi("A me so la ti do");
}

Locals
Name Value Type
this {CSBreakPoints.SomethingToD CSBreakl
d 0.667 double
x 20 int
eLocal 13.784889000000002 double
```

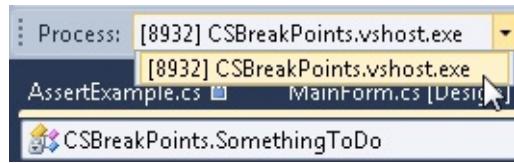
## Debug Location Toolbar

You can use the Debug Location toolbar to change the context of the Locals window. If you don't have it up, you can right-click any toolbar and select Debug Location to make it appear:



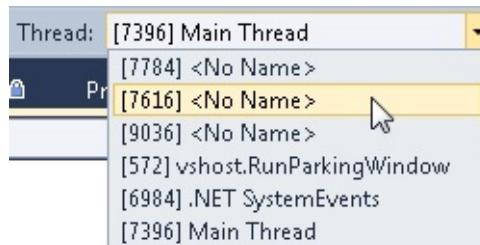
## Process

If you have multiple processes running, you can use the Process combo box (Ctrl+5) to change between them. In this example, we only have one process running:



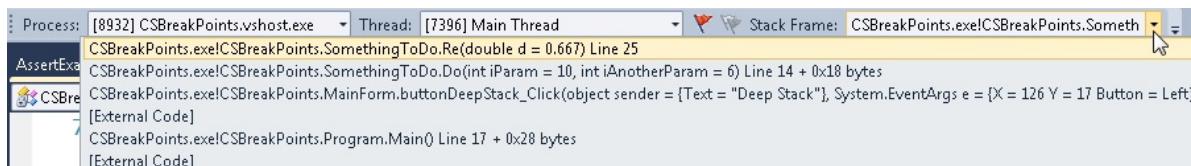
## Thread

The Thread combo (Ctrl+6) lets you change the thread context to different threads. While not of much use in this example, it is very useful in debugging multithreaded applications:



## Stack Frame

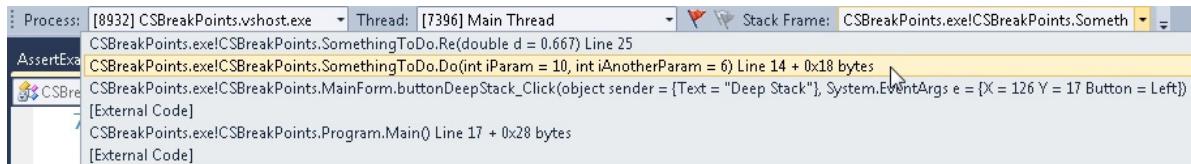
The stack frame is probably the more common item you use when dealing with the Locals window. The Stack Frame combo box (Ctrl+7) allows you to switch between items in the call stack:



Currently, we're in the one method and the Locals window shows the following:

Locals			
Name	Value	Type	
this	{CSBreakPoints.SomethingToDo}	CSBreak	
d	0.667	double	
x	20	int	
eLocal	13.784889000000002	double	

Let's change to another method in our code:



Now our Locals window shows the following:

Name	Value	Type
this	{CSBreakPoints.SomethingToDo}	CSBreak
iParam	10	int
iAnotherParam	6	int
jLocal	10	int

Now you too can take advantage of the Debug Location toolbar to change your context and get new information in the Locals window.

## 07.48 Understanding the Autos Window

<b>DEFAULT</b>	Ctrl+Alt+V, A
<b>VISUAL BASIC 6</b>	Ctrl+Alt+V, A
<b>VISUAL C# 2005</b>	Ctrl+Alt+V, A; Ctrl+D, Ctrl+A; Ctrl+D, A
<b>Visual C++ 2:</b>	Ctrl+Alt+V, A
<b>VISUAL C++ 6</b>	Ctrl+Alt+V, A
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+V, A
<b>WINDOWS</b>	Alt,D, W, A
<b>MENU</b>	Debug   Windows   Autos
<b>COMMAND</b>	Debug.Autos
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0103

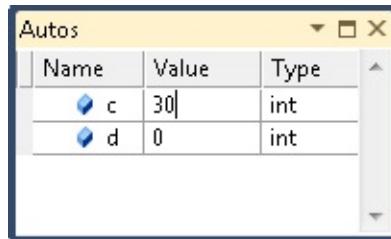
The Autos window looks a lot like the Locals window, but let's take a closer look. According to the MSDN website ([http://msdn.microsoft.com/en-us/library/aa290702\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/aa290702(VS.71).aspx)), here is what it does:

*“The Autos window displays variables used in the current statement and the previous statement. (For Visual Basic, it displays variables in the current statement and three statements on either side of the current statement.)”*

The current statement is the statement at the current execution location (the statement that will be executed next if execution continues). The debugger identifies these variables for you automatically, hence the window name.”

## Changing Values

Just like the Locals window, you can change values in the Autos window:

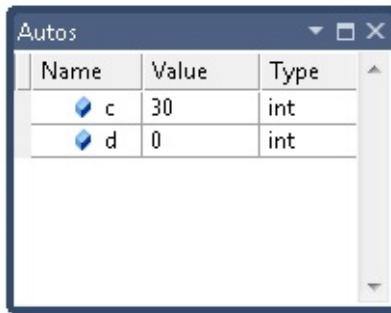


# Current and Previous Statement

## Simple Example

OK let's break this down. The Autos window does, in fact, show variables used in the current statement and previous statement. Following is a simple example in C# with some variables:

```
int a = 10;
int b = 20;
int c = 30;
int d = 40; // This line is highlighted
int e = 50;
int f = 60;
int g = 70;
```

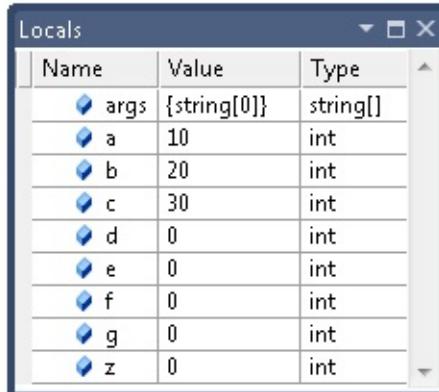


The Autos window displays the current and previous statement variables. It shows two rows: one for variable 'c' with value 30 and type int, and another for variable 'd' with value 0 and type int. The variable 'd' is highlighted in yellow, indicating it is the current statement variable.

Name	Value	Type
c	30	int
d	0	int

As you can see, it just shows the variable on the current line and the one before it —nothing else. Even though several lines have variables around them, the Autos window is not concerned with those variables. Now compare this to the Locals window that shows every variable in scope:

```
int a = 10;
int b = 20;
int c = 30;
int d = 40; // This line is highlighted
int e = 50;
int f = 60;
int g = 70;
```



The Locals window displays all variables in scope. It shows ten rows: args (string[0]), a (10), b (20), c (30), d (0), e (0), f (0), g (0), and z (0). The variable 'd' is highlighted in yellow, indicating it is the current statement variable.

Name	Value	Type
args	{string[0]}	string[]
a	10	int
b	20	int
c	30	int
d	0	int
e	0	int
f	0	int
g	0	int
z	0	int

## Another Example

I thought it would be instructive to see, step-by-step, how this works. Here I have stopped in some code:

```

d = 0.667;
Debug.WriteLine("blah");
Debug.WriteLine("blah");
Debug.WriteLine("blah");
Debug.WriteLine("blah");
double eLocal = 0.002;
eLocal = eLocal * d;
Mi("A me so la ti do");

```

Autos		
Name	Value	Type
d	31.45	double
this	{CSBreakPoints.SomethingTo CS}	

Notice that it shows the current variable and the object variable that exists all the time. Now I go down two lines:

```

d = 0.667;
Debug.WriteLine("blah");
Debug.WriteLine("blah");
Debug.WriteLine("blah");
Debug.WriteLine("blah");
double eLocal = 0.002;
eLocal = eLocal * d;
Mi("A me so la ti do");

```

Autos		
Name	Value	Type
this	{CSBreakPoints.SomethingTo CS}	

All I'm left with is the object variable. I go down another three lines:

```

d = 0.667;
Debug.WriteLine("blah");
Debug.WriteLine("blah");
Debug.WriteLine("blah");
Debug.WriteLine("blah");
double eLocal = 0.002;
eLocal = eLocal * d;
Mi("A me so la ti do");

```

Autos		
Name	Value	Type
eLocal	0.0	do
this	{CSBreakPoints.SomethingTo	CS

Now I can see that “eLocal” is about to be used, and I go down one more line:

```

d = 0.667;
Debug.WriteLine("blah");
Debug.WriteLine("blah");
Debug.WriteLine("blah");
Debug.WriteLine("blah");
double eLocal = 0.002;
eLocal = eLocal * d;
Mi("A me so la ti do");

```

Autos		
Name	Value	Type
d	0.667	do
eLocal	0.002	do
this	{CSBreakPoints.SomethingTo	CS

I can finally see that “eLocal” has changed, and “d” has shown up again because it is being used on the current line.

## VB Shows Three Statements on Either Side

In versions prior to 2010, VB would, in fact, show three statements on either side of the current line. This is no longer true in Visual Studio 2010. The Autos window in Visual Basic acts just like the C++ and C# Autos windows and shows only the current line and the one prior.

## Part II. Extensions for Visual Studio

In this part:

[Chapter 8](#)

# **Chapter 8. Visual Studio Extensions**

The new code editor is one of the most impressive improvements of the Visual Studio 2010 Integrated Development Environment (IDE). Built on top of Windows Presentation Foundation (WPF) and the Managed Extensibility Framework, the combination of user interface-rich flexibility plus plug-in extensibility takes innovation within Visual Studio to the next level.

# **Introducing Visual Studio Extensions**

## **Introducing Visual Studio Extensions**

In addition to the editor, the Visual Studio 2010 IDE introduces a new WPF-based shell. Many core user interface features are now WPF-based, including menus, window layouts, toolbars, start page, and so forth.

Even if you've never developed any applications for use within Visual Studio, it is important to know that the SDK tooling has significantly improved, paving the way for a much richer Visual Studio ecosystem.

Gone are the days of requesting a package load key or a shell load key to develop Visual Studio applications. No more having to make modifications to the registry to install a Visual Studio plug-in.

The goals of this chapter are twofold: to provide a basic overview on how to find and install extensions and to catalogue the “must-have” extensions available for free for the Visual Studio 2010 IDE.

Welcome to the new world of Visual Studio extensions.

## **Installing an Extension**

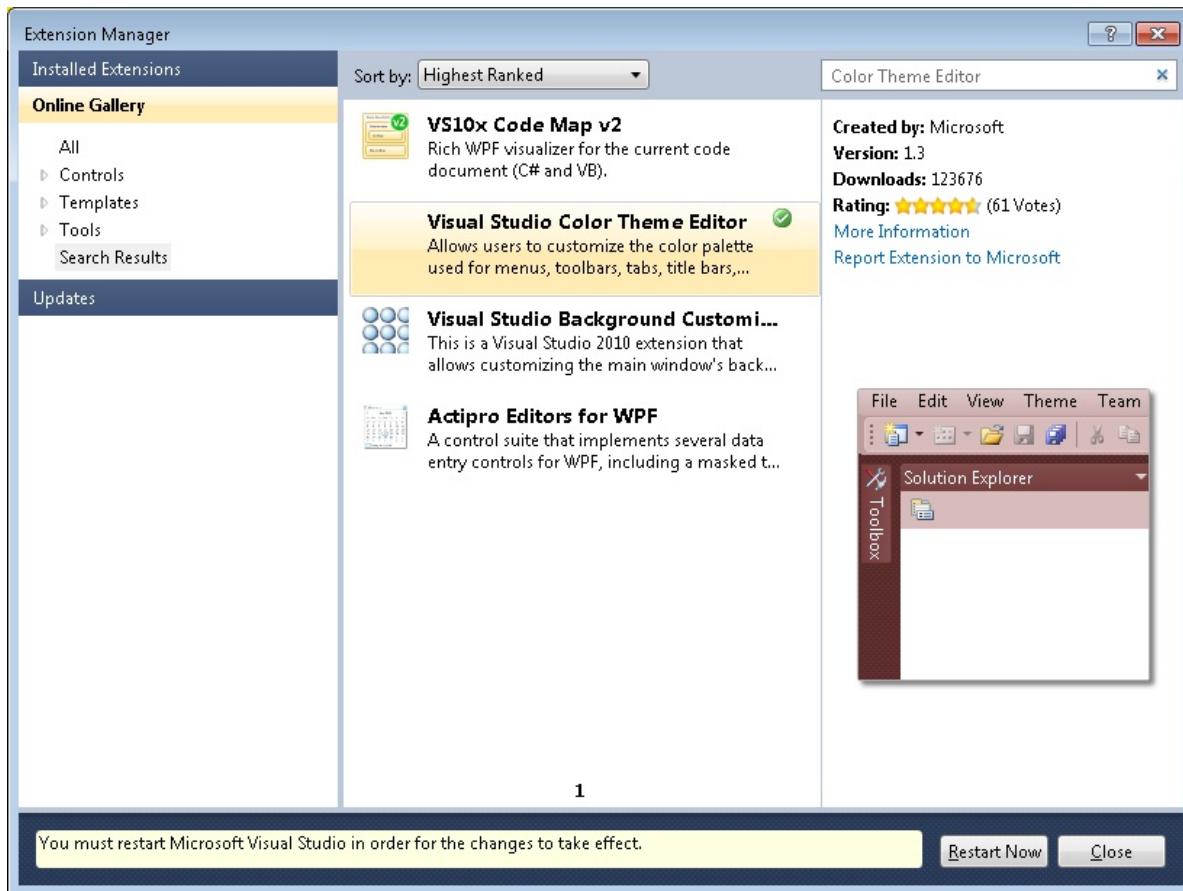
The development experience has greatly improved for Visual Studio 2010, providing additional ways for installing extensions. The preferred and most straightforward way to install an extension is through a new Visual Studio 2010 feature called the Extension Manager. Located within the IDE, the Extension Manager searches an online gallery, called the Visual Studio Gallery, for extensions. The Extension Manager also manages the installed and currently in-use extensions.

## **Installing from the Extension Manager**

To open the Extension Manager from within Visual Studio, go to Tools | Extension Manager. To search for an extension online, select Online Gallery in the left navigation pane. Type in the name of the extension to search for within the Search Online Gallery text box. For example, you can search for “Color Theme Editor” to find the Visual Studio Color Theme Editor extension.

When the desired extension is found, click the Download button to install. When the install has completed, a message appears at the bottom of the Extension

Manager dialog box, as shown in the following illustration.



A Visual Studio restart is required for all extensions, except for templates. Click Restart Now to restart Visual Studio and enable the extension.

## Installing from the Visual Studio Gallery

Another addition to Visual Studio 2010 is the Visual Studio Gallery, a website that provides a catalog of free and commercial extensions for Visual Studio. Naturally, developers writing extensions have the option of hosting the extension on their own websites, but hosting in the Visual Studio Gallery greatly increases discoverability of the extension. Additionally, you can find a significant number of tools for all past Visual Studio IDE versions from 2002 onward in the Visual Studio Gallery.

The Visual Studio Gallery is located at [www.visualstudiogallery.com](http://www.visualstudiogallery.com). To browse for extensions built for Visual Studio 2010, click Browse and then select Visual Studio 2010 under Visual Studio Versions in the left navigation bar.

When the desired extension is found, click the Download image link to begin the download and installation.

## Visual Studio Color Theme Editor Free

CREATED BY	<a href="#">Matthew Johnson [MSFT] (Microsoft)</a>	LAST UPDATED
REVIEWS	 (61) <a href="#">Review</a>	VERSION
SUPPORTS	Visual Studio 2010	SHARE
DOWNLOADS	<a href="#">Download</a> (123,677)	FAVORITES
TAGS	<a href="#">theme</a> , <a href="#">themes</a> , <a href="#">Visual Studio 2010</a> , <a href="#">theming</a> ,	

You can either open the .vsix file to begin the installation or save the file to install at a later time. If you chose to save the file, you can double-click the .vsix file at any time to start the installation. Otherwise, simply click Open to begin installing the extension. The Visual Studio Extension Manager prompts you to accept a license if one was provided with the extension.

A restart is required to finish installing the extension. The Extension Manager shows a “restart required” message at the bottom of the dialog box. A Visual Studio restart is required for all extensions, except for templates. Click Restart Now to restart Visual Studio and enable the extension.

## Installing Through Xcopy

Because the SDK tooling has been greatly simplified, such that registry modifications are no longer necessary, it is possible to install an extension simply by copying the extension file into a specific Visual Studio folder. This method of installation through copying files is known as Xcopy deployment.

All installed extensions can be found in an unzipped format at the following location:

```
%LocalAppData%\Microsoft\VisualStudio\10.0\Extensions\<Company>\<Product>\<Version>
```

Extensions installed through the Xcopy method are disabled by default within Visual Studio. You must manually enable the extension, as described next.

To install an extension through this method, you must do the following:

1. Unzip the .vsix file.
2. Copy the .vsix raw folder into the Extensions folder.
3. Start Visual Studio, and enable the extension in the Extension Manager.

Conversely, you can uninstall any extension simply by deleting its corresponding

folder from the %LocalAppData%\Microsoft\VisualStudio\10.0\Extensions directory.

## Inside a .vsix File

Although creating extensions is beyond the scope of this book, it's important to be aware how content written by members of the community is installed and run on your computer, especially in the case where you need to troubleshoot a faulty extension.

All Visual Studio extensions use the .vsix file extension. A .vsix file is a .zip file (with its file extension renamed to .vsix) that uses the Open Packaging Convention to package the code and manifest for the extension. You can read more about the contents of a .vsix file on Quan To's blog at

<http://blogs.msdn.com/b/quanto/archive/2009/05/26/what-is-a-vsix.aspx>.

As described on Quan To's article, the .vsix file is derived from the Visual Studio Installer found in the Visual Studio 2005 and 2008 versions. The original .vsi file launches the Content Installer from these past Visual Studio versions. The .vsi file represented many various types of content for the Visual Studio IDE (macros, add-ins, toolbox controls, code snippets, and so forth). To make the distinction clearer from this previous method of installing content, an 'x' was placed at the end of the extension, hence a .vsix file.

For more information about the Open Packaging Convention, please see the article titled "A New Standard For Packaging Your Data," at

<http://msdn.microsoft.com/en-us/magazine/cc163372.aspx>.

## Disabling an Extension

If you need to disable an extension for any reason—for example, the Visual Studio performance is slower than expected, the IDE crashes repeatedly, or the extension is simply not working—you can disable an extension via Extension Manager.

Select Installed Extensions in the left navigation pane of the Extension Manager, and then click Disable on any extension in the list. Again, a restart is required to disable the extension.

### NOTE

You can disable or re-enable multiple extensions at the same time within the Extension Manager, requiring only one restart of the Visual Studio IDE.

## Uninstalling an Extension

The most straightforward way to uninstall an extension is through the Extension Manager. Select Installed Extension in the left navigation pane of the Extension Manager, and then click Uninstall on any extension in the list. Again, a restart is required to remove the extension.

### NOTE

You can uninstall or reinstall multiple extensions at the same time within the Extension Manager, requiring only one restart of the Visual Studio IDE.

The other way to uninstall an extension is to delete the contents of the extension from the Visual Studio extension folder on your hard drive. Simply delete the extension .vsix file from the %LocalAppData%\Microsoft\VisualStudio\10.0\Extensions directory, and restart Visual Studio.

## Resources for Developing Extensions

As noted, creating and developing extensions are beyond the scope of this book. However, many great resources are available, such as the following:

- **Visual Studio Extensibility Developer Center**—<http://msdn.com/vsx>
- **Extensibility samples**—<http://code.msdn.microsoft.com/vsx>

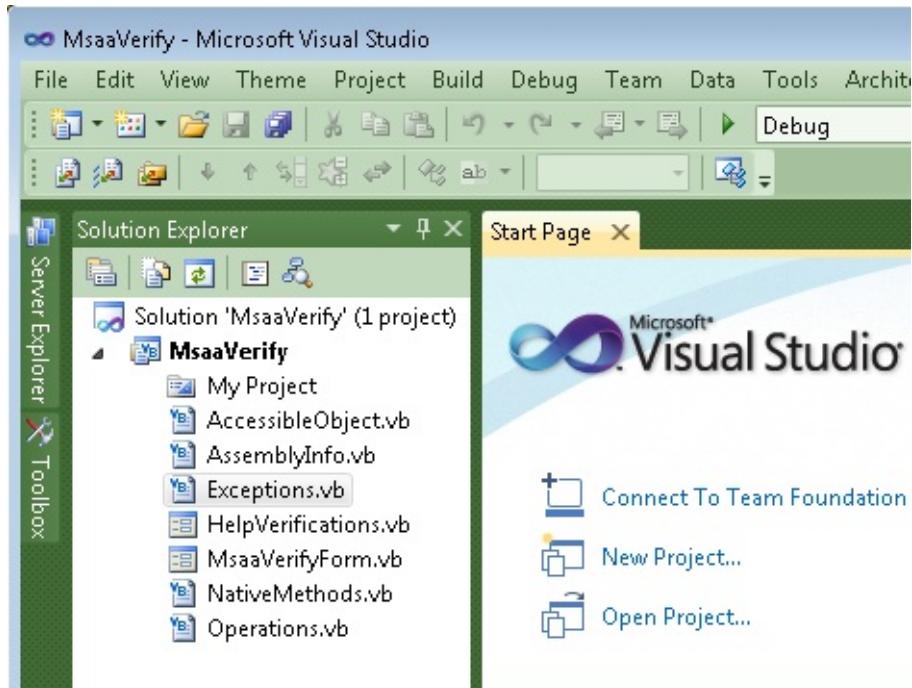
## 08.01 Create Themes Using All Visual Studio Elements

NAME	Visual Studio Color Theme Editor
CREATED BY	Matthew Johnson (Microsoft)
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/en-us/20cd93a2-c435-4d00-a797-499f16402378?SRC=Home">http://visualstudiogallery.msdn.microsoft.com/en-us/20cd93a2-c435-4d00-a797-499f16402378?SRC=Home</a>

You can use the Visual Studio Color Theme Editor to create themes consisting of colors that go beyond those listed in the Tools | Options | Fonts And Colors page.

### Visual Studio Color Theme Editor

The most useful feature of the Visual Studio Color Theme Editor extension is that it provides a central location for controlling almost all possible colors in the IDE. No more having to edit individual fonts and colors options under Tools | Options. Refer to the “To Customize” section later in this section for more information about how to use this feature.

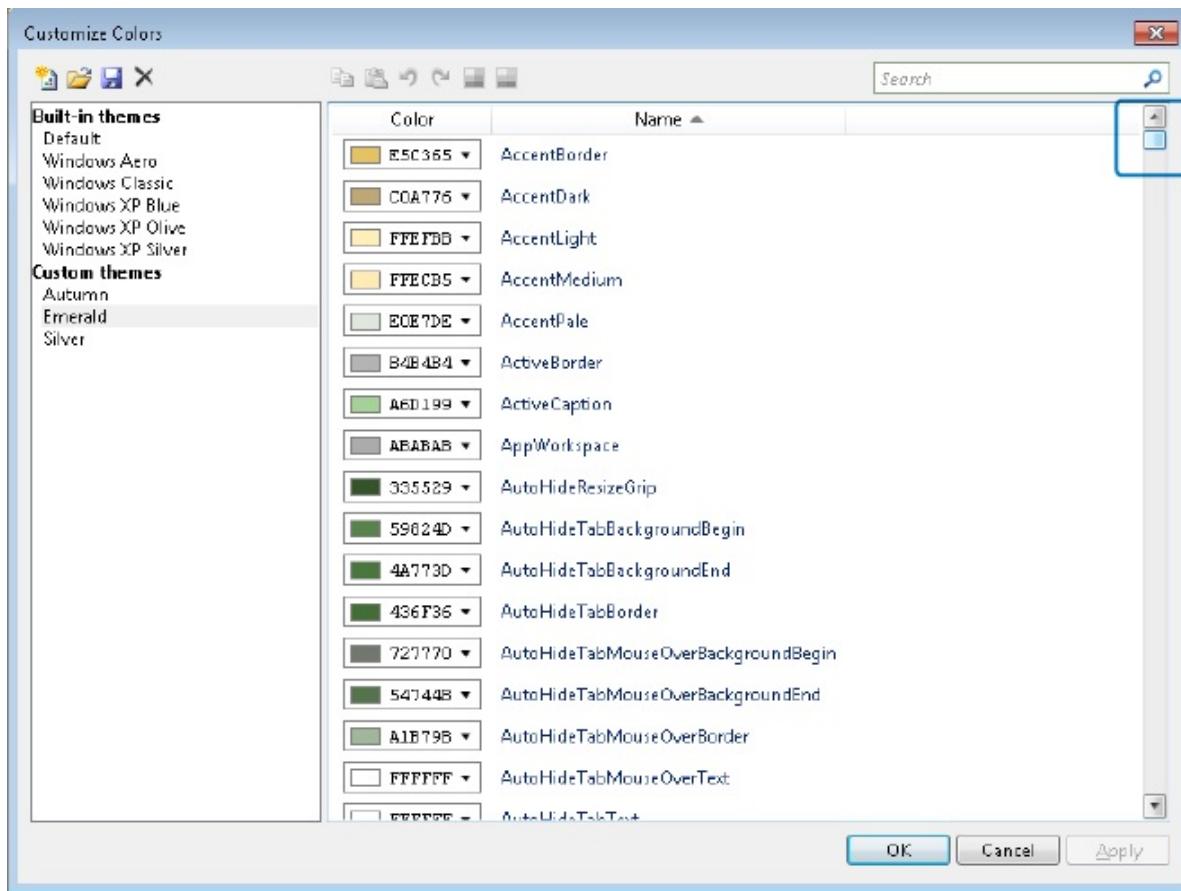


### To Use

A Theme menu option is located on the Visual Studio file menu. The Theme menu lists multiple themes for you to choose from, including the Windows Classic look.

## To Customize

You can create your own themes by clicking Theme | Customize Colors and choosing from hundreds of customized colors. This list of colors is significantly longer than the list found on the Tools | Options | Fonts And Colors page, because this extension lists every user interface element that implements the Visual Studio color service.



You can create new themes by clicking the Save button located in the toolbar in the upper-left corner. All themes are saved as .xml files, using the .vstheme file extension name. You can share a .vstheme file with others.

## More Information

Make sure you check out the extension developer's blog post at <http://blogs.msdn.com/visualstudio/archive/2010/01/04/changing-visual-studio-s-color-palette.aspx> for an in-depth overview of the extension.

## 08.02 Insert Images into Your Code

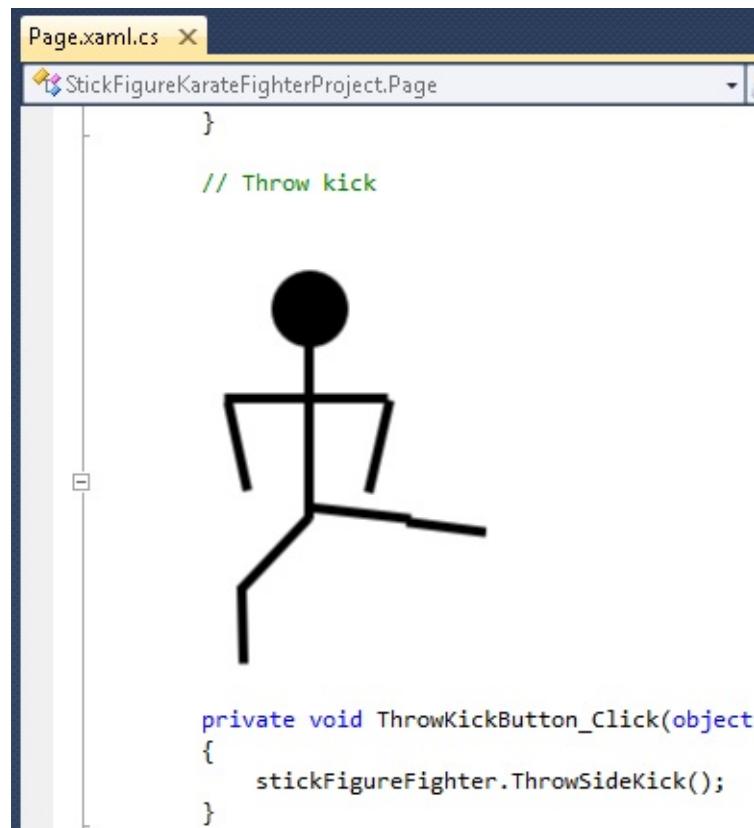
NAME	Image Insertion
CREATED BY	Microsoft
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/en-us/793d16d0-235a-439a-91df-4ce7c721df12">http://visualstudiogallery.msdn.microsoft.com/en-us/793d16d0-235a-439a-91df-4ce7c721df12</a>

The Image Insertion tool allows you insert images, such as a UML diagram, directly into your code. Another example is including user interface designs within your code for documentation purposes.

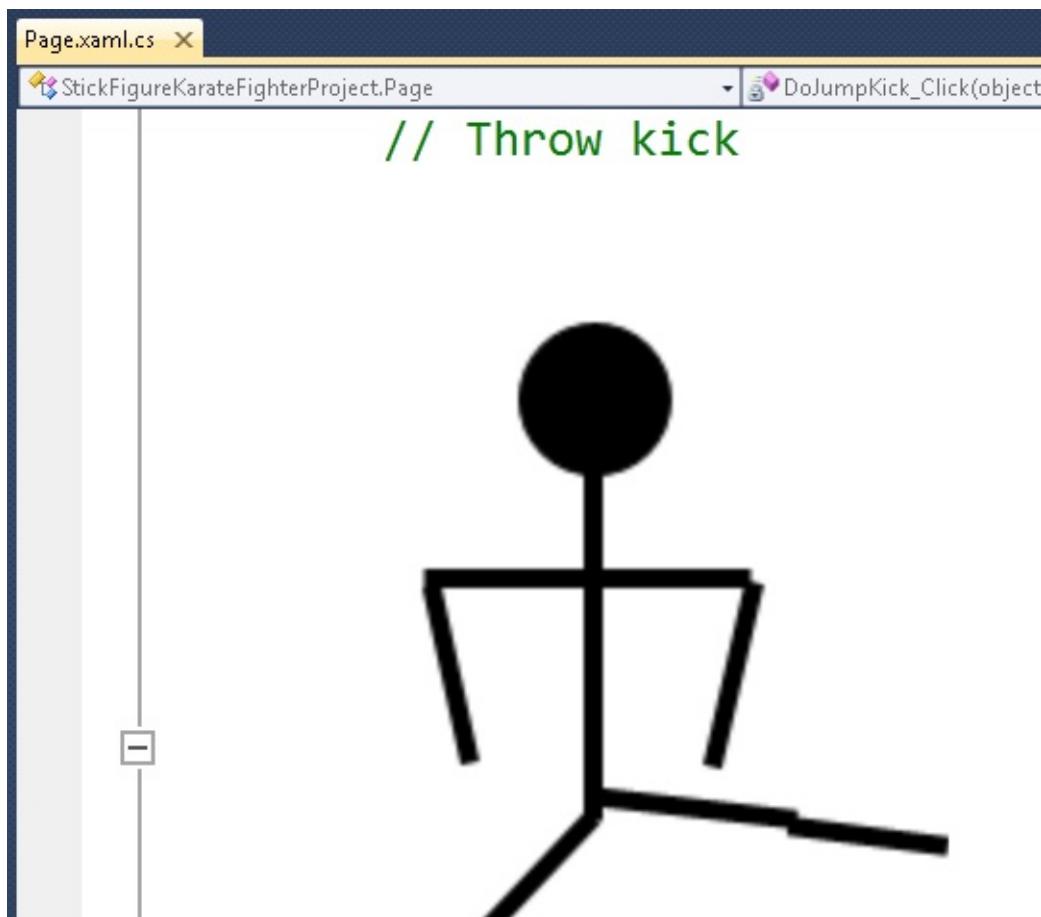
### Image Insertion

Regardless of usage, the Image Insertion extension demonstrates that the Visual Studio 2010 editor is new and that creative extensions are possible.

Recall that the Visual Studio code editor is written using the Windows Presentation Foundation (WPF). The following illustration shows a Silverlight illustration of a hand-drawn stick figure at 100 percent zoom.



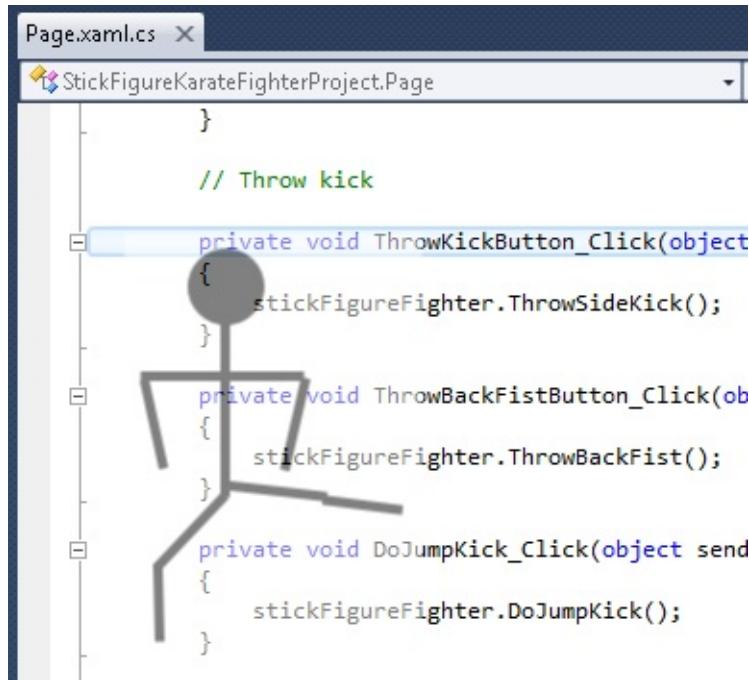
When the editor zoom is increased to 200 percent, notice how both the text *and the stick figure image* increase at the same ratio.



It is worth mentioning that this is not a separate designer built solely for this purpose. This is the actual Visual Studio code editor displaying the image.

## To Use

Select an image from either a Windows Explorer window or Solution Explorer, and drag the image onto the editor surface. To successfully insert the image, you must see the blue placeholder line, as shown in the following illustration. Also note that Visual Studio inserts the image directly *above* the blue line.



To remove the inserted image, place the mouse directly over the image until a Close button appears. Click the X button in the upper-right corner to remove the inserted image. The user interface is sensitive, so it is best to move the mouse slowly towards the X button to avoid having the controls disappear.

To resize the inserted image, use the mouse to grip the arrow controls, just as you would resize any standard window.

## To Save

The images are not saved as part of the code file. The images are saved in a separate .resx file. In this example, because the image is inserted in a Page.xaml.cs file, the image is saved in a Page.xaml.Images.resx file, which is viewable in Solution Explorer. Most source control management systems allow users to customize whether .resx files should be included in a source code check-in.

## To Customize

Although this extension doesn't come with any options to configure, the source code is available for tweaking at <http://editorsamples.codeplex.com>.

## 08.03 Add Visual Guidelines to Your Code

NAME	Editor Guidelines, Editor Guidelines UI
CREATED BY	Paul Harrington (Microsoft)
LOCATIONS	<a href="http://visualstudiogallery.msdn.microsoft.com/0fbf2878-e678-4577-9fdb-9030389b338c">http://visualstudiogallery.msdn.microsoft.com/0fbf2878-e678-4577-9fdb-9030389b338c</a> <a href="http://visualstudiogallery.msdn.microsoft.com/en-us/7f2a6727-2993-4c1d-8f58-ae24df14ea91">http://visualstudiogallery.msdn.microsoft.com/en-us/7f2a6727-2993-4c1d-8f58-ae24df14ea91</a>

The Editor Guidelines displays vertical lines within your code to help you visualize regions, end of printable margins, and so forth.

### Editor Guidelines

Although this feature has been in many previous versions of Visual Studio, it was necessary to modify the registry to use the feature. This extension brings the guidelines feature back to Visual Studio 2010 and includes options within the Visual Studio IDE to modify the guidelines.

```
#Region "Public Methods"

    ' get the control's hwnd
    Public ReadOnly Property Hwnd() As IntPtr
        Get
            If Me.m_hwnd.ToInt32 = 0 Then
                Dim handle As IntPtr

                ' get a handle from the accessible object we have
                NativeMethods.WindowFromAccessibleObject(m_msaa, handle)

                ' create an accessible object
                Dim msaa As New AccessibleObject(Me.IAccessible)

                ' cache the hwnd for future use
                m_hwnd = handle

                Return handle
            Else
                Return Me.m_hwnd
            End If
        End Get
    End Region
```

### To Install

The Editor Guidelines extension consists of two extensions:

- **Editor Guidelines** (<http://visualstudiogallery.msdn.microsoft.com/en-us/0fbf2878-e678-4577-9fdb-9030389b338c>)—This extension brings back the guidelines feature; however, it requires modifications to the registry to use the guidelines.
- **Editor Guidelines UI** (<http://visualstudiogallery.msdn.microsoft.com/en-us/7f2a6727-2993-4c1d-8f58-ae24df14ea91>)—This extension provides options within Visual Studio to use the guidelines, instead of manually editing the registry

#### NOTE

The Editor Guidelines UI extension is an extension of the Editor Guidelines extension. The ability to extend an extension from another extension is a feature of the Managed Extensibility Framework. To install the Editor Guidelines UI extension, the Editor Guidelines extension must be installed first; otherwise, the Extension Manager displays an error message.

## To Use

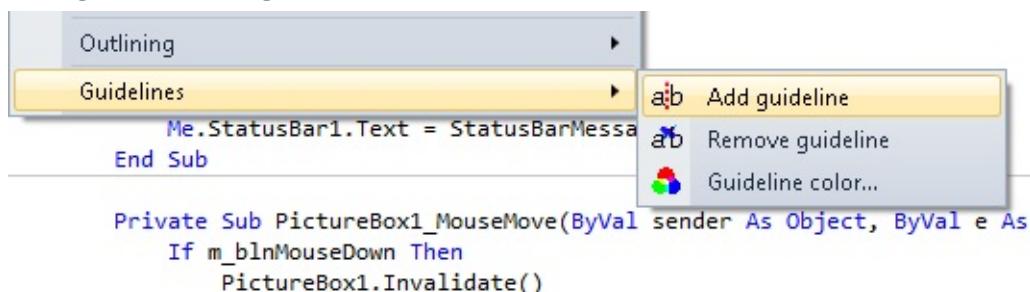
If you plan to use only the first extension, you need to manually edit the registry, so use at your own risk! The registry key for the guidelines feature is the same as in previous versions of Visual Studio:

[HKEY\_CURRENT\_USER]\Software\Microsoft\VisualStudio\10.0\Text Editor.

You need to create a new REG\_SZ string called “Guides” to store the information for the guideline. For example, the string “RGB(0,255,0) 100” sets a green guideline on column 100. To add additional guidelines, add the other column numbers to the Guides value separated by commas, such as RGB(0, 255, 0) 5, 60, 100.

If you have the Editor Guidelines UI extension installed, things are a lot simpler. Right-click wherever you wish to place the guideline, and from the context menu, select **Add Guideline**.

To remove the guideline, place your cursor on the column where you wish to remove the guideline, right-click, and select **Remove Guideline**.



## To Customize

To customize the colors for the guideline, select **Guideline Color** from the context menu, and select colors from the color palette window.

## 08.04 Get More IntelliSense in Your XAML Editor

NAME	XAML IntelliSense Presenter
CREATED BY	Karl Shifflett
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/1a67eee3-fdd1-4745-b290-09d649d07ee0">http://visualstudiogallery.msdn.microsoft.com/1a67eee3-fdd1-4745-b290-09d649d07ee0</a>

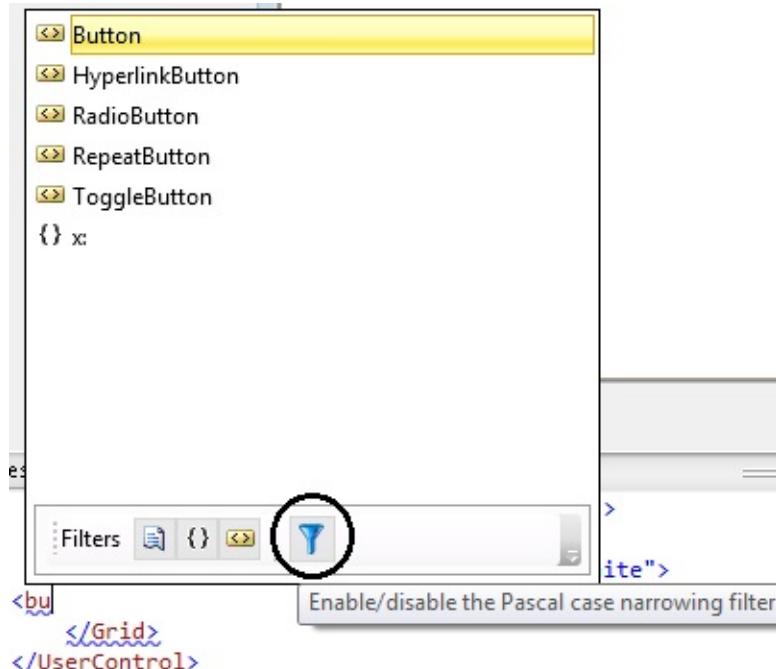
The XAML IntelliSense Presenter extension allows you to use the new Pascal casing and list filtering IntelliSense features in your XAML code.

### XAML IntelliSense Presenter

If you've grown accustomed to the new Pascal casing and list filtering features in IntelliSense, you'll be happy to know that the Visual Studio 2010 XAML Editor IntelliSense extension gives you these features and more in the XAML editor.

#### To Use

Start typing in any XAML editor to bring up IntelliSense. As you start typing, you can see the Pascal case narrowing the available selection. The Pascal case narrowing filter option is enabled by default, as circled in the following illustration.



You can also toggle whether code snippets, namespaces, or element tags appear in IntelliSense.

## For More Information

Check out the developer's blog post for this extension:

<http://karlshifflett.wordpress.com/2010/03/21/visual-studio-2010-xaml-editor-intellisense-presenter-extension>.

## 08.05 Sync the Solution Explorer to the Current File

NAME	Solution Explorer Tools
CREATED BY	Chris McGraph
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/ef4ac3e9-d056-4383-8ca2-11721bd879b4">http://visualstudiogallery.msdn.microsoft.com/ef4ac3e9-d056-4383-8ca2-11721bd879b4</a>

The Solution Explorer Tools extension provides you with greater control over how and when the Solution Explorer syncs to the currently opened document.

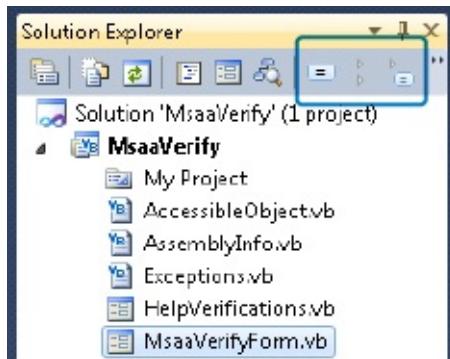
### Solution Explorer Tools

The Solution Explorer Tools extension provides additional features on the Solution Explorer toolbar for controlling the opened solution.

### To Use

For this extension to work properly, you must *uncheck* the Track Active Item in Solution Explorer option found on the Tools | Options | Projects and Solutions | General page.

When the Tools extension is installed, several new buttons appear on the Solution Explorer toolbar, as shown in the following illustration.



The following list describes the buttons from left to right:

- **Sync Item**—This command places the active selection in the Solution Explorer to the currently opened document.
- **Collapse All**—This command recursively collapses everything in the Solution

Explorer.

- **Collapse to Item**—This command combines the two previous commands by placing the active selection in the Solution Explorer to the currently opened item and collapsing everything else in the Solution Explorer.

You can also assign keyboard shortcuts to each of the following commands on the Tools | Options | Environment | Keyboard page:

- Project.SyncItem
- Project.Collapseall
- Project.CollapseToItem

## 08.06 Add PowerCommands Options to the IDE

NAME	PowerCommands
CREATED BY	Microsoft
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/en-us/e5f41ad9-4edc-4912-bca3-91147db95b99">http://visualstudiogallery.msdn.microsoft.com/en-us/e5f41ad9-4edc-4912-bca3-91147db95b99</a>

The PowerCommands extension provides additional tweaks for customizing the IDE.

### PowerCommands for Visual Studio 2010

The PowerCommands have been a popular add-in for the past several previous versions of Visual Studio.

#### To Use

The PowerCommands extension provides additional options for customizing the IDE. These options are found on the Tools | Options | PowerCommands | General and Commands pages.

The following is a list of each of the PowerCommands options to help you discover all the ways you can further customize the IDE:

- **Format Document On Save** —Saving the file automatically formats the document.
- **Remove And Sort Usings On Save** —Saving the file automatically removes any unused using statements. This option automatically sorts all using statements in alphabetical order.
- **Remove and Sort Usings** —This command can be found on the Solution Explorer context menu at both the Solution level and at the individual project levels. It performs the same functionality as the previous command.
- **Clear All Panes** —This command clears all Output Window subwindows at once. The Clear All Panes button is located just to the right of the Show Output From drop-down list on the Output Window toolbar.
- **Copy Path** —This command copies the full file path of the currently selected item in the Solution Explorer to the clipboard. It works with the solution node, a project node, any project item node, and any folder.

- **Email Code Snippet** —Select code in the editor, right-click to open the context menu, and select Email Code Snippet to email the code snippet. The subject of the generated email is based on the file name—for example, “Subject: CodeSnippet from Program.cs”.
- **Insert Guid Attribute** —To add a GUID attribute to a class, right-click anywhere within the class and select Insert Guid Attribute from the context menu.
- **Show All Files** This option places an additional Show All Files button on the Solution Explorer toolbar. The leftmost Show All Files button is for the current project, and the rightmost Show All Files button is for the entire solution.
- **Undo Close** To reopen the most-recently closed document, go to Edit | Undo Close. The keyboard shortcut is Ctrl+Shift+Z. The cursor is placed at its last known position. To see a list of all the most-recently closed documents, go to View | Other Windows | Undo Close Window.
- **Collapse Projects** To collapse everything under a project node, including any sub-nodes (this being the key difference from the standard Windows behavior), right-click and select Collapse Project from the Solution Explorer context menu. This command works for solutions, projects, and solution folders.
- **Copy Class / Paste Class** To copy and paste a class, select a project item (usually a class file) on the Solution Explorer that you want to copy. Right-click and select Copy Class from the context menu. Navigate to the project node, and right-click to see the Paste Class command.
- **Copy As Project Reference** Found on the Solution Explorer context menu, this command copies a project (from its project node) and pastes it into another project node as a project reference.
- **Edit Project File** The Edit Project File option unloads a project and automatically opens the file in the editor for editing.
- **Open Containing Folder** Similar to the command found on the file tabs and in the Solution Explorer for a node, this command opens the containing folder for an item in the Solution Explorer.
- **Open Command Prompt** This option opens a Visual Studio command prompt directly from the Solution Explorer.
- **Unload Projects / Reload Projects** Similar to the built-in command for unloading a project, the Unload Projects command (found on the Solution Explorer context menu for a project node) unloads all the projects in a

solution.

- **Extract Constant** Found under the Refactor menu, the Extract Constant creates a constant based on the selected text.
- **Clear Recent File List** This command removes specified recent files. In past versions of Visual Studio, this clearing required manual modifications to the registry. The command is found on the File | Recent Files menu as Clear Recent File List. This command opens a window for you to specify which files to clear.
- **Clear Recent Project List** Similar to the Clear Recent File List, this command removes projects specified by the user.
- **Close All** Found on the file tab context menu, this command closes all open documents.

## For More Information

For more information about the PowerCommands extension, you can read the following blog post: <http://saraford.net/2010/09/07/power-commands-for-visual-studio-2010-extension-8>.

## 08.07 Use Emacs Commands in the Editor

NAME	Emacs Emulation
CREATED BY	Microsoft
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/en-us/09dc58c4-6f47-413a-9176-742be7463f92">http://visualstudiogallery.msdn.microsoft.com/en-us/09dc58c4-6f47-413a-9176-742be7463f92</a>

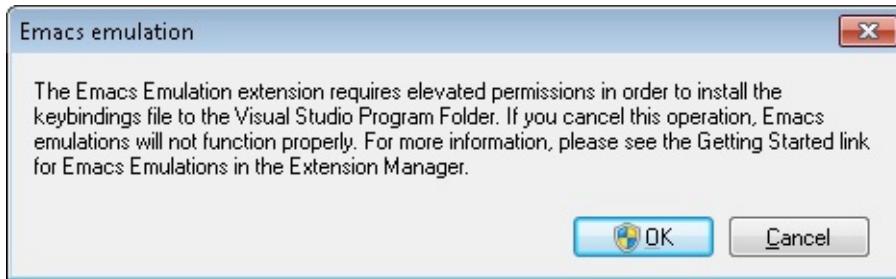
The Emacs Commands extension provides basic support for Emacs keybindings and text editing commands.

### Emacs Emulation

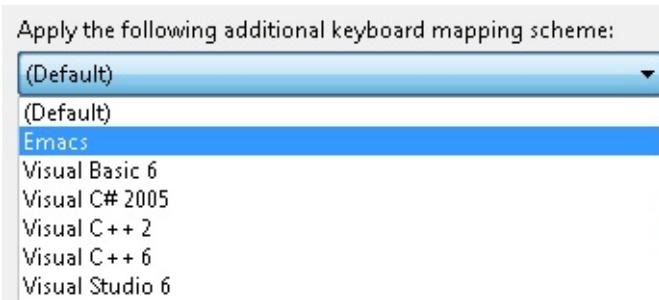
This extension brings back the Emacs commands for the Visual Studio 2010 editor.

### To Use

To finish installing the extension, you must open a project or create a new project. Opening a project prompts you for elevated permissions to finish installing the Emacs.vsk file (the Emacs keyboard shortcut file) to the Visual Studio IDE folder located under Program Files directory. Elevated permissions are required because only user accounts with administrator privileges can reach this folder by default in Windows.



Select **Emacs** in the Tools | Options | Environment | Keyboard page's scheme drop-down list to enable the Emacs emulation experience.



## To Uninstall

When you disable or uninstall the extension, the Emacs option remains selected in the Tools | Options | Keyboard list. You must go back to the Tools | Options | Environment | Keyboard page and select the “(Default)” key bindings or your preferred keyboard mapping scheme.

If you forget to change the keyboard mapping scheme, you’ll notice a great many commands and keyboard shortcuts not working in the editor, such as Ctrl+F for the Find window.

Because administrator rights were required to add this file, you also need administrator rights to remove it. Unlike the install scenario for opening or creating a project, there isn’t an opportunity for the extension to prompt you to remove the .vsk file. For example, the extension cannot prompt you the next time you open a project (as in the install scenario) because the extension itself is already gone.

You can manually delete the Emacs.vsk file at your own risk, but there is no impact to Visual Studio if you decide to keep the file.

## More Information

You can find more information about all the Emacs keyboard shortcuts supported by the extension at the following location:

<http://visualstudiogallery.msdn.microsoft.com/en-us/09dc58c4-6f47-413a-9176-742be7463f92>.

## 08.08 Submit to “The Daily WTF”

NAME	The Daily WTF
CREATED BY	Inedo.com
LOCATION	<a href="http://inedo.com/Downloads/SubmitToWTF.aspx">http://inedo.com/Downloads/SubmitToWTF.aspx</a>
SOURCE CODE LOCATION	<a href="http://code.google.com/p/submittotdwtf">http://code.google.com/p/submittotdwtf</a>
VERSIONS	2005, 2008, 2010

Have you ever inherited source code so poorly written that you’ve asked a colleague to come into your office to take a look? If so, this extension is for you.

### Share Bad Code with the World

Started in May 2004, “The Daily WTF” (also known as “The Daily Worse Than Failure”) is a blog dedicated to poorly written code, poor project management decisions, and bizarre interview stories.

This extension allows you to submit a code snippet directly from your code editor to “The Daily WTF.” It is the equivalent of using the website’s contact form to submit a code suggestion.

It is a little-known fact that Sara Ford’s writing style for the “Visual Studio 2008 Tip of the Day” series was inspired by Alex Papadimoulis’ writing style for “The Daily WTF.” The idea of providing personal commentary alongside factual information was very appealing for a daily tip series. (Thank you, Alex!)

### To Install

At the time of this writing, this extension is not located on the Visual Studio Gallery.

You can download the extension from <http://inedo.com/Downloads/SubmitToWTF.aspx>. Links for both Visual Studio 2010 and Visual Studio 2005/2008 appear on the page. After you have downloaded the extension, unzip the folder and double-click the SubmitToWTF2010.vsix file.

### To Use

Select code within a code editor, and open the context menu to see the Submit To

TDWTF command appear. Click the Submit To TDWTF command to open a dialog box that requests additional information about the code snippet. Additionally, there is an option to request that the code snippet is not published.

## More Information

You can find more information about The Daily WTF at <http://thedailywtf.com>.

## 08.09 Diff Files Using the Editor

NAME	CodeCompare
CREATED BY	Devart Software
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/dace3633-0b51-4629-85d4-c59cdce5bb3b">http://visualstudiogallery.msdn.microsoft.com/dace3633-0b51-4629-85d4-c59cdce5bb3b</a>
VERSIONS	2008, 2010

You can compare code side-by-side within the Visual Studio Editor by using the CodeCompare extension.

### CodeCompare

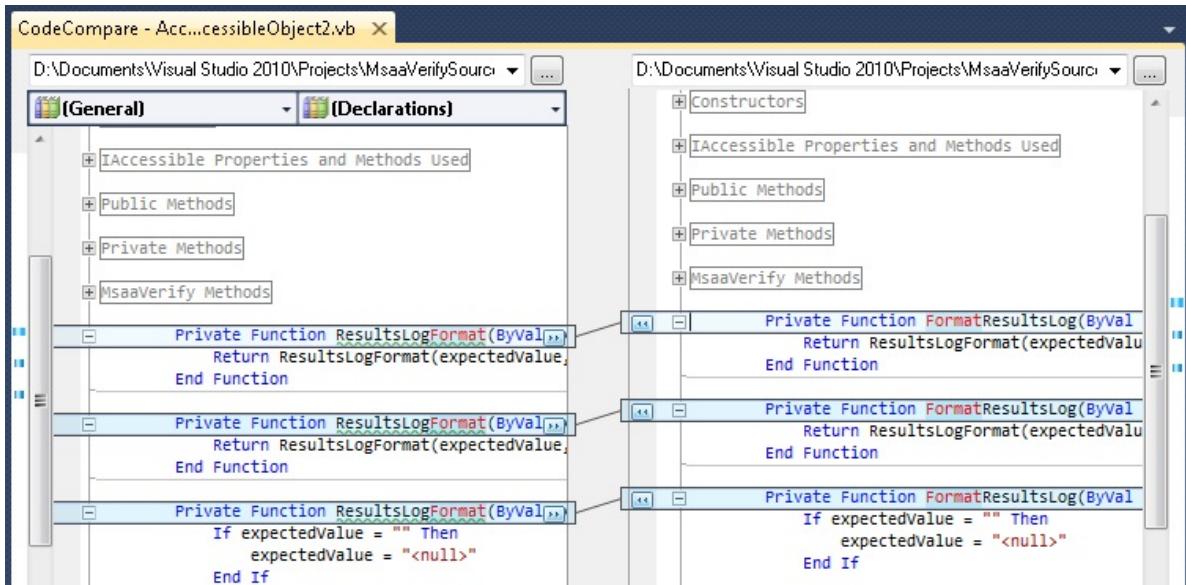
The CodeCompare tool uses the Visual Studio code editor to compare two files side-by-side, allowing you to continue using all the Visual Studio code editor functionality, like syntax coloring, IntelliSense, Ctrl+Mouse Wheel zooming, and so on.

### To Install

The CodeCompare tool does not use the new Visual Studio Extensibility model, so it is not found in the Extension Manager. To install, run the executable file hosted on the Visual Studio Gallery and follow the instructions in the setup wizard.

### To Use

To start the CodeCompare tool, go to Tools | CodeCompare | New Comparison. This action opens a new code comparison view alongside your code files using the Visual Studio code editor. Simply select two different files to compare to start seeing a visual representation of the differences between the files.



## Features

CodeCompare is a fully functional code comparing tool, but a few features are worth mentioning, including the following:

- **Merge Code** Move code from one comparison view to the other. Each indicated line that is different per view has a << or a >> symbol that illustrates it is possible to merge that line or block of code to the other file.
- **Structure Comparison** Compare code by its structure, namely its classes, fields, and methods. This comparison is shown in the Difference Explorer. Oddly enough, in the free version of CodeCompare, attempting to click within the Difference Explorer clears its contents. The cursor focus must remain within the code editor for the Difference Explorer to show the structural differences between the code files.
- **New Folder Comparison** Compare code at the file level. Select two folders, and the compare tool shows you which files are different or not included. Double-click a file to do a code comparison at the file level in a new window.

## To Uninstall

Go to Windows Control Panel | Uninstall A Program window, and select the Devart CodeCompare application. Click the Uninstall command on the menu to uninstall CodeCompare.

## More Information

You can find more information about CodeCompare at  
<http://www.devart.com/codecompare>.

## 08.10 Run Windows PowerShell Within the IDE

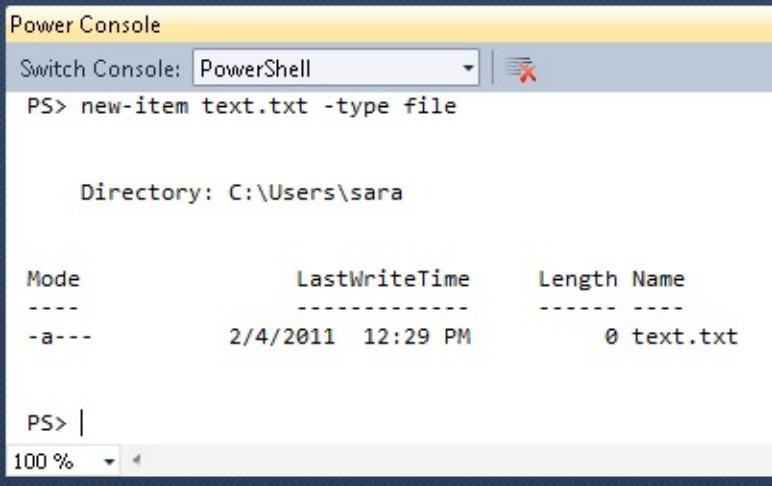
NAME	PowerConsole
CREATED BY	Jianchun Xu (Microsoft)
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/67620d8c-93dd-4e57-aa86-c9404acbd7b3">http://visualstudiogallery.msdn.microsoft.com/67620d8c-93dd-4e57-aa86-c9404acbd7b3</a>
VERSIONS	2010

You can use the PowerConsole extension to run both Windows PowerShell commands and Visual Studio object model (DTE) commands within the IDE.

### To Use

To open the Power Console tool window, go to View | Other Windows | Power Console.

The Power Console allows you to run more than just Windows PowerShell commands. Additionally, you can call the Visual Studio object model (DTE), access Visual Studio services, interact with Visual Studio MEF components, and host your own scripting language within the Power Console.



A screenshot of the Power Console tool window. The title bar says "Power Console". The toolbar has a dropdown labeled "Switch Console: PowerShell" and a close button. The main area shows a PowerShell command being run:

```
PS> new-item text.txt -type file
```

Output shows the creation of a file:

```
Directory: C:\Users\sara

Mode          LastWriteTime    Length Name
----          -----        ---- 
-a---  2/4/2011 12:29 PM      0 text.txt
```

The bottom status bar shows "PS> |" and "100 %".

### More Information

You can find more information about the Power Console at  
<http://visualstudiogallery.msdn.microsoft.com/67620d8c-93dd-4e57-aa86-c9404acbd7b3>.

## 08.11 Visualize OData in a Graphical View

NAME	Open Data Protocol Visualizer
CREATED BY	Microsoft
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/f4ac856a-796e-4d78-9a3d-0120d8137722">http://visualstudiogallery.msdn.microsoft.com/f4ac856a-796e-4d78-9a3d-0120d8137722</a>

You can use the Open Data Protocol Visualizer extension to get a graphical view of an OData service.

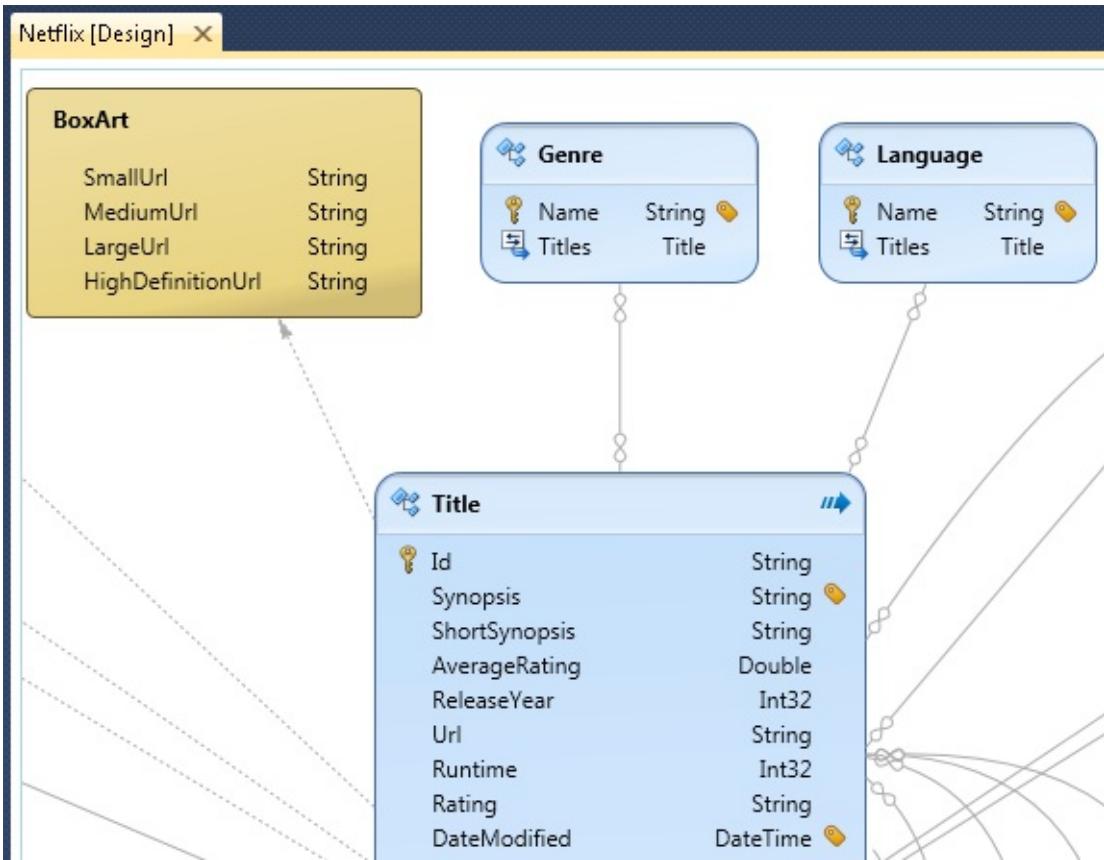
### Open Data Protocol Visualizer

The Open Data Protocol Visualizer provides a read-only graphical view of the types and relationships provided by a WCF Data Service.

#### To Use

Create a service reference through the Add Service Reference dialog box. Open the context menu for the service reference in the Solution Explorer, and select the new View In Diagram command to open the visualizer.

At the bottom of the visualizer, select the elements you want to view, or select “all” to visualize everything. You can also browse elements by using the Open Data Protocol Model Browser found under the View menu.



## More Information

You can find a video explaining how to use the Open Data Protocol Visualizer extension located at

<http://odataprimer.com/ODataVisualizerExtensionForVS2010Screencast.ashx>.

You can also find a list of OData services at <http://www.odata.org/producers>.

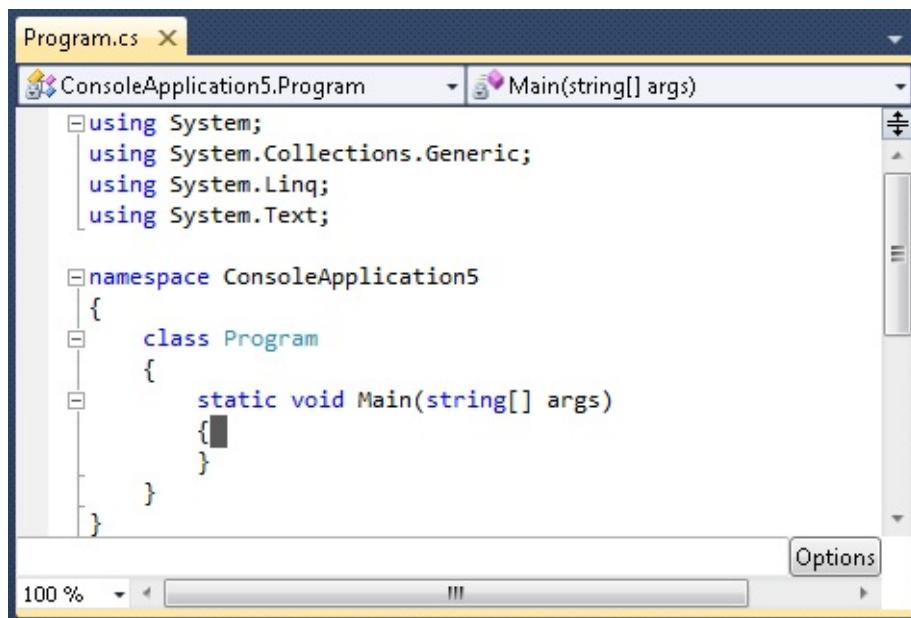
## 08.12 Run VIM Commands in the Editor

NAME	VsVim
CREATED BY	Jared Parsons (Microsoft)
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/59ca71b3-a4a3-46ca-8fe1-0e90e3f79329">http://visualstudiogallery.msdn.microsoft.com/59ca71b3-a4a3-46ca-8fe1-0e90e3f79329</a>
SOURCE CODE	<a href="http://github.com/jaredpar/VsVim">http://github.com/jaredpar/VsVim</a>

The VsVim extension provides a Vim editing experience within the Visual Studio editor.

### To Use

When enabled, the VIM extension immediately prompts you at the top of the editor to customize how many Visual Studio keyboard shortcut defaults you want to use. You can make additional modifications by clicking the Options button at the bottom of the editor next to the VIM extension status bar, as shown in the following illustration.



### More Information

You can find a frequently asked question list for the VIM extension located at <https://github.com/jaredpar/VsVim/wiki/faq>.

## 08.13 Check Spelling in Your Code

NAME	Spell Checker
CREATED BY	Noah Richards with Roman Golovin and Michael Lehenbauer
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/7c8341f1-ebac-40c8-92c2-476db8d523ce">http://visualstudiogallery.msdn.microsoft.com/7c8341f1-ebac-40c8-92c2-476db8d523ce</a>

You can check for spelling errors in your code by using the Spell Checker extension.

### Spell Checker

The Spell Checker extension provides a spell checker within the Visual Studio editor.

### To Use

The Spell Checker extension provides spelling corrections for plain text files, comments and strings in source code, and text that isn't included within HTML tags in .html and .aspx files.

The Spell Checker uses Smart Tags, meaning that you can simply press **Ctrl+.** to invoke the Smart Tag window.

If you want to change the color of the red squiggles so that spelling errors are not confused with syntax errors, you can go to the Tools | Options | Environment | Fonts And Colors page, and under Display Items, select Spelling Error and change the Item Foreground color to the appropriate color.



## 08.14 Zoom Across All Files

NAME	Presentation Zoom
CREATED BY	Chris Granger (Microsoft)
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/6a7a0b57-7059-470d-bcfa-60ceb78dc752">http://visualstudiogallery.msdn.microsoft.com/6a7a0b57-7059-470d-bcfa-60ceb78dc752</a>

You can maintain a consistent zoom percentage across all open files using the Presentation Zoom extension.

### Presentation Zoom

By default, the new Presentation Zoom extension in Visual Studio 2010 works only for the current file. Additional files are not affected by the zoom. This is especially noticeable when giving presentations, where the presenter has to zoom each file separately.

This extension provides a global zoom level, so all open files can share the same zoom level.

### To Use

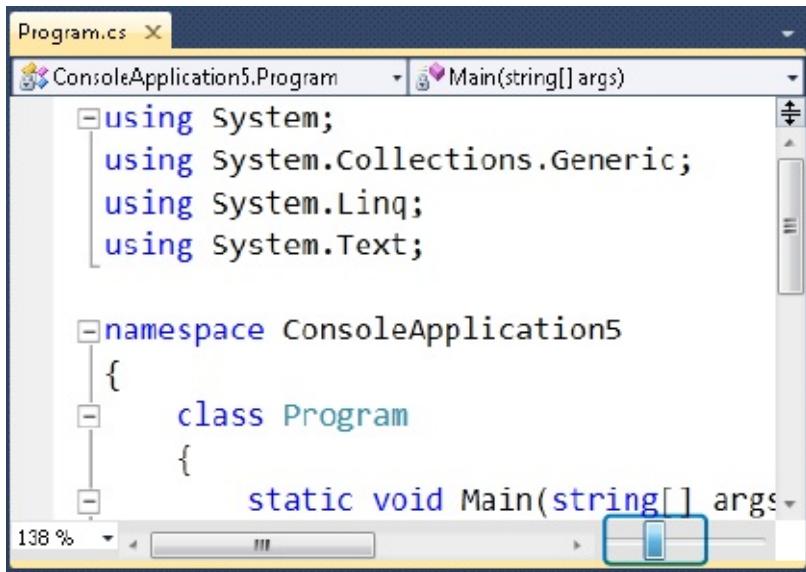
Simply use the zoom feature, either by Ctrl+Mouse wheel or updating the zoom percentage at the bottom-left corner of the editor. The next file opened or navigated to persists the same zoom percentage.

### Control Zooming with a Slider Using the ZoomEditorMargin Extension

NAME	ZoomEditorMargin
CREATED BY	Benjamin Gopp
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/c271d574-a481-4974-b7dd-f319404de898">http://visualstudiogallery.msdn.microsoft.com/c271d574-a481-4974-b7dd-f319404de898</a>

Similar to the Presentation Zoom extension discussed in the preceding section, the ZoomEditorMargin extension provides a slider control in the bottom-right corner of the editor for specifying the zoom percentage for the file in view. The

ZoomEditorMargin extension works in conjunction with the Presentation Zoom extension.



A screenshot of a code editor window titled "Program.cs X". The window shows the following C# code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
    }
}
```

The code editor has a vertical zoom control on the right side. The status bar at the bottom left indicates a zoom level of 138%.

## 08.15 View Code Blocks Using Vertical Lines

NAME	StructureAdornment
CREATED BY	David Pugh (Microsoft)
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/203f22f4-3e9f-4dbb-befc-f2606835834e">http://visualstudiogallery.msdn.microsoft.com/203f22f4-3e9f-4dbb-befc-f2606835834e</a>

The StructureAdornment extension displays vertical lines in the editor to show the block structure of the code file.

### StructureAdornment

The StructureAdornment extension uses different colors to indicate different blocks.

```
#Region "Public Methods"

    ' get the control's hwnd
    Public ReadOnly Property Hwnd() As IntPtr
        Get
            If Me.m_hwnd.ToInt32 = 0 Then
                Dim handle As IntPtr

                ' get a handle from the accessible object we have
                NativeMethods.WindowFromAccessibleObject(m_msaa, handle)

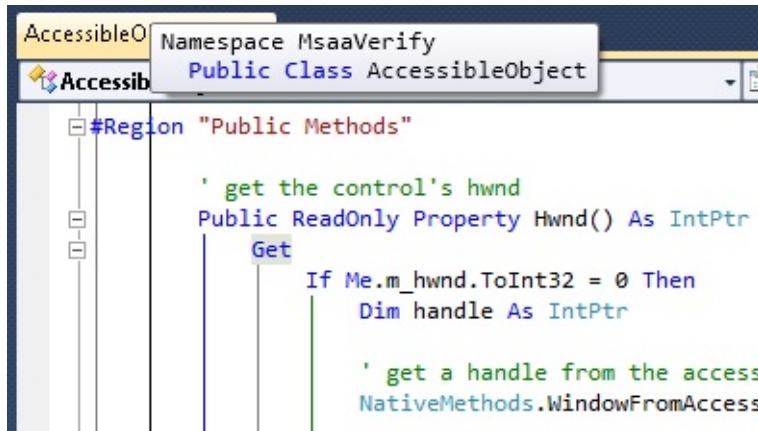
                ' create an accessible object
                Dim msaa As New AccessibleObject(Me.IAccessible)

                ' cache the hwnd for future use
                m_hwnd = handle

                Return handle
            Else
                Return Me.m_hwnd
            End If
        End Get
    End Property
```

### To Use

By default, vertical lines appear in the editor. To see the beginning of a particular code block that is outside the view, mouse-over the vertical line to display the beginning of that particular code block.



## To Customize

This extension does not have a page in Tools | Options; however, you can manually edit the registry settings for the extension. The following editor options are stored in the registry under  
HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\10.0\Text Editor:

- **StructureAdornment/LineWidth** Width of the lines used to show the various scopes.
- **StructureAdornment/ClassColor** Color used to show the scope of class definitions as ARGB.
- **StructureAdornment/ConditionalColor** Color used to show the scope of conditionally executed code as ARGB.
- **StructureAdornment/LoopColor** Color used to show the scope of loop bodies as ARGB.
- **StructureAdornment/MethodColor** Color used to show the scope of method bodies as ARGB.
- **StructureAdornment/UnknownColor** Color used to show the scope of unknown blocks as ARGB.
- **StructureAdornment/MethodSeparatorColor** Color used draw a horizontal separator line at the end of a method as ARGB (off by default).
- **StructureAdornment/Enabled** Show structure adornments.

Modifications made to the extension appear the next time a code editor is opened.

## To Uninstall

This extension installs five separate extensions. To completely remove this

extension, you need to remove the following extensions:

- BlockTagger
- BlockTaggerImpl
- SettingsStore
- SettingsStoreImpl
- StructureAdornment

You can uninstall all five extensions at the same time within the Extension Manager. If prompted about a Dependence Alert, click Uninstall Anyways to continue uninstalling all extensions.

## 08.16 Get a Bird's-Eye View of Your Code in an Editor Margin

NAME	AllMargins
CREATED BY	David Pugh (Microsoft)
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/465a0d53-5133-4edd-a0cd-94484fe3d853">http://visualstudiogallery.msdn.microsoft.com/465a0d53-5133-4edd-a0cd-94484fe3d853</a>

The AllMargins extension provides a complete overview of your code to improve navigation.

### AllMargins

The AllMargins extension installs multiple extensions. The two most notable are the OverviewMargin extension, a bird's-eye view of the logical layout of your code all within an editor margin, and the StructureAdornment extension, which displays vertical lines in the editor, representing structural code blocks.

See the extension [08.15 View Code Blocks Using Vertical Lines](#) for more information about the StructureAdornment extension.

### To Use

By default, both the StructureAdornment extension and the OverviewMargin extension are visible in the editor. The OverviewMargin appears on the right side of the editor.

A screenshot of the Visual Studio IDE showing the code editor for a file named "AccessibleObject.vb". The window title is "AccessibleObject.vb". The code is for a class named "AccessibleObject" with a property "Hwnd". The "Get" block of the property contains several lines of code. The left side of the code editor has a vertical margin labeled "OverviewMargin" which displays a code snippet for the selected section of code. In this case, the snippet shows the code for the "Get" block of the "Hwnd" property.

```
' get the control's hwnd
Public ReadOnly Property Hwnd() As IntPtr
    Get
        If Me.m_hwnd.ToInt32 = 0 Then
            Dim handle As IntPtr

            ' get a handle from the accessible object
            NativeMethods.WindowFromAccessibleObject(Me.IAccObject, handle)

            ' create an accessible object
            Dim msaa As New AccessibleObject(Me.IAccObject, handle)

            ' cache the hwnd for future use
            m_hwnd = handle

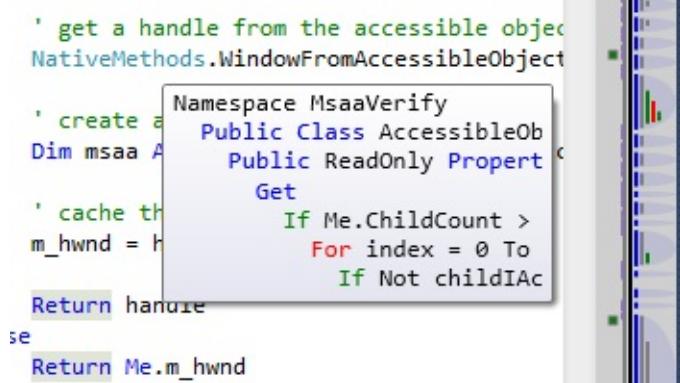
            Return handle
        Else
            Return Me.m_hwnd
        End If
    End Get
End Property
```

Mouse-over the left side of the OverviewMargin to see a code snippet for that particular section of code.

A screenshot of the Visual Studio IDE showing the code editor for a file named "AccessibleObject.vb". The window title is "AccessibleObject.vb". The code is for a class named "AccessibleObject" with a property "Hwnd". The "Get" block of the property contains several lines of code. The right side of the code editor has a vertical margin labeled "OverviewMargin" which displays a tooltip containing a condensed form of the structural code block. In this case, the tooltip shows the "Try", "Catch", and "End Try" blocks of the "If" statement.

```
' get the control's hwnd
Public ReadOnly Property Hwnd() As IntPtr
    Get
        If Me.m_hwnd.ToInt32 = 0 Then
            Dim handle
            Try
                Dim x0, y0, w, h As Integer
                m_msaa.accLocate(handle, 0, 0, x0, y0, w, h)
                Return y0
            Catch
                Return 0
            End Try
        Else
            Return Me.m_hwnd
        End If
    End Get
End Property
```

Mouse-over the right side of the OverviewMargin to see the structural code block in a condensed form.



## To Uninstall

This extension installs 12 separate extensions. To completely remove this extension, you need to remove the following:

- AllMargins
- BlockTagger
- BlockTaggerImpl
- CaretMargin
- ErrorsToMarks
- MarkersToMarks
- OverviewMargin
- OverviewMarginImpl
- SettingsStore
- SettingsStoreImpl
- StructureAdornment
- StructureMargin

You can uninstall all 12 extensions at the same time within the Extension Manager. If prompted about a Dependence Alert, click Uninstall Anyways to continue uninstalling all extensions.

## 08.17 Build Projects from the Windows 7 Taskbar

NAME	Win7 Taskbar Extension
CREATED BY	Dmitry Sitnikov
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/0c92dd87-50ac-489e-882b-b99de7624502">http://visualstudiogallery.msdn.microsoft.com/0c92dd87-50ac-489e-882b-b99de7624502</a>

You can use the Win7 Taskbar Extension to start a build or debug session directly from the Windows 7 taskbar.

### Win7 Taskbar Extension

The Win7 Taskbar Extension allows you to start a build or debug session for the specified Visual Studio application directly from the Windows 7 taskbar. This extension can save you time when you need to build different projects located in multiple instances of Visual Studio.

### To Install

The Win7 Taskbar Extension tool is an extension of the Windows 7 taskbar, so it is not found in the Extension Manager. To install, run the executable file hosted on the Visual Studio Gallery location and follow the instructions in the setup wizard.

### To Use

Mouse-over any Visual Studio window in the Windows 7 taskbar to see the extension appear. Simply click the corresponding button for Build, Start Debugging, or Start Without Debugging.



## To Uninstall

Open the Windows Control Panel | Uninstall a Program window, and select the Visual Studio Win7 Taskbar Add-in application. Click the Uninstall command listed on the menu to remove the extension.

## 08.18 Triple-Click to Select an Entire Line

NAME	Triple Click
CREATED BY	Noah Richards
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/2bbdc70c-32f7-4b69-8cff-d8190cae0cc7">http://visualstudiogallery.msdn.microsoft.com/2bbdc70c-32f7-4b69-8cff-d8190cae0cc7</a>
SOURCE CODE	<a href="https://github.com/NoahRic/TripleClick">https://github.com/NoahRic/TripleClick</a>

You can use the Triple Click extension to quickly select a line of code.

### Triple Click

The Triple Click extension selects an entire line of code when the left mouse button is triple-clicked.

```
' get the control's hwnd
Public ReadOnly Property Hwnd() As IntPtr
    Get
        If Me.m_hwnd.ToInt32 = 0 Then
            Dim handle As IntPtr

            ' get a handle from the accessible object we have
            NativeMethods.WindowFromAccessibleObject(m_msaa, handle)

            ' create an accessible object
            Dim msaa As New AccessibleObject(Me.IAccessible)

            ' cache the hwnd for future use
            m_hwnd = handle

            Return handle
        Else
            Return Me.m_hwnd
        End If
    End Get
End Property
```

### More Information

For more information, please see the extension author's blog post at <http://blogs.msdn.com/b/noahric/archive/2009/10/19/beta-2.aspx>.

## 08.19 Create Regular Expressions Within Your Code

NAME	Regex Editor
CREATED BY	Microsoft
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/55c24bf1-2636-4f94-831d-28db8505ce00">http://visualstudiogallery.msdn.microsoft.com/55c24bf1-2636-4f94-831d-28db8505ce00</a>
SOURCE CODE	<a href="http://editorsamples.codeplex.com">http://editorsamples.codeplex.com</a>

The Regex Editor helps you write regular expressions faster and easier.

### Regex Editor

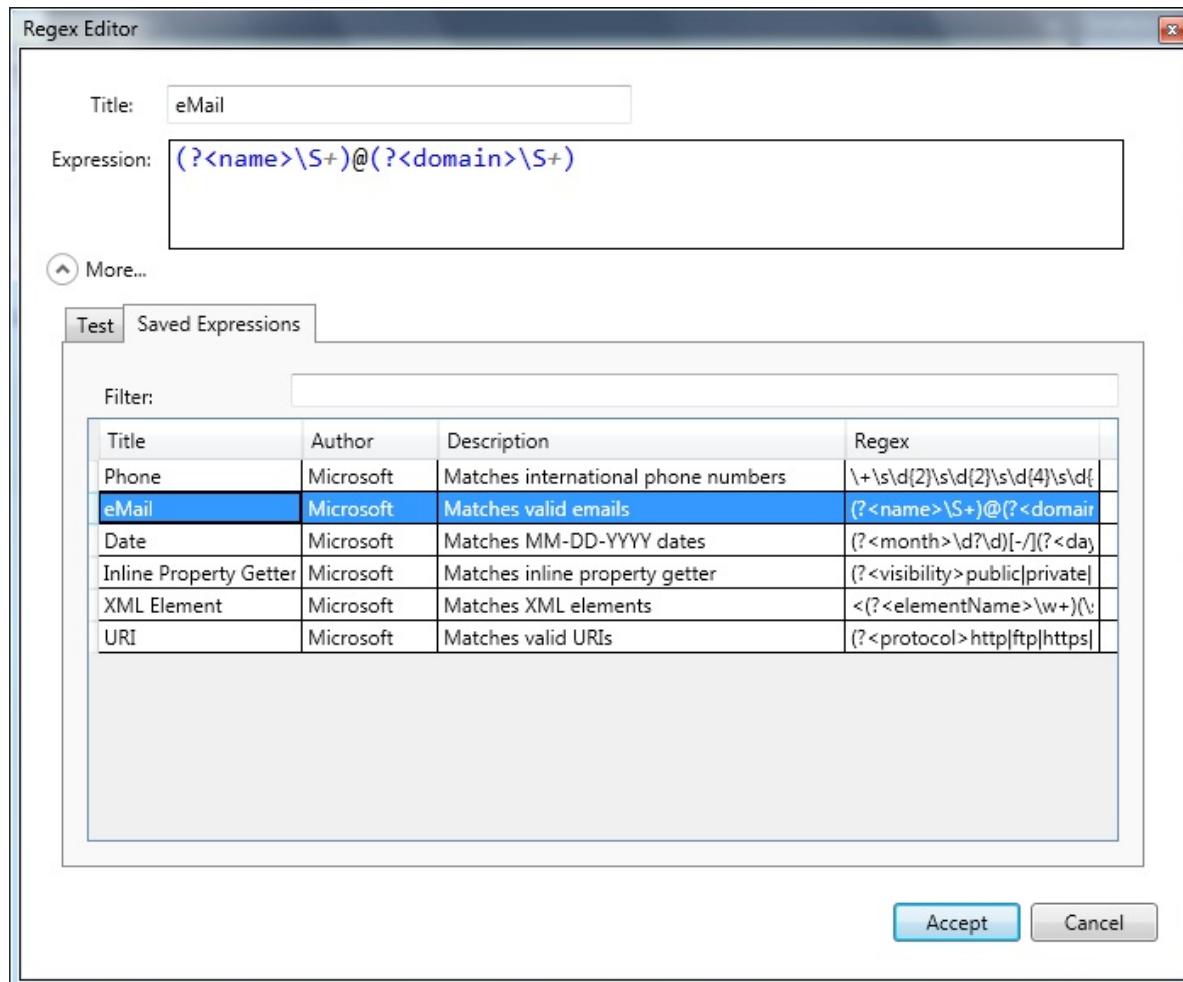
The Regex Editor extension is an aid for creating, testing, and saving regular expressions.

### To Use

To invoke the Regex Editor window, you need to create a new Regex class within your code. For example, type in the following:

```
Regex r = new Regex(
```

This opens the Regex Editor window, shown in the following illustration.



## More Information

For more information, please visit the project's homepage at  
<http://editorsamples.codeplex.com>.

## 08.20 Get More Productivity Tools in the IDE

NAME	Visual Studio Productivity Power Tools
CREATED BY	Microsoft
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/d0d33361-18e2-46c0-8ff2-4adea1e34fef">http://visualstudiogallery.msdn.microsoft.com/d0d33361-18e2-46c0-8ff2-4adea1e34fef</a>

The Visual Studio Productivity Power Tools are a collection of extensions intended to maximize a developer's productivity.

### To Use

The tools include a variety of IDE productivity tweaks, such as the following:

- **Tools Options Support** Allows for all of the included tools to be turned on or off by selecting Tools | Options | Productivity Power Tools.
- **Document Tab Well User Interface** Customize the look and functionality of open Visual Studio tabs.
- **Searchable Add Reference Dialog** Replace the default Add Reference Dialog with a dialog box that includes a substring search of the current pane.
- **Quick Access** Search and execute common Visual Studio tasks.
- **Auto Brace Completion** Improve code productivity by automatically inserting the matching closing code construct for the following characters: (), {}, [], “”, and ”.
- **Highlight Current Line** Highlight the line in the code editor where the caret is located.
- **HTML Copy** Copy code to the Windows Clipboard using the HTML Clipboard format, which makes it easy to retain code formatting.
- **Triple Click** Triple-click anywhere on a line to select the entire line.
- **Fix Mixed Tabs** Fix files that have a mix between tabs and spaces.
- **Ctrl+Click Go to Definition** Jump quickly to a symbol definition via Ctrl+Click.
- **Align Assignments** Increase code readability by aligning variable assignments. To use, press Ctrl+Alt+J.
- **Move Line Up/Down Commands** Hold down the Alt key while tapping the

Up/Down arrow to move the current line of code up or down.

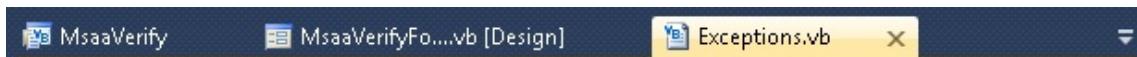
- **Column Guides** Everyone's favorite Visual Studio registry modification, adding a vertical line guide to the code editor, can be set via the code editor context menu.
- **Colorized Parameter Help** Add syntax color coding to Parameter Help.

## To Customize the Document Tab Well User Interface

By default, the extension changes the Document Tab Well by coloring and grouping tabs based on their project, which is helpful when working with large solutions. You can customize these default settings by selecting Tools | Options | Productivity Power Tools | Document Tab Well.

Several preset configurations are available, including the following: Visual Studio 2008, Visual Studio 2010, Web Browser, Scrollable Tab Well, Vertical Tab Well, and Dynamic Tab Well.

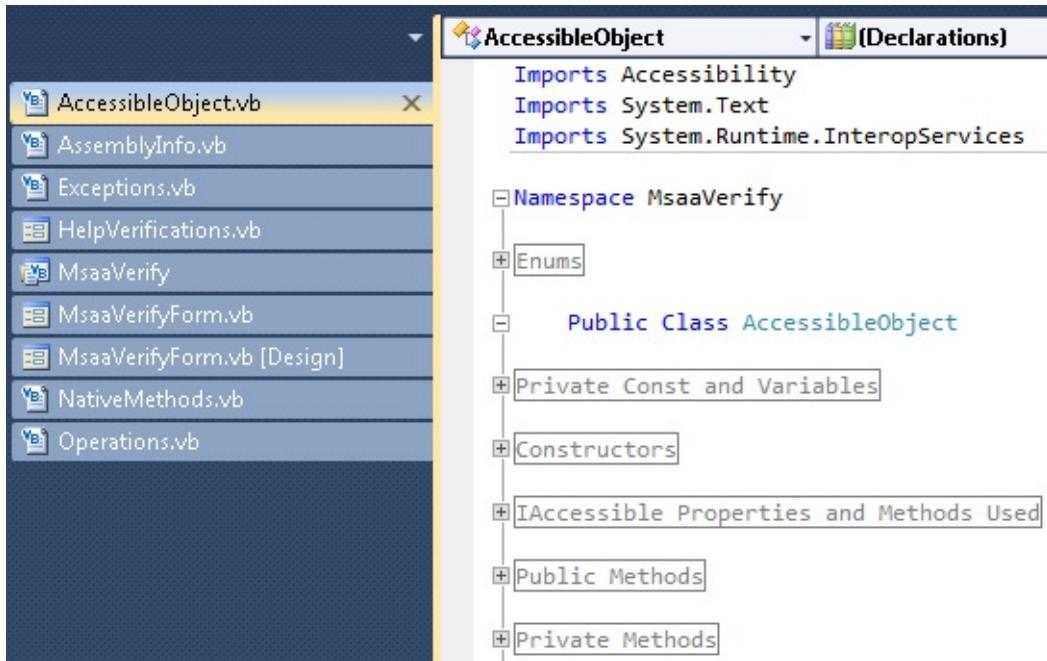
- The Visual Studio 2008 and 2010 configurations act exactly as their names imply, causing the default tab well to behave like those versions of the IDE.
- Both the Web Browser and Dynamic Tab Wells add file type icons to what would otherwise be a typical Visual Studio 2010 tab. The difference between the two is that the Dynamic Tab Well closes the least-recently used tab when there is no more space available in the file tab channel.



- The Scrollable Tab Well is the default for the extension.



- The Vertical Tab Well has the same characteristics as the Scrollable Tab Well but places the tabs vertically, allowing for more viewable tabs.



It is possible to tweak all of the configurations. The ability to *pin tabs* is probably the most useful customization. For example, you probably often want to view the contents of many files while actually working on only one or two. The ability to pin specific tabs that you are focusing on, to keep them from falling out of focus, is a huge time-saver. When you pin many tabs, you can use the Show Pinned Tabs In A Separate Row/Column option. This keeps the pinned tabs from taking up too much space.



The biggest tab productivity boost comes in the form of added keyboard navigation. In Windows, Ctrl+Tab switches to the next child window for a given program. It's known that many users do not prefer the Ctrl+Tab window that appears in Visual Studio. The Productivity Power Tools extension offers the option of moving backward and forward in visual order via Ctrl+Alt+Page Down/Up. Even more exciting, you can jump to a pinned tab via Ctrl+Num Pad 1 through 0. Talk about keyboard efficiency!

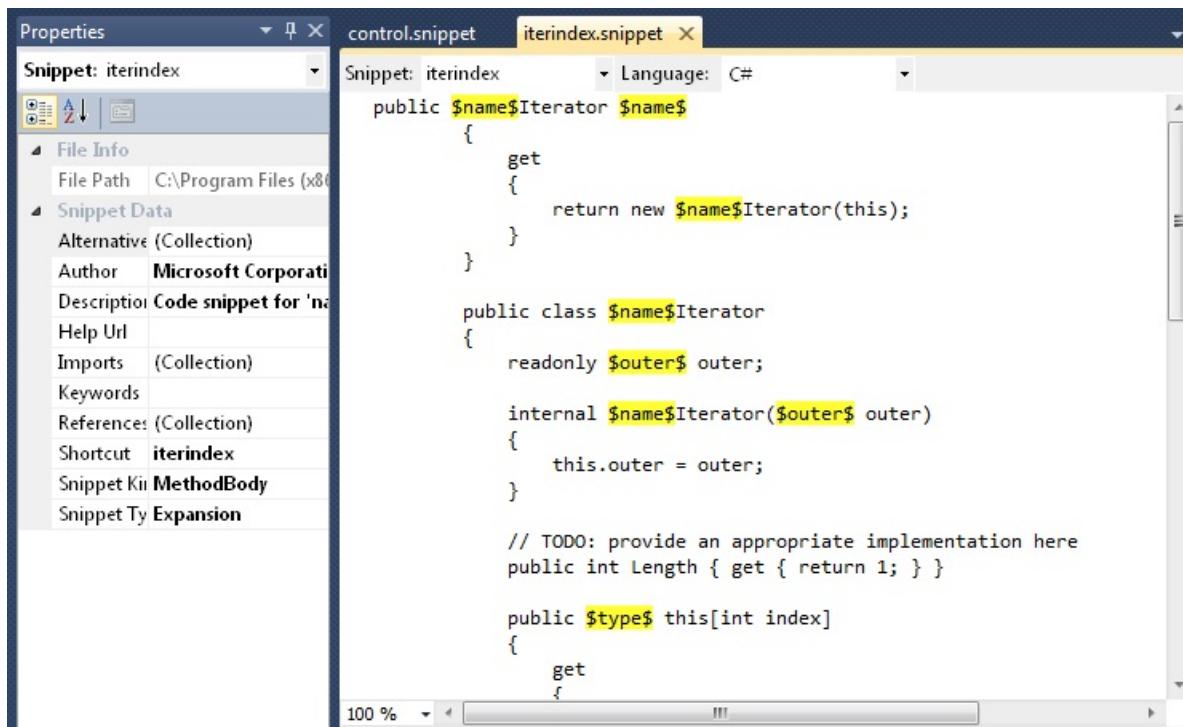
## 08.21 Create and Find Code Snippets

NAME	Snippet Designer
CREATED BY	Matt Manela
LOCATION	<a href="http://visualstudiogallery.msdn.microsoft.com/B08B0375-139E-41D7-AF9B-FAEE50F68392">http://visualstudiogallery.msdn.microsoft.com/B08B0375-139E-41D7-AF9B-FAEE50F68392</a>

The Snippet Designer is an open source extension for Visual Studio that makes it easier to create and manage code snippets.

### Snippet Designer

The Snippet Designer extension adds both an editor for .snippet files as well as a tool window to search for existing code snippets.



### To Use

You can interact with the Snippet Designer within Visual Studio in several different ways, including the following:

- You can create a new code snippet from scratch by launching the new file

dialog box (File | New File) and then choosing Snippet Designer | Code Snippet. This launches the Snippet Editor, which lets you enter the code snippet, create variable replacements, and set metadata about the snippet.

- You can create a snippet from existing code in your editor (for C#, Visual Basic, JavaScript, SQL, XML, or HTML) by highlighting the text and choosing Export As Snippet from the context menu. This launches the Snippet Editor prepopulated with the highlighted code.
- You can search for existing code snippets by launching the Snippet Explorer tool window at View | Other Windows | Snippet Explorer. This tool window enables you to perform text search on all the code snippets on your computer. From this window, you can easily preview, edit, and delete any code snippet.

## More Information

For more information about the Snippet Designer extension, including documentation, source code, issue tracker, and forum discussions on its CodePlex page, visit <http://snippetdesigner.codeplex.com>.

## 08.22 Document Your Code with Three Keystrokes

<b>NAME</b>	GhostDoc
<b>CREATED BY</b>	SubMain
<b>LOCATION</b>	<a href="http://submain.com/download/ghostdoc">http://submain.com/download/ghostdoc</a>

GhostDoc generates XML comments for your code by using a simple rules engine.

### GhostDoc

Pairing this tool with a documentation generator such as NDoc (<http://ndoc.sourceforge.net>) or SandCastle (<http://www.sandcastledocs.com>) can automate much of the work of documenting your API.

### To Use

GhostDoc adds a Document This command both to the Tools | GhostDoc menu and to the code editor context menu. By default, this command is available using the keyboard shortcut Ctrl+Shift+D.

When you invoke the command on a code element, the element's name, parameters, and context information are passed through the GhostDoc rules engine. The rules engine performs a few linguistic tricks to produce a documentation comment that makes sense in the context of your code.

GhostDoc can produce documentation comments for classes, interfaces, structs, enums, constructors, finalizers, methods, properties, fields, events, indexers, and delegates.

The generated documentation comments are not complete, but they do provide a great starting point. The documentation comments in the code below were generated by GhostDoc and have not been edited from the original output:

```
/// <summary>
///
/// </summary>
public class Request : IDisposable
{
    /// <summary>
    /// Initializes a new instance of the <see cref="Request"/> class.
    /// </summary>
```

```

public Request() {}

/// <summary>
/// Performs application-defined tasks associated with
/// freeing, releasing, or resetting unmanaged resources.
/// </summary>
public void Dispose() {}

/// Gets the status for user.
/// </summary>
/// <param name="userId">The user id.</param>
/// <returns>
///   <see cref="API.RequestStatus"/> The request status
/// </returns>
public RequestStatus GetStatusForUser( string userId ) /* ... */

/// <summary>
/// Gets a value indicating whether this instance is passive.
/// </summary>
/// <value>
///   <c>true</c> if this instance is passive; otherwise,
<c>false</c>.
/// </value>
public bool IsPassive { get; private set; }

/// <summary>
/// Gets or sets the timeout.
/// </summary>
/// <value>
/// The timeout.
/// </value>
public TimeSpan Timeout { get; set; }
}

```

Some conventions and niceties that drive GhostDoc:

- Code symbols, such as parameter and member names, are split into words by case changes. For example, “userId” becomes “user id.”
- Certain common members, such as the Dispose method and equality members, are recognized and documented with specific verbiage.
- Method names are assumed to be describing an activity. For example, GetStatusForUser is interpreted as “Gets the status for user.” When the method name contains only a verb, such as Check, GhostDoc assumes the action applies to the class and documents the method as “Checks this instance.”
- Property names that start with certain words, such as “Is,” “Has”, “Can”, and

so on, are documented as state verifications. Property names containing a single word are documented as states. The documentation comment respects modifiers on property accessors, so read-only properties are documented as such.

- GhostDoc is smart enough to add references to types used in member signatures.
- Documentation comments applied to base types and members are inherited in deriving types.

## To Customize

You can customize the way GhostDoc treats individual code element from the Tools | GhostDoc | Options page. The options for code-matching rules are extensive enough to adapt to any coding culture that relies on consistent naming conventions.

From the GhostDoc options dialog box, you can also configure linguistic operations and export or import your GhostDoc settings.

## More Information

For more information about the GhostDoc extension, see  
<http://submain.com/products/ghostdoc.aspx>.

## 08.23 Customize Visual Studio Using Windows PowerShell

NAME	StudioShell
CREATED BY	Code Owls LLC
LOCATION	<a href="http://studioshell.codeplex.com">http://studioshell.codeplex.com</a>

StudioShell is a deeply integrated Windows PowerShell module that simplifies access to many of the extensibility features of Visual Studio.

### StudioShell

Using StudioShell, with only a few lines of PowerShell script you can perform activities that typically require an add-in.

### To Use

StudioShell ships with a hosted PowerShell console available from the main menu at View | StudioShell. By default, this console is available using the keyboard shortcut Ctrl+Shift+Enter.

You can use StudioShell features in other hosted PowerShell consoles, such as NuGet (<http://nuget.codeplex.com>) or PowerGUI VSX (<http://powerguivsx.codeplex.com>). From these consoles, enter the following command in your console to activate StudioShell:

```
Import-Module StudioShell
```

StudioShell exposes many of the extensibility features of Visual Studio as a drive named “DTE.” This drive lets you explore and access the extensibility features as if they were a file system on your local computer, using the same PowerShell commands you would use to manipulate files, the registry, and so on.

For example, you can see a list of existing Visual Studio window configurations by typing:

```
get-childitem dte:/windowConfigurations
```

You can create a new named window configuration by arranging your user interface and then entering:

```
new-item dte:/windowConfigurations -name MyConfig
```

You can apply the new window configuration at any time using:

```
invoke-item dte:/windowConfigurations/MyConfig
```

The standard PowerShell item commands apply to every area of the DTE drive. For example, you can use the same new-item command to create new project items:

```
new-item dte:/solution/projects/myProject -name MyClass.cs -type class
```

The Visual Studio features exposed on the DTE drive include the following:

- **User interface elements**—Manipulate windows, modify menus, add custom commands implemented in PowerShell, manage items to the task and error lists, and drop data into the output pane.
- **Visual Studio settings**—Script custom font and color settings for different environments.
- **Solutions and Projects**—Create new projects and project items, explore and manipulate your code model, or add and remove project references.
- **Breakpoints and the Debugger**—Create and remove breakpoints, set tracepoint conditions, write minidumps, or explore the current stack trace from the console.

StudioShell adds new features to Visual Studio as well, including the following:

- **Simple grid, chart, and graphing output windows usable from the console**—For example, use the console to get a list of code metrics and drop them in a graph.
- **Profile scripts for environment customization**—For example, import any other PowerShell modules you use regularly.
- **Solution script modules for per-solution environment customization**—Modify your menus and windows on a solution-by-solution basis.

## To Get Help

StudioShell documentation is available within the console. To get started, type:

```
get-help about_studioshell
```

For information about using standard PowerShell commands on the DTE drive, navigate to the area of the drive where you want to work and use the **get-help** command to retrieve help for the command you want to use:

```
cd dte:/debugger.breakpoints
```

```
get-help new-item
```

## To Customize

StudioShell settings are available from the Tools | Options dialog in the StudioShell pane. You can indicate a specific console to use when StudioShell is invoked, including the default StudioShell hosted console, the process console, or no console (for example, if you want to use StudioShell from NuGet). In addition, you can change the profile script behavior of StudioShell and enable startup activity logging.

You can customize your console session with any PowerShell module or script. To find scripts relevant to your needs, visit the PowerShell Code Repository at <http://www.poshcode.org>.

## More Information

For more information about the StudioShell module, see <http://studioshell.codeplex.com> and <http://www.beefycode.com/post/Announcing-StudioShell.aspx>.

# **Appendix A. Visual Studio Keyboard Shortcut Posters**

Microsoft produces keyboard shortcut posters tailored for both general use and for each major language. Knowing these shortcuts can dramatically help you in your day-to-day work. Get these now. Seriously...*now!* This appendix lists the locations where you can find the posters:

## Visual Studio 2010 Shortcuts

### Visual Studio 2010 Shortcuts

- **All Languages 2010** <http://go.microsoft.com/fwlink/?LinkId=220091>

### Visual Studio 2008 Shortcuts

- **C# 2008** <http://go.microsoft.com/fwlink/?LinkId=220094>
- **VB 2008** <http://go.microsoft.com/fwlink/?LinkId=220111>
- **C++ 2008** <http://go.microsoft.com/fwlink/?LinkId=220112>

### Visual Studio 2005 Shortcuts

- **C# 2005** <http://go.microsoft.com/fwlink/?LinkId=220113>
- **VB 2005** <http://go.microsoft.com/fwlink/?LinkId=220114>
- **C++ 2005** <http://go.microsoft.com/fwlink/?LinkId=220115>

#### NOTE

Wow! You've read all the tips in the book, and that's super cool! As a thank you, we would like to offer you even more tips online in the form of traditional New Orleans-style lagniappe—a little something extra on the side. Make sure to download the appendix full of lagniappe tips, free of charge, at <http://go.microsoft.com/fwlink/?LinkId=223758>.

# Index

## A NOTE ON THE DIGITAL INDEX

A link in an index entry is displayed as the section title in which that entry appears. Because some sections have multiple index markers, it is not unusual for an entry to have several links to the same section. Clicking on any link will take you directly to the place in the text in which the marker appears.

## Symbols

#If DEBUG compiler option, [Compiler Directive](#)

%comspec% variable, [03.31 Using External Tools](#)

<deployment retail=true> setting, [07.30 Application and Page Level Tracing](#)

“A New Standard For Packaging Your Data”, [Inside a .vsix File](#)

“Create a Shortcut Key for a Macro”, [03.34 Creating and Using a Macro](#)

“Disable Mouse Wheel Zoom” extension, [06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel](#)

“How to Customize What Files to Search with Find In Files”, [Look in](#)

“Optimizing Visual Studio 2010 and WPF Applications for Remote Desktop”,  
[01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

“Side Effects and Expressions”, [Circling arrows](#)

“The Daily WTF” extension (VS), [08.08 Submit to “The Daily WTF”](#)–[08.09 Diff Files Using the Editor](#), [Share Bad Code with the World](#), [08.09 Diff Files Using the Editor](#)

“Understanding Commands: Aliases”, [Make an Alias](#)

“Understanding Commands: Running Commands”, [03.22 Testing a Command](#)

“Using External Tools”, [Close On Exit](#)

# A

Actions area (Exception Assistant), [Actions–Turning Off the Exception Assistant](#), [Turning Off the Exception Assistant](#)

Actions folder (VS Image Library), [Actions](#)

Active Files area (IDE navigator), [Navigator Areas](#)

active items, tracking in Solution Explorer, [02.07 Track Active Item in Solution Explorer](#)–[02.08 Type-Ahead Selection Support in Solution Explorer](#), [02.08 Type-Ahead Selection Support in Solution Explorer](#), [02.08 Type-Ahead Selection Support in Solution Explorer](#)

Active Tool Windows area (IDE navigator), [Navigator Areas](#)

Add Command dialog box, [Modify Selection–Modify Selection](#), [Modify Selection](#), [Modify Selection](#)

Add New Item dialog box, [02.17 Reorganize the Default Item Templates](#)

Add Or Remove Buttons (toolbars), [01.11 Customize Your Toolbars in Visual Studio 2010: Toolbars Tab](#)

aliases, [03.20 Understanding Commands: Aliases](#)–[03.21 Understanding Commands: Arguments and Switches](#), [03.20 Understanding Commands: Aliases](#), [03.21 Understanding Commands: Arguments and Switches](#), [03.21 Understanding Commands: Arguments and Switches](#), [03.33 Exporting Your Command Window Aliases and External Tools List](#)–[03.34 Creating and Using a Macro](#), [03.33 Exporting Your Command Window Aliases and External Tools List](#), [03.34 Creating and Using a Macro](#)

command aliases, [03.20 Understanding Commands: Aliases](#)–[03.21 Understanding Commands: Arguments and Switches](#), [03.20 Understanding Commands: Aliases](#), [03.21 Understanding Commands: Arguments and Switches](#), [03.21 Understanding Commands: Arguments and Switches](#)

Command Window Aliases, [03.33 Exporting Your Command Window](#)

[Aliases and External Tools List](#)–[03.34 Creating and Using a Macro](#), [03.33 Exporting Your Command Window Aliases and External Tools List](#), [03.34 Creating and Using a Macro](#)

All Components option (Look In dialog), [All Components](#)

All tab/Common tab in statement completion, [06.17 Move Between the Common Tab and All Tab in Statement Completion \(VB\)](#)

AllMargins extension (VS), [08.16 Get a Bird's-Eye View of Your Code in an Editor Margin](#)–[To Install](#), [To Uninstall](#), [To Uninstall](#), [To Install](#)

ampersand (&) for accelerator keys, [03.32 Create Keyboard Accelerators for External Tools](#)

animations, [01.07 Change Tool Window Animations](#)–[01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#), [Animations](#)

Animations folder (VS Image Library), [Animations](#)

tool window, changing, [01.07 Change Tool Window Animations](#)–[01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#)

Annotations\_Buttons folder (VS Image Library),  
[Annotations\\_Buttons](#)–[Objects](#), [Objects](#), [Objects](#)

application-level tracing, [07.30 Application and Page Level Tracing](#)–[Combined Tracing](#), [Attributes](#), [Attributes](#), [Page Level Tracing](#), [Combined Tracing](#)

arguments and switches (commands), [03.21 Understanding Commands: Arguments and Switches](#)–[03.22 Testing a Command](#), [Arguments and Switches](#), [Using the Arguments and Switches](#), [03.22 Testing a Command](#)

asterisk (\*), [03.05 Expand and Collapse All in the Toolbox](#), [05.05 Undo Quick Replace and Replace in Files](#)

for undoing Quick Replace, [05.05 Undo Quick Replace and Replace in Files](#)

to expand Toolbox, [03.05 Expand and Collapse All in the Toolbox](#) attributes, trace, [Attributes–Page Level Tracing](#), [Attributes, Page Level Tracing](#)

Auto Hide Channel, [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#)–[03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#)

auto-hide all tool windows, [03.12 Auto-Hide All Tool Windows](#)–[03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.12 Auto-Hide All Tool Windows](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#)

auto-populating, stopping Toolbox from, [03.30 Stop the Toolbox from Auto-Populating from the Solution](#)

Automatically Adjust Visual Experience Based On Client Performance check box, [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

AutoRecover feature, [01.05 AutoRecover](#), [01.05 AutoRecover](#), [02.12 Creating Temporary Projects](#)

Autos window, [07.35 The Watch Window: Adding Watches from Variable Windows](#), [07.48 Understanding the Autos Window](#), [VB Shows Three Statements on Either Side](#)

## B

Begin A Group option (Modify Selection), Modify Selection

blank lines, 06.03 How to Keep from Accidentally Copying a Blank Line,  
06.09 Insert a Blank Line Above or Below the Current Line

accidentally copying, 06.03 How to Keep from Accidentally Copying a Blank Line

inserting above/below current line, 06.09 Insert a Blank Line Above or Below the Current Line

Boolean expressions, 07.22 Set a Simple Breakpoint Condition

Boolean logic, 07.23 Set a Complex Breakpoint Condition

box selections, 06.28 Replacing Text with a Box Selection–06.29 Pasting the Contents of One Box Selection into Another, 06.28 Replacing Text with a Box Selection, 06.29 Pasting the Contents of One Box Selection into Another–06.30 Pasting a Single Selection into a Box Selection, 06.29 Pasting the Contents of One Box Selection into Another, 06.30 Pasting a Single Selection into a Box Selection, 06.30 Pasting a Single Selection into a Box Selection–06.31 Using Zero-Length Box Selection, 06.30 Pasting a Single Selection into a Box Selection, 06.31 Using Zero-Length Box Selection, 06.31 Using Zero-Length Box Selection–06.32 View White Space, 06.32 View White Space

pasting content between, 06.29 Pasting the Contents of One Box Selection into Another–06.30 Pasting a Single Selection into a Box Selection, 06.30 Pasting a Single Selection into a Box Selection, 06.30 Pasting a Single Selection into a Box Selection, 06.30 Pasting a Single Selection into a Box Selection–06.31 Using Zero-Length Box Selection

pasting single selection into, 06.30 Pasting a Single Selection into a Box Selection–06.31 Using Zero-Length Box Selection, 06.31 Using Zero-Length Box Selection

replacing text with, [06.28 Replacing Text with a Box Selection](#)–[06.29 Pasting the Contents of One Box Selection into Another](#), [06.28 Replacing Text with a Box Selection](#), [06.29 Pasting the Contents of One Box Selection into Another](#)

zero-length box selection, [06.31 Using Zero-Length Box Selection](#)–[06.32 View White Space](#), [06.32 View White Space](#)

braces, moving/selecting between matching, [06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)](#)–[06.16 Invoke Statement Completion](#), [Moving](#), [06.16 Invoke Statement Completion](#)

[Break All Processes When One Process Breaks](#), [07.46 Understanding Break All Processes When One Process Breaks](#)–[07.47 Changing Context in the Locals Window](#), [07.46 Understanding Break All Processes When One Process Breaks](#), [07.47 Changing Context in the Locals Window](#), [07.47 Changing Context in the Locals Window](#)

breakpoints, [03.03 Cycle Through Your Open Tool Windows](#), [Debugging](#)–[VB, Compiler Directive](#), [VB](#), [07.02 Using Ctrl+Alt+B to Open the Breakpoints Window](#), [07.03 Adding Labels to Breakpoints](#)–[07.03 Adding Labels to Breakpoints](#), [07.03 Adding Labels to Breakpoints](#), [07.04 Enable or Disable All Breakpoints](#)–[C++, C++, C++](#), [07.19 Searching Breakpoints](#)–[07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#), [07.20 Breakpoint Hit Count](#)–[07.21 Set a Breakpoint on a Function](#), [Break When The Hit Count Is Equal To](#), [Break When The Hit Count Is Greater Than Or Equal To](#), [07.21 Set a Breakpoint on a Function](#), [07.22 Set a Simple Breakpoint Condition](#)–[Special Notes](#), [Is True](#), [Special Notes](#), [07.23 Set a Complex Breakpoint Condition](#)–[07.23 Set a Complex Breakpoint Condition](#), [07.23 Set a Complex Breakpoint Condition](#), [07.24 Setting a Breakpoint Filter](#)–[07.25 Setting a Tracepoint in Source Code](#), [07.25 Setting a Tracepoint in Source Code](#), [07.43 Setting a Breakpoint in the Call Stack Window](#)–[07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in](#)

## the Call Stack Window

adding labels to, [07.03 Adding Labels to Breakpoints](#)–[07.03 Adding Labels to Breakpoints](#), [07.03 Adding Labels to Breakpoints](#), [07.03 Adding Labels to Breakpoints](#)

enabling/disabling all, [07.04 Enable or Disable All Breakpoints](#)–[C++](#), [C++](#), [C++](#)

hidden Breakpoints tool window, [03.03 Cycle Through Your Open Tool Windows](#)

Hit Count, [07.20 Breakpoint Hit Count](#)–[07.21 Set a Breakpoint on a Function](#), [Break When The Hit Count Is Equal To](#), [Break When The Hit Count Is Greater Than Or Equal To](#), [07.21 Set a Breakpoint on a Function](#)

searching, [07.19 Searching Breakpoints](#)–[07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#)

setting complex conditions for, [07.23 Set a Complex Breakpoint Condition](#)–[07.23 Set a Complex Breakpoint Condition](#), [07.23 Set a Complex Breakpoint Condition](#)

setting filters for, [07.24 Setting a Breakpoint Filter](#)–[07.25 Setting a Tracepoint in Source Code](#), [07.25 Setting a Tracepoint in Source Code](#)

setting in Call Stack window, [07.43 Setting a Breakpoint in the Call Stack Window](#)–[07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#)

setting simple conditions of, [07.22 Set a Simple Breakpoint Condition](#)–[Special Notes](#), [Is True](#), [Special Notes](#)

setting with code, [Debugging](#)–[VB](#), [Compiler Directive](#), [VB](#) window, opening with Ctrl+Alt+B, [07.02 Using Ctrl+Alt+B to Open the Breakpoints Window](#)

buttons, [Controls–Modify Selection](#), [Modify Selection](#), [Modify Selection](#), [Buttons](#), [06.46 Insert Quotes When Typing Attribute Values](#)

Buttons area (customize dialog box), [Controls–Modify Selection](#), [Modify Selection](#), [Modify Selection](#)

Code Snippets Manager, [06.46 Insert Quotes When Typing Attribute Values](#)

Quick Find, [Buttons](#)

# C

C# language, [VB](#)

setting breakpoints in, [VB](#)

Call Hierarchy window, [07.18 Using the Call Hierarchy](#)–[07.19 Searching Breakpoints](#), [07.18 Using the Call Hierarchy](#), [07.18 Using the Call Hierarchy](#), [07.19 Searching Breakpoints](#)

Call Stack window, [07.43 Setting a Breakpoint in the Call Stack Window](#)–[07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#), [07.44 Setting a Tracepoint in the Call Stack Window](#)–[07.45 Using the WPF Tree Visualizer](#), [07.44 Setting a Tracepoint in the Call Stack Window](#), [07.45 Using the WPF Tree Visualizer](#)

setting breakpoints in, [07.43 Setting a Breakpoint in the Call Stack Window](#)–[07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#)

setting tracepoints in, [07.44 Setting a Tracepoint in the Call Stack Window](#)–[07.45 Using the WPF Tree Visualizer](#), [07.44 Setting a Tracepoint in the Call Stack Window](#), [07.45 Using the WPF Tree Visualizer](#)

call stacks, unwinding on unhandled exceptions, [Unwind The Call Stack On Unhandled Exceptions](#)

Char variables (customizing search results), [Char](#) characters, transposing, [06.10 Transpose Lines, Words, and Characters](#)–[06.11 How to Cycle Through the Clipboard Ring](#), [06.11 How to Cycle Through the Clipboard Ring](#)

Choose Search Folders option (Find What combo box), [Look in](#)

Choose Settings To Export dialog box, [01.03 Exporting Your Environment](#)

## Settings

circling arrows icon, [Circling arrows](#)

[Class Diagrams](#), [03.24 Find Keyboard Shortcuts](#)

[Class View](#), [Search Results](#), [07.36 Create Folders in Class View–Creating Subfolders](#), [Create a New Folder](#), [Creating Subfolders](#), [Creating Subfolders](#)

creating folders in, [07.36 Create Folders in Class View–Creating Subfolders](#), [Create a New Folder](#), [Creating Subfolders](#), [Creating Subfolders](#)

“[Class View and Object Browser Icons](#)”, [Search Results](#)

Classic view (Visual Studio 2010 Online Help), [Using Classic View–01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings](#)

Clear All command (Find Symbol), [Clear All](#)

Clear All option (Find Results window), [Clear All](#)

click and drag for docking/undocking tool windows, [Click and drag–Control box \(Visual Studio 2010 only\)](#), [Control box \(Visual Studio 2010 only\)](#), [Result](#) client projects, launching, [02.05 Multiple Startup Projects](#)

Clipboard Ring, cycling through, [06.11 How to Cycle Through the Clipboard Ring–06.12 Using the Undo and Redo Stack](#), [06.12 Using the Undo and Redo Stack](#)

Close On Exit option (Initial Directory), [Close On Exit](#)

closing current document window, [04.05 Close the Current Document Window](#)

closing tool windows, [03.04 Closing Tool Windows](#)

code, [Rearrange](#), [06.21 Drag and Drop Code into the Toolbox–06.21 Drag and Drop Code into the Toolbox](#), [06.21 Drag and Drop Code into the Toolbox](#),

[06.25 Toggle Designer](#), [06.33 Collapsing Your Code with Outlining](#)–[Click Anywhere in Area \(Menu Item\)](#), [Click Anywhere in Area \(Menu Item\)](#), [Click Anywhere in Area \(Menu Item\)](#), [06.34 Using Hide Selection](#)–[06.35 Collapse to Definitions with Outlining](#), [06.35 Collapse to Definitions with Outlining](#), [07.01 Setting a Breakpoint with Code](#)–[VB](#), [VB](#), [VB](#), [07.07 Create Code Shortcuts in the Task List](#)–[07.08 Code Definition Window](#), [07.07 Create Code Shortcuts in the Task List](#), [07.08 Code Definition Window](#), [07.08 Code Definition Window](#), [08.02 Insert Images into Your Code](#)–[To Save](#), [To Save](#), [To Save](#), [08.03 Add Visual Guidelines to Your Code](#)–[08.05 Sync the Solution Explorer to the Current File](#), [To Use](#), [08.05 Sync the Solution Explorer to the Current File](#), [08.09 Diff Files Using the Editor](#)–[To Use](#), [To Install](#), [To Uninstall](#), [To Use](#), [08.15 View Code Blocks Using Vertical Lines](#)–[To Use](#), [To Customize](#), [AllMargins](#), [To Use](#), [Regex Editor](#)–[To Use](#), [To Use](#), [GhostDoc](#)–[08.23 Customize Visual Studio Using Windows PowerShell](#), [08.23 Customize Visual Studio Using Windows PowerShell](#)

adding visual guidelines to, [08.03 Add Visual Guidelines to Your Code](#)–[08.05 Sync the Solution Explorer to the Current File](#), [To Use](#), [08.05 Sync the Solution Explorer to the Current File](#)

Code Definition window, [07.08 Code Definition Window](#)

Code view, [06.25 Toggle Designer](#)

Code Window option (Editor Context Menu), [Rearrange](#)

Code Compare extension (VS), [08.09 Diff Files Using the Editor](#)–[To Use](#), [To Install](#), [To Uninstall](#), [To Use](#)

collapsing with outlining, [06.33 Collapsing Your Code with Outlining](#)–[Click Anywhere in Area \(Menu Item\)](#), [Click Anywhere in Area \(Menu Item\)](#), [Click Anywhere in Area \(Menu Item\)](#)

creating Regular Expressions within, [Regex Editor](#)–[To Use](#), [To Use](#)

documenting with 3 keystrokes, [GhostDoc](#)–[08.23 Customize Visual Studio Using Windows PowerShell](#), [08.23 Customize Visual Studio Using](#)

## **Windows PowerShell**

drag and drop into Toolbox, [06.21 Drag and Drop Code into the Toolbox](#)–[06.21 Drag and Drop Code into the Toolbox, 06.21 Drag and Drop Code into the Toolbox](#)

Hide Selection, [06.34 Using Hide Selection](#)–[06.35 Collapse to Definitions with Outlining](#), [06.35 Collapse to Definitions with Outlining](#)

inserting images into, [08.02 Insert Images into Your Code](#)–[To Save, To Save](#)

setting breakpoints with, [07.01 Setting a Breakpoint with Code](#)–[VB, VB, VB](#)

Task Lists, creating code shortcuts in, [07.07 Create Code Shortcuts in the Task List](#)–[07.08 Code Definition Window](#), [07.07 Create Code Shortcuts in the Task List, 07.08 Code Definition Window](#)

viewing code blocks with vertical lines, [08.15 View Code Blocks Using Vertical Lines](#)–[To Use, To Customize, AllMargins, To Use](#)

code snippets, [06.40 Inserting Code Snippets](#)–[06.41 Surround with a Code Snippet](#), [06.41 Surround with a Code Snippet](#)–[06.41 Surround with a Code Snippet](#), [06.41 Surround with a Code Snippet, 06.41 Surround with a Code Snippet](#)–[06.42 Using Code Snippets](#)–[06.43 HTML Code Snippets, 06.43 HTML Code Snippets](#)–[06.44 JavaScript Code Snippets, 06.43 HTML Code Snippets](#), [06.44 JavaScript Code Snippets](#)–[06.45 Using the Code Snippets Manager](#), [06.44 JavaScript Code Snippets, 06.45 Using the Code Snippets Manager](#)–[06.45 Using the Code Snippets Manager](#)–[06.45 Using the Code Snippets Manager](#), [06.45 Using the Code Snippets Manager](#)–[06.45 Using the Code Snippets Manager](#)–[06.45 Using the Code Snippets Manager](#), [06.45 Using the Code Snippets Manager](#)–[06.45 Using the Code Snippets Manager](#), [06.45 Using the Code Snippets Manager](#)–[06.54 Create New Code Snippets from Existing Ones](#)–[06.54 Create New Code Snippets from Existing Ones](#), [06.54 Create New Code Snippets from Existing Ones, 06.54 Create New Code Snippets from Existing Ones](#)–[08.21](#)

## Create and Find Code Snippets–To Use, To Use, To Use

Code Snippets Manager, 06.45 Using the Code Snippets Manager–06.45 Using the Code Snippets Manager, 06.45 Using the Code Snippets Manager–06.45 Using the Code Snippets Manager, 06.45 Using the Code Snippets Manager, 06.45 Using the Code Snippets Manager, 06.45 Using the Code Snippets Manager

HTML, 06.43 HTML Code Snippets–06.44 JavaScript Code Snippets, 06.44 JavaScript Code Snippets

inserting, 06.40 Inserting Code Snippets–06.41 Surround with a Code Snippet, 06.41 Surround with a Code Snippet

JavaScript, 06.44 JavaScript Code Snippets–06.45 Using the Code Snippets Manager, 06.45 Using the Code Snippets Manager

new, creating from existing, 06.54 Create New Code Snippets from Existing Ones–06.54 Create New Code Snippets from Existing Ones, 06.54 Create New Code Snippets from Existing Ones, 06.54 Create New Code Snippets from Existing Ones

Snippet Designer, 08.21 Create and Find Code Snippets–To Use, To Use, To Use

surrounding existing code with, 06.41 Surround with a Code Snippet–06.41 Surround with a Code Snippet, 06.41 Surround with a Code Snippet

using, 06.42 Using Code Snippets–06.43 HTML Code Snippets, 06.43 HTML Code Snippets

code, writing, 06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel–06.03 How to Keep from Accidentally Copying a Blank Line, 06.02 Zoom In or Out of Text in the Editor, 06.03 How to Keep from Accidentally Copying a Blank Line–06.04 Make IntelliSense Transparent, 06.03 How to Keep from Accidentally Copying a Blank Line, 06.03 How to Keep from Accidentally Copying a Blank Line, 06.04 Make IntelliSense Transparent,

Cut, 06.17 Move Between the Common Tab and All Tab in Statement Completion (VB), 06.17 Move Between the Common Tab and All Tab in Statement Completion (VB), 06.18 Using Parameter Information–06.20 Word Completion, 06.19 Using Quick Info, 06.19 Using Quick Info–06.21 Drag and Drop Code into the Toolbox, 06.20 Word Completion, 06.21 Drag and Drop Code into the Toolbox, Remove Unused Usings, Sort Usings, 06.24 Switch Between Design and Source in Web Projects, 06.26 Change the Default View in the HTML Editor–06.28 Replacing Text with a Box Selection, 06.27 Jump Back to the Editor from Just About Anywhere, 06.27 Jump Back to the Editor from Just About Anywhere, 06.28 Replacing Text with a Box Selection, 06.28 Replacing Text with a Box Selection, 06.29 Pasting the Contents of One Box Selection into Another–06.30 Pasting a Single Selection into a Box Selection, 06.30 Pasting a Single Selection into a Box Selection, 06.31 Using Zero-Length Box Selection, 06.31 Using Zero-Length Box Selection–06.32 View White Space, 06.32 View White Space, 06.32 View White Space, 06.33 Collapsing Your Code with Outlining–Click Anywhere in Area (Menu Item), Minus Sign, Click Anywhere in Area (Menu Item), 06.34 Using Hide Selection–06.35 Collapse to Definitions with Outlining, 06.35 Collapse to Definitions with Outlining–06.36 Cut, Copy, and Paste Collapsed Code with Outlining, 06.36 Cut, Copy, and Paste Collapsed Code with Outlining, 06.36 Cut, Copy, and Paste Collapsed Code with Outlining, 06.37 Understanding Word Wrap–06.38 Properties Window Keyboard Shortcuts, 06.37 Understanding Word Wrap, 06.38 Properties Window Keyboard Shortcuts–Working with Categories, 06.38 Properties Window Keyboard Shortcuts, Working with the Tool Window, Working with Categories, 06.39 Document Outline: Web Projects–06.40 Inserting Code Snippets, 06.40 Inserting Code Snippets, 06.47 Format the Current Document or Selection (Web)–06.48 Using the Navigation Bar, 06.48 Using the Navigation Bar, 06.49 HTML Editor Tag Navigation–06.51 Display HTML/CSS Warnings as Errors, 06.49 HTML Editor Tag Navigation, 06.50 Format HTML on Paste, 06.51 Display HTML/CSS Warnings as Errors–06.52 Updating JScript IntelliSense, 06.51

[Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#), [06.52 Updating JScript IntelliSense](#), [06.55 Understanding the Navigation Stack](#)–[06.55 Understanding the Navigation Stack](#), [06.55 Understanding the Navigation Stack](#), [06.57 Select from the Current Cursor Location to the Last Go Back Marker](#), [06.57 Select from the Current Cursor Location to the Last Go Back Marker](#), [Embedded Styles](#), [06.61 Understanding Tag Specific Options](#)

blank lines, avoiding copying, [06.03 How to Keep from Accidentally Copying a Blank Line](#)–[06.04 Make IntelliSense Transparent](#), [06.04 Make IntelliSense Transparent](#)

Collapse to Definitions with outlining, [06.35 Collapse to Definitions with Outlining](#)–[06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#), [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#), [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#)

collapsing code with outlining, [06.33 Collapsing Your Code with Outlining](#)–[Click Anywhere in Area \(Menu Item\)](#), [Minus Sign](#), [Click Anywhere in Area \(Menu Item\)](#)

CSS versions, choosing, [Embedded Styles](#), [06.61 Understanding Tag Specific Options](#)

current line, cutting/deleting, [Cut](#)

default view, changing in HTML editor, [06.26 Change the Default View in the HTML Editor](#)–[06.28 Replacing Text with a Box Selection](#), [06.27 Jump Back to the Editor from Just About Anywhere](#), [06.28 Replacing Text with a Box Selection](#)

Design and Source views, switching between, [06.24 Switch Between Design and Source in Web Projects](#)

Document Outline (web projects), [06.39 Document Outline: Web Projects](#)–[06.40 Inserting Code Snippets](#), [06.40 Inserting Code Snippets](#)

documents/selections, formatting for HTML, [06.47 Format the Current Document or Selection \(Web\)](#)–[06.48 Using the Navigation Bar](#), [06.48 Using the Navigation Bar](#)

Esc to return to Editor, [06.27 Jump Back to the Editor from Just About Anywhere](#)

formatting HTML on paste, [06.50 Format HTML on Paste](#)

Hide Selection, using, [06.34 Using Hide Selection](#)–[06.35 Collapse to Definitions with Outlining](#), [06.35 Collapse to Definitions with Outlining](#)

HTML Editor tag navigation, [06.49 HTML Editor Tag Navigation](#)–[06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#)

HTML/CSS warnings, displaying as errors, [06.51 Display HTML/CSS Warnings as Errors](#)–[06.52 Updating JScript IntelliSense](#), [06.51 Display HTML/CSS Warnings as Errors](#), [06.52 Updating JScript IntelliSense](#)

moving between Common tab/All tab in statement completion, [06.17 Move Between the Common Tab and All Tab in Statement Completion \(VB\)](#)

navigating backwards/forwards with Go Back markers, [06.57 Select from the Current Cursor Location to the Last Go Back Marker](#)

Navigation Bar, using, [06.49 HTML Editor Tag Navigation](#)

navigation stack, [06.55 Understanding the Navigation Stack](#)–[06.55 Understanding the Navigation Stack](#), [06.55 Understanding the Navigation Stack](#)

parameter information, using, [06.18 Using Parameter Information](#)–[06.20 Word Completion](#), [06.19 Using Quick Info](#), [06.20 Word Completion](#)

pasting contents between box selections, [06.29 Pasting the Contents of One Box Selection into Another](#)–[06.30 Pasting a Single Selection into a](#)

[\*\*Box Selection\*\*](#), [\*\*06.30 Pasting a Single Selection into a Box Selection\*\*](#)

pasting single selections into box selection, [\*\*06.31 Using Zero-Length Box Selection\*\*](#)

Properties window keyboard shortcuts, [\*\*06.38 Properties Window\*\*](#)

[\*\*Keyboard Shortcuts–Working with Categories\*\*](#), [\*\*Working with the Tool\*\*](#)

[\*\*Window\*\*](#), [\*\*Working with Categories\*\*](#)

Quick Info, using, [\*\*06.19 Using Quick Info–06.21 Drag and Drop Code into the Toolbox\*\*](#), [\*\*06.21 Drag and Drop Code into the Toolbox\*\*](#)

replacing text with box selection, [\*\*06.28 Replacing Text with a Box Selection\*\*](#)

selecting from current cursor to last Go Back marker, [\*\*06.57 Select from the Current Cursor Location to the Last Go Back Marker\*\*](#)

statement completion, invoking, [\*\*06.17 Move Between the Common Tab and All Tab in Statement Completion \(VB\)\*\*](#)

text, zooming in/out in Editor, [\*\*06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel\*\*](#)–[\*\*06.03 How to Keep from Accidentally Copying a Blank Line\*\*](#), [\*\*06.02 Zoom In or Out of Text in the Editor\*\*](#), [\*\*06.03 How to Keep from Accidentally Copying a Blank Line\*\*](#), [\*\*06.03 How to Keep from Accidentally Copying a Blank Line\*\*](#)

using statements, organizing, [\*\*Remove Unused Usings\*\*](#), [\*\*Sort Usings\*\*](#)

white space, viewing, [\*\*06.32 View White Space\*\*](#)

word wrap, [\*\*06.37 Understanding Word Wrap–06.38 Properties Window Keyboard Shortcuts\*\*](#), [\*\*06.37 Understanding Word Wrap\*\*](#), [\*\*06.38 Properties Window Keyboard Shortcuts\*\*](#)

zero-length box selection, [\*\*06.31 Using Zero-Length Box Selection–06.32 View White Space\*\*](#), [\*\*06.32 View White Space\*\*](#)

codes, vstip, [\*\*01.01 Running Multiple Versions of Visual Studio Side-By-Side\*\*](#),

[01.02 Getting Table of Contents in Visual Studio 2010 Online Help](#), [01.03 Exporting Your Environment Settings](#)–[01.03 Exporting Your Environment Settings](#), [01.04 Remove Projects from the Recent Projects List](#), [01.05 AutoRecover](#)–[01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#), [01.05 AutoRecover](#), [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#), [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#), [01.07 Change Tool Window Animations](#), [01.08 Importing or Changing Your Environment Settings](#)–[01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#), [01.09 Change Your Visual Studio Color Scheme](#), [01.11 Customize Your Toolbars in Visual Studio 2010: Toolbars Tab](#)–[Custom Toolbars](#), [Custom Toolbars](#), [01.12 Customize Your Toolbars in Visual Studio 2010: Commands Tab](#)–[01.12 Customize Your Toolbars in Visual Studio 2010: Commands Tab](#), [01.12 Customize Your Toolbars in Visual Studio 2010: Commands Tab](#), [01.13 Visual Studio Logging](#)–[01.14 Visual Studio Safe Mode](#), [01.14 Visual Studio Safe Mode](#), [01.14 Visual Studio Safe Mode](#), [01.15 The ResetSettings Switch](#)–[Same Machine](#), [Two Different Machines](#), [Same Machine](#), [Projects and Items](#)–[02.03 Using Older Frameworks with Multi-Targeting](#), [02.02 Recent Project Templates in the New Project Dialog Box](#), [02.03 Using Older Frameworks with Multi-Targeting](#), [02.03 Using Older Frameworks with Multi-Targeting](#), [02.04 Create Web Application or Virtual Directory in IIS](#)–[02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#), [02.05 Multiple Startup Projects](#)–[02.06 Change the Default New Project Location](#), [02.06 Change the Default New Project Location](#), [02.09 Using Solution Folders](#)–[02.11 Pin a Project to the Recent Projects List](#), [02.10 Navigating Property Tabs in the Project Properties](#), [02.11 Pin a Project to the Recent Projects List](#), [02.11 Pin a Project to the Recent Projects List](#), [02.13 Create Your Own Item Template](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.15 Organizing Your Custom Item Templates](#)–[02.16 Organizing Your Custom Project Templates](#), [02.16 Organizing Your Custom Project Templates](#)

[Organizing Your Custom Project Templates](#)–[02.17 Reorganize the Default Item Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.17 Reorganize the Default Item Templates](#), [02.17 Reorganize the Default Item Templates](#), [02.18 Reorganize the Default Project Templates](#), [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#), [03.02 Dock a Floating Tool Window Back to Its Previous Location](#), [03.03 Cycle Through Your Open Tool Windows](#), [03.05 Expand and Collapse All in the Toolbox](#), [03.06 Searching in the Toolbox](#), [03.07 Navigate Among Tabs in the Toolbox](#), [03.08 Window Layouts: The Four Modes](#), [03.09 Window Layouts: Design, Debug, and Full Screen](#), [03.10 Working with Tabs in the Toolbox](#), [03.11 Using Additional Browsers for Web Development](#)–[Browser Window Size](#), [Browser Window Size](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#)–[03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.14 Moving Tool Windows Around with Your Keyboard](#)–[03.15 Keyboard Access to a Tool Window's Toolbar](#), [03.15 Keyboard Access to a Tool Window's Toolbar](#), [03.16 Command Prompt History](#), [03.17 Command Prompt Tab Completion](#)–[Click and drag](#), [Finally](#), [Click and drag](#), [03.19 Understanding Commands: Simple Commands](#)–[03.20 Understanding Commands: Aliases](#), [03.19 Understanding Commands: Simple Commands](#), [03.19 Understanding Commands: Simple Commands](#), [03.20 Understanding Commands: Aliases](#), [03.20 Understanding Commands: Aliases](#), [03.24 Find Keyboard Shortcuts](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)–[03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#), [03.28 Understanding Commands: Logging Commands](#), [03.29 Export Your Window Layouts](#), [03.31 Using External Tools](#)–[Prompt For Arguments](#), [03.31 Using External Tools](#), [Prompt For Arguments](#), [03.33 Exporting Your Command Window Aliases and External](#)

[Tools List](#)–[03.34 Creating and Using a Macro](#), [03.34 Creating and Using a Macro](#), [03.35 Visual Studio Image Library](#), [04.01 Insert Documents to the Right of Existing Tabs](#)–[04.02 Recent Files](#), [04.02 Recent Files](#), [04.02 Recent Files](#), [04.03 Working with Documents on Multiple Monitors](#)–[04.03 Working with Documents on Multiple Monitors](#), [04.03 Working with Documents on Multiple Monitors](#), [04.04 Navigate Open Document Windows](#)–[04.06 Open a File Location from the File Tab](#), [04.06 Open a File Location from the File Tab](#), [04.10 Closing Just the Selected Files You Want](#), [04.11 Understanding the File Open Location](#)–[04.12 Show Previous Versions](#), [04.12 Show Previous Versions](#), [04.12 Show Previous Versions](#), [04.13 Using Custom File Extension Associations](#)–[04.13 Using Custom File Extension Associations](#), [04.13 Using Custom File Extension Associations](#), [05.02 Using Quick Find](#)–[05.03 Using a Simple Quick Replace](#), [05.03 Using a Simple Quick Replace](#)–[05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.03 Using a Simple Quick Replace](#), [05.04 Hide the Quick Find and Quick Replace](#), [05.03 Using a Simple Quick Replace](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.06 Using the Find Combo Box Keyboard Shortcuts](#)–[05.07 Using Incremental Search](#), [05.07 Using Incremental Search](#)–[05.08 Search the Currently Selected String Without the Find Window](#), [05.07 Using Incremental Search](#), [05.08 Search the Currently Selected String Without the Find Window](#)–[05.09 Find In Files: Find Options](#), [05.08 Search the Currently Selected String Without the Find Window](#), [05.09 Find In Files: Find Options](#)–[Navigation](#), [05.09 Find In Files: Find Options](#), [05.10 Find In Files: Result Options](#)–[Display File Names Only](#), [Navigation](#), [Display File Names Only](#), [05.11 Replace In Files: Basic Options](#)–[05.12 Go To Definition for Cascading Style Sheets](#), [05.12 Go To Definition for Cascading Style Sheets](#), [05.13 How to Use Navigate To](#)–[05.14 Understanding Find Symbol](#), [05.14 Understanding Find Symbol](#)–[05.15 Find Symbol Results Shortcuts](#), [05.14 Understanding Find Symbol](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#), [05.15 Find Symbol Results Shortcuts](#)–[05.16 Replace in Files: Tagged](#)

[Expressions](#), [05.15 Find Symbol Results Shortcuts](#), [05.16 Replace in Files: Tagged Expressions](#), [05.17 Customize Results in Find In Files Searches](#)–[Char](#), [Char](#), [06.03 How to Keep from Accidentally Copying a Blank Line](#), [06.04 Make IntelliSense Transparent](#), [06.05 Cut or Delete the Current Line](#)–[06.06 Using the New IntelliSense: Keywords](#), [06.06 Using the New IntelliSense: Keywords](#)–[06.07 Using the New IntelliSense: Pascal Case](#), [06.07 Using the New IntelliSense: Pascal Case](#), [06.08 Comment and Uncomment in Web Pages](#)–[06.09 Insert a Blank Line Above or Below the Current Line](#), [06.09 Insert a Blank Line Above or Below the Current Line](#), [06.09 Insert a Blank Line Above or Below the Current Line](#), [06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)](#), [06.18 Using Parameter Information](#), [06.20 Word Completion](#), [06.21 Drag and Drop Code into the Toolbox](#)–[06.21 Drag and Drop Code into the Toolbox](#), [06.21 Drag and Drop Code into the Toolbox](#), [06.22 Using Smart Tags from the Keyboard](#)–[Remove Unused Usings](#), [06.22 Using Smart Tags from the Keyboard](#), [Remove Unused Usings](#), [Remove Unused Usings](#), [06.24 Switch Between Design and Source in Web Projects](#)–[06.26 Change the Default View in the HTML Editor](#), [06.25 Toggle Designer](#), [06.26 Change the Default View in the HTML Editor](#)–[06.28 Replacing Text with a Box Selection](#), [06.26 Change the Default View in the HTML Editor](#), [06.27 Jump Back to the Editor from Just About Anywhere](#), [06.28 Replacing Text with a Box Selection](#), [06.28 Replacing Text with a Box Selection](#), [06.28 Replacing Text with a Box Selection](#)–[06.30 Pasting a Single Selection into a Box Selection](#), [06.30 Pasting a Single Selection into a Box Selection](#), [06.30 Pasting a Single Selection into a Box Selection](#), [06.30 Pasting a Single Selection into a Box Selection](#)–[06.32 View White Space](#), [06.32 View White Space](#)–[06.33 Collapsing Your Code with Outlining](#), [06.32 View White Space](#), [06.33 Collapsing Your Code with Outlining](#)–[Click Anywhere in Area \(Menu Item\)](#), [06.33 Collapsing Your Code with Outlining](#), [Click Anywhere in Area \(Menu Item\)](#), [06.34 Using Hide Selection](#)–[06.34 Using Hide Selection](#), [06.34 Using Hide Selection](#), [06.35](#)

Collapse to Definitions with Outlining, 06.36 Cut, Copy, and Paste Collapsed Code with Outlining, 06.37 Understanding Word Wrap–06.38 Properties Window Keyboard Shortcuts, 06.37 Understanding Word Wrap, 06.38 Properties Window Keyboard Shortcuts–Working with Categories, 06.38 Properties Window Keyboard Shortcuts, Working with Categories, 06.39 Document Outline: Web Projects–06.40 Inserting Code Snippets, 06.40 Inserting Code Snippets, 06.40 Inserting Code Snippets, 06.41 Surround with a Code Snippet–06.41 Surround with a Code Snippet, 06.41 Surround with a Code Snippet, 06.42 Using Code Snippets, 06.43 HTML Code Snippets–06.44 JavaScript Code Snippets, 06.44 JavaScript Code Snippets, 06.45 Using the Code Snippets Manager, 06.46 Insert Quotes When Typing Attribute Values, 06.47 Format the Current Document or Selection (Web), 06.48 Using the Navigation Bar–06.50 Format HTML on Paste, 06.49 HTML Editor Tag Navigation–06.51 Display HTML/CSS Warnings as Errors, 06.50 Format HTML on Paste, 06.50 Format HTML on Paste, 06.50 Format HTML on Paste, 06.51 Display HTML/CSS Warnings as Errors, 06.51 Display HTML/CSS Warnings as Errors, 06.52 Updating JScript IntelliSense, 06.53 Using JScript Libraries in Other JScript Files, 06.54 Create New Code Snippets from Existing Ones–06.54 Create New Code Snippets from Existing Ones, 06.54 Create New Code Snippets from Existing Ones, 06.54 Create New Code Snippets from Existing Ones, 06.55 Understanding the Navigation Stack–06.55 Understanding the Navigation Stack, 06.55 Understanding the Navigation Stack, 06.56 Navigate Backward and Navigate Forward Using Go Back Markers, 06.57 Select from the Current Cursor Location to the Last Go Back Marker–06.57 Select from the Current Cursor Location to the Last Go Back Marker, 06.57 Select from the Current Cursor Location to the Last Go Back Marker, 06.58 Track Changes in the Editor–06.59 Edit Read-Only Files, 06.58 Track Changes in the Editor, 06.59 Edit Read-Only Files, 06.59 Edit Read-Only Files–Dedicated Style Sheets, Make Writable, 06.60 Choosing CSS Versions–Exploring the Tag Specific Options Dialog Box, Dedicated Style Sheets, 06.61 Understanding Tag Specific Options, Exploring the Tag Specific Options Dialog Box, Debugging–VB, VB, VB, 07.02 Using

[Ctrl+Alt+B to Open the Breakpoints Window](#)–[07.03 Adding Labels to Breakpoints](#), [07.02 Using Ctrl+Alt+B to Open the Breakpoints Window](#), [07.03 Adding Labels to Breakpoints](#), [07.03 Adding Labels to Breakpoints](#), [07.04 Enable or Disable All Breakpoints](#)–[C++](#), [C++](#), [C++](#)–[07.07 Create Code Shortcuts in the Task List](#), [07.06 Create Custom Tokens for the Task List](#), [07.07 Create Code Shortcuts in the Task List](#), [07.07 Create Code Shortcuts in the Task List](#), [07.08 Code Definition Window](#), [07.09 Save Changes Before Building](#)–[07.10 Navigate Errors in the Error List](#), [07.10 Navigate Errors in the Error List](#)–[Column Ordering](#), [07.10 Navigate Errors in the Error List](#), [Column Ordering](#), [07.12 Pin a DataTip to Source Code](#)–[07.13 Create a Floating DataTip](#), [07.13 Create a Floating DataTip](#), [07.13 Create a Floating DataTip](#), [07.14 Adding Comments to a DataTip](#)–[07.15 Use a DataTip to Edit a Value](#), [07.14 Adding Comments to a DataTip](#), [07.15 Use a DataTip to Edit a Value](#), [07.16 DataTip Value from the Last Debug Session](#), [07.19 Searching Breakpoints](#)–[07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#), [07.20 Breakpoint Hit Count](#)–[07.21 Set a Breakpoint on a Function](#), [Break When The Hit Count Is Equal To](#), [07.21 Set a Breakpoint on a Function](#), [07.21 Set a Breakpoint on a Function](#), [07.21 Set a Breakpoint on a Function](#)–[Breakpoints Window](#), [Breakpoints Window](#), [Breakpoints Window](#)–[Special Notes](#), [Special Notes](#), [07.24 Setting a Breakpoint Filter](#), [07.25 Setting a Tracepoint in Source Code](#)–[Continue execution](#), [Continue execution](#), [07.26 Import and Export Breakpoints](#), [07.27 Run to Cursor](#), [07.28 Using the Exception Assistant](#)–[Actions](#), [Actions](#), [07.30 Application and Page Level Tracing](#)–[Combined Tracing](#), [Page Level Tracing](#), [Combined Tracing](#), [Combined Tracing](#), [07.31 The Watch Window: Watching and Changing Values](#)–[What Does It Do?](#), [07.32 Understanding QuickWatch](#)–[07.33 The Watch Window: Visualizers](#), [What Does It Do?](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#)–[07.34 The Watch Window: Refreshing Data](#), [07.33 The Watch Window: Visualizers](#), [07.34 The Watch Window: Refreshing Data](#), [07.35 The Watch Window: Adding Watches from Variable Windows](#)–[07.36 Create Folders in Class View](#), [07.36 Create Folders in Class View](#)–[Creating Subfolders](#), [07.36 Create Folders in Class](#)

[View](#), [Creating Subfolders](#), [07.37 Search in Class](#)  
[View](#)–[View](#).[Class View](#)[Search Command](#), [View](#).[Class View](#)[Search Command](#),  
[07.38 Synchronize Your Class View](#), [07.39 The Misnamed and Misunderstood](#)  
[Object Browser](#), [07.40 The Object Browser: Setting the Browsing](#)  
[Scope](#)–[07.41 The Object Browser: Navigation and References](#), [07.41 The](#)  
[Object Browser: Navigation and References](#)–[References](#), [07.41 The Object](#)  
[Browser: Navigation and References](#), [References](#), [07.42 The Exceptions](#)  
[Dialog Box](#)–[07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43](#)  
[Setting a Breakpoint in the Call Stack Window](#), [07.44 Setting a Tracepoint in](#)  
[the Call Stack Window](#), [07.45 Using the WPF Tree Visualizer](#)–[07.45 Using the](#)  
[WPF Tree Visualizer](#), [07.45 Using the WPF Tree Visualizer](#), [07.47 Changing](#)  
[Context in the Locals Window](#)–[07.48 Understanding the Autos Window](#), [07.48](#)  
[Understanding the Autos Window](#), [07.48 Understanding the Autos](#)  
[Window](#)–[VB Shows Three Statements on Either Side](#), [VB Shows Three](#)  
[Statements on Either Side](#)

[AutoRecover function](#), [01.05 AutoRecover](#)–[01.06 Improving Performance](#)  
[by Changing the Visual Experience in Visual Studio 2010](#), [01.05](#)  
[AutoRecover](#), [01.06 Improving Performance by Changing the Visual](#)  
[Experience in Visual Studio 2010](#)

[Autos window](#), [07.48 Understanding the Autos Window](#)–[VB Shows Three](#)  
[Statements on Either Side](#), [VB Shows Three Statements on Either Side](#)

blank lines, accidentally copying, [06.03 How to Keep from Accidentally](#)  
[Copying a Blank Line](#)

blank lines, inserting above/below current line, [06.09 Insert a Blank Line](#)  
[Above or Below the Current Line](#)

box selections, pasting content between, [06.28 Replacing Text with a Box](#)  
[Selection](#)–[06.30 Pasting a Single Selection into a Box Selection](#), [06.30](#)  
[Pasting a Single Selection into a Box Selection](#)

box selections, zero-length, [06.30 Pasting a Single Selection into a Box](#)  
[Selection](#)–[06.32 View White Space](#), [06.32 View White Space](#)

breakpoint filters, setting, [07.24 Setting a Breakpoint Filter](#)

breakpoint Hit Count, [07.20 Breakpoint Hit Count](#)–[07.21 Set a Breakpoint on a Function](#), [Break When The Hit Count Is Equal To](#), [07.21 Set a Breakpoint on a Function](#), [07.21 Set a Breakpoint on a Function](#)

breakpoints window, opening, [07.02 Using Ctrl+Alt+B to Open the Breakpoints Window](#)

breakpoints, adding labels to, [07.02 Using Ctrl+Alt+B to Open the Breakpoints Window](#)–[07.03 Adding Labels to Breakpoints](#), [07.03 Adding Labels to Breakpoints](#)

breakpoints, enabling/disabling all, [07.04 Enable or Disable All Breakpoints](#)–[C++](#), [C++](#)

breakpoints, importing/exporting, [07.26 Import and Export Breakpoints](#)

breakpoints, setting on functions, [07.21 Set a Breakpoint on a Function](#)–[Breakpoints Window](#), [Breakpoints Window](#)

breakpoints, setting with code, [Debugging](#)–[VB](#), [VB](#), [VB](#)

browsers for web development, [03.11 Using Additional Browsers for Web Development](#)–[Browser Window Size](#), [Browser Window Size](#)

Call Stack window, setting tracepoints in, [07.44 Setting a Tracepoint in the Call Stack Window](#)

changing templates in New Project/Items dialogs, [02.18 Reorganize the Default Project Templates](#)

changing visual experience in Visual Studio 2010, [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

changing Visual Studio color schemes, [01.09 Change Your Visual Studio Color Scheme](#)

Class View, creating folders in, [07.36 Create Folders in Class View](#)–[Creating Subfolders](#), [Creating Subfolders](#)

Class View, searching in, [07.37 Search in Class](#)

[View–View.ClassViewSearch Command](#), [View.ClassViewSearch Command](#)

Class View, synchronizing, [07.38 Synchronize Your Class View](#)

closing only specified files, [04.10 Closing Just the Selected Files You Want](#)

Code Definition window, [07.08 Code Definition Window](#)

Code Snippets Manager, [06.45 Using the Code Snippets Manager](#)

code snippets, creating new from existing, [06.54 Create New Code](#)

[Snippets from Existing Ones](#)–[06.54 Create New Code Snippets from](#)

[Existing Ones](#), [06.54 Create New Code Snippets from Existing Ones](#),

[06.54 Create New Code Snippets from Existing Ones](#)

code snippets, inserting, [06.40 Inserting Code Snippets](#)

code snippets, surrounding existing code with, [06.41 Surround with a Code](#)

[Snippet](#)–[06.41 Surround with a Code Snippet](#), [06.41 Surround with a Code](#)

[Snippet](#)

code snippets, using, [06.42 Using Code Snippets](#)

Collapse To Definitions with outlining, [06.35 Collapse to Definitions with](#)  
[Outlining](#)

command prompt history, [03.16 Command Prompt History](#)

command prompt tab completion, [03.17 Command Prompt Tab](#)

[Completion](#)–[Click and drag](#), [Finally](#), [Click and drag](#)

Commands tab, customizing (toolbars), [01.12 Customize Your Toolbars in](#)  
[Visual Studio 2010: Commands Tab](#)–[01.12 Customize Your Toolbars in](#)  
[Visual Studio 2010: Commands Tab](#), [01.12 Customize Your Toolbars in](#)  
[Visual Studio 2010: Commands Tab](#)

commands, simple, [03.19 Understanding Commands: Simple](#)

[Commands](#)–[03.20 Understanding Commands: Aliases](#), [03.19](#)

[Understanding Commands: Simple Commands](#), [03.19 Understanding Commands: Simple Commands](#), [03.20 Understanding Commands: Aliases](#),  
[03.20 Understanding Commands: Aliases](#)

commenting/uncommenting code in web pages, [06.08 Comment and Uncomment in Web Pages](#)–[06.09 Insert a Blank Line Above or Below the Current Line](#), [06.09 Insert a Blank Line Above or Below the Current Line](#)  
comments, adding to DataTips, [07.14 Adding Comments to a DataTip](#)–[07.15 Use a DataTip to Edit a Value](#), [07.14 Adding Comments to a DataTip](#), [07.15 Use a DataTip to Edit a Value](#)

conditional breakpoints, setting, [Breakpoints Window–Special Notes, Special Notes](#)

creating Web Applications/Virtual Directories in IIS, [02.04 Create Web Application or Virtual Directory in IIS](#)–[02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#)

CSS versions, choosing, [06.60 Choosing CSS Versions–Exploring the Tag Specific Options Dialog Box](#), [Exploring the Tag Specific Options Dialog Box](#)

current line, cutting/deleting, [06.05 Cut or Delete the Current Line](#)–[06.06 Using the New IntelliSense: Keywords](#), [06.06 Using the New IntelliSense: Keywords](#)

custom file extension associations, [04.13 Using Custom File Extension Associations](#)–[04.13 Using Custom File Extension Associations](#), [04.13 Using Custom File Extension Associations](#)

custom item templates, organizing, [02.15 Organizing Your Custom Item Templates](#)–[02.16 Organizing Your Custom Project Templates](#), [02.16 Organizing Your Custom Project Templates](#)

custom project templates, organizing, [02.16 Organizing Your Custom](#)

[Project Templates](#)–[02.17 Reorganize the Default Item Templates, 02.16 Organizing Your Custom Project Templates, 02.16 Organizing Your Custom Project Templates, 02.17 Reorganize the Default Item Templates, 02.17 Reorganize the Default Item Templates](#)

cut/copy/paste collapsed code with outlining, [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#)

cycling through open tool windows, [03.03 Cycle Through Your Open Tool Windows](#)

DataTip values, viewing from last debug session, [07.16 DataTip Value from the Last Debug Session](#)

DataTips, pinning to source code, [07.12 Pin a DataTip to Source Code](#)–[07.13 Create a Floating DataTip, 07.13 Create a Floating DataTip](#)

default view in HTML editor, changing, [06.26 Change the Default View in the HTML Editor](#)–[06.28 Replacing Text with a Box Selection, 06.28 Replacing Text with a Box Selection](#)

Design and Source views in web projects, [06.24 Switch Between Design and Source in Web Projects](#)–[06.26 Change the Default View in the HTML Editor, 06.26 Change the Default View in the HTML Editor](#)

Design view, toggling, [06.25 Toggle Designer](#)

displaying HTML/CSS warnings as errors, [06.51 Display HTML/CSS Warnings as Errors](#)

docking floating tool windows to previous location, [03.02 Dock a Floating Tool Window Back to Its Previous Location](#)

Document Outline (web projects), [06.39 Document Outline: Web Projects](#)–[06.40 Inserting Code Snippets, 06.40 Inserting Code Snippets](#)

documents on multiple monitors, [04.03 Working with Documents on Multiple Monitors](#)–[04.03 Working with Documents on Multiple Monitors, 04.03 Working with Documents on Multiple Monitors](#)

editing read-only files, [06.59 Edit Read-Only Files–Dedicated Style Sheets, Make Writable, Dedicated Style Sheets](#)

Esc for returning to Editor return to, [06.27 Jump Back to the Editor from Just About Anywhere](#)

Exception Assistant, [07.28 Using the Exception Assistant–Actions, Actions](#)

Exceptions dialog box, [07.42 The Exceptions Dialog Box–07.43 Setting a Breakpoint in the Call Stack Window, 07.43 Setting a Breakpoint in the Call Stack Window](#)

Export Template Wizard, [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

exporting environment settings, [01.03 Exporting Your Environment Settings–01.03 Exporting Your Environment Settings, 01.03 Exporting Your Environment Settings](#)

exporting external tools list, [03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro, 03.34 Creating and Using a Macro](#)

exporting window layouts, [03.29 Export Your Window Layouts](#)

external tools, running, [03.31 Using External Tools–Prompt For Arguments, 03.31 Using External Tools, Prompt For Arguments](#)

file open location, [04.11 Understanding the File Open Location–04.12 Show Previous Versions, 04.12 Show Previous Versions](#)

Find Combo box keyboard shortcuts, [05.06 Using the Find Combo Box Keyboard Shortcuts–05.07 Using Incremental Search, 05.07 Using Incremental Search](#)

Find in Files, [05.09 Find In Files: Find Options–Navigation, 05.10 Find In Files: Result Options–Display File Names Only, Navigation, Display File Names Only](#)

**Find in Files search results, customizing, [05.17 Customize Results in Find In Files Searches–Char, Char](#)**

**Find Symbol, [05.14 Understanding Find Symbol](#)–[05.15 Find Symbol Results Shortcuts](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#), [05.15 Find Symbol Results Shortcuts](#)**

**Find Symbol results shortcuts, [05.15 Find Symbol Results Shortcuts](#)–[05.16 Replace in Files: Tagged Expressions](#), [05.16 Replace in Files: Tagged Expressions](#)**

**finding keyboard shortcuts in Visual Studio, [03.24 Find Keyboard Shortcuts](#)**

**floating DataTips, [07.13 Create a Floating DataTip](#)**

**formatting documents/selections for HTML, [06.47 Format the Current Document or Selection \(Web\)](#)**

**formatting on HTML paste, [06.50 Format HTML on Paste](#)**

**Go Back markers, navigating with, [06.56 Navigate Backward and Navigate Forward Using Go Back Markers](#)**

**Guide Diamond, rearranging windows in Visual Studio 2010 with, [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#)**

**Hide Selection, [06.34 Using Hide Selection](#)–[06.34 Using Hide Selection](#), [06.34 Using Hide Selection](#)**

**HTML code snippets, [06.43 HTML Code Snippets](#)–[06.44 JavaScript Code Snippets](#), [06.44 JavaScript Code Snippets](#)**

**HTML Editor tag navigation, [06.49 HTML Editor Tag Navigation](#)–[06.51 Display HTML/CSS Warnings as Errors](#), [06.50 Format HTML on Paste](#), [06.51 Display HTML/CSS Warnings as Errors](#)**

**Image Library (Visual Studio), [03.35 Visual Studio Image Library](#)**

**importing/changing environment settings, [01.08 Importing or Changing](#)**

[Your Environment Settings](#)–[01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#)

incremental search, [05.07 Using Incremental Search](#)–[05.08 Search the Currently Selected String Without the Find Window](#), [05.08 Search the Currently Selected String Without the Find Window](#)

inserting documents to right of existing tabs, [04.01 Insert Documents to the Right of Existing Tabs](#)–[04.02 Recent Files](#), [04.02 Recent Files](#)

inserting quotes when typing attribute values, [06.46 Insert Quotes When Typing Attribute Values](#)

Intellisense keywords, [06.06 Using the New IntelliSense: Keywords](#)–[06.07 Using the New IntelliSense: Pascal Case](#), [06.07 Using the New IntelliSense: Pascal Case](#), [06.07 Using the New IntelliSense: Pascal Case](#)

Intellisense, making transparent, [06.04 Make IntelliSense Transparent](#)

item templates, creating, [02.13 Create Your Own Item Template](#)

JScript Intellisense, updating, [06.52 Updating JScript IntelliSense](#)

JScript libraries, using in JScript files, [06.53 Using JScript Libraries in Other JScript Files](#)

keyboard access to tool window toolbar, [03.15 Keyboard Access to a Tool Window's Toolbar](#)

Keyboard Mapping Schemes, [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)–[03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#)

Locals window, changing context in, [07.47 Changing Context in the Locals Window](#)–[07.48 Understanding the Autos Window](#), [07.48 Understanding the Autos Window](#)

logging commands, [03.28 Understanding Commands: Logging Commands](#)

matching braces, selecting/moving between, [06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)](#)

moving tool windows with keyboard, [03.14 Moving Tool Windows Around with Your Keyboard](#)–[03.15 Keyboard Access to a Tool Window's Toolbar](#), [03.15 Keyboard Access to a Tool Window's Toolbar](#)

multiple start-up projects, [02.05 Multiple Startup Projects](#)–[02.06 Change the Default New Project Location](#), [02.06 Change the Default New Project Location](#)

Navigate To dialog, [05.13 How to Use Navigate To](#)–[05.14 Understanding Find Symbol](#), [05.14 Understanding Find Symbol](#)

navigating among tabs in Toolbox, [03.07 Navigate Among Tabs in the Toolbox](#)

navigating errors in Errors List, [07.10 Navigate Errors in the Error List](#)–[Column Ordering](#), [Column Ordering](#)

navigating open document windows, [04.04 Navigate Open Document Windows](#)–[04.06 Open a File Location from the File Tab](#), [04.06 Open a File Location from the File Tab](#)

navigating property tabs in project properties, [02.10 Navigating Property Tabs in the Project Properties](#)

Navigation Bar, using, [06.48 Using the Navigation Bar](#)–[06.50 Format HTML on Paste](#), [06.50 Format HTML on Paste](#)

navigation stack, [06.55 Understanding the Navigation Stack](#)–[06.55 Understanding the Navigation Stack](#), [06.55 Understanding the Navigation Stack](#)

Object Browser navigation/references, [07.41 The Object Browser: Navigation and References](#)–[References](#), [References](#)

Object Browser overview, [07.39 The Misnamed and Misunderstood](#)

## Object Browser

Object Browser scope settings, [07.40 The Object Browser: Setting the Browsing Scope](#)–[07.41 The Object Browser: Navigation and References](#), [07.41 The Object Browser: Navigation and References](#)

opening file location from file tab, [04.06 Open a File Location from the File Tab](#)

outlining, collapsing code with, [06.33 Collapsing Your Code with Outlining](#)–[Click Anywhere in Area \(Menu Item\)](#), [Click Anywhere in Area \(Menu Item\)](#)

parameter information, [06.18 Using Parameter Information](#)

Pascal case (Intellisense), [06.07 Using the New IntelliSense: Pascal Case](#)

pasting single selection into box selections, [06.30 Pasting a Single Selection into a Box Selection](#)

pinning projects to Recent Projects list, [02.11 Pin a Project to the Recent Projects List](#)

Properties window, [06.38 Properties Window Keyboard Shortcuts](#)–[Working with Categories](#), [Working with Categories](#)

Quick Find, [05.02 Using Quick Find](#)–[05.03 Using a Simple Quick Replace](#), [05.03 Using a Simple Quick Replace](#)

Quick Replace (searching), [05.03 Using a Simple Quick Replace](#)–[05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.03 Using a Simple Quick Replace](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#)

QuickWatch, [07.32 Understanding QuickWatch](#)–[07.33 The Watch Window: Visualizers](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#)

recent files, [\*\*04.02 Recent Files\*\*](#)

removing projects from Recent Projects list, [\*\*01.04 Remove Projects from the Recent Projects List\*\*](#)

Replace in Files basic options, [\*\*05.11 Replace In Files: Basic Options\*\*](#)–[\*\*05.12 Go To Definition for Cascading Style Sheets\*\*](#), [\*\*05.12 Go To Definition for Cascading Style Sheets\*\*](#)

ResetSettings switch, [\*\*01.15 The ResetSettings Switch\*\*](#)—[\*\*Same Machine\*\*](#), [\*\*Two Different Machines, Same Machine\*\*](#)

resetting all keyboard shortcuts, [\*\*03.27 Keyboard Shortcuts: Reset All Your Shortcuts\*\*](#)

Run To Cursor, [\*\*07.27 Run to Cursor\*\*](#)

running multiple versions of Visual Studio side-by-side, [\*\*01.01 Running Multiple Versions of Visual Studio Side-By-Side\*\*](#)

saving changes before building, [\*\*07.09 Save Changes Before Building\*\*](#)–[\*\*07.10 Navigate Errors in the Error List\*\*](#), [\*\*07.10 Navigate Errors in the Error List\*\*](#)

searching breakpoints, [\*\*07.19 Searching Breakpoints\*\*](#)–[\*\*07.19 Searching Breakpoints\*\*](#), [\*\*07.19 Searching Breakpoints\*\*](#), [\*\*07.19 Searching Breakpoints\*\*](#)

searching currently selected string without Find window, [\*\*05.08 Search the Currently Selected String Without the Find Window\*\*](#)–[\*\*05.09 Find In Files: Find Options\*\*](#), [\*\*05.09 Find In Files: Find Options\*\*](#)

searching for project templates, [\*\*Projects and Items\*\*](#)–[\*\*02.03 Using Older Frameworks with Multi-Targeting\*\*](#), [\*\*02.02 Recent Project Templates in the New Project Dialog Box\*\*](#), [\*\*02.03 Using Older Frameworks with Multi-Targeting\*\*](#)

searching in Toolbox, [\*\*03.06 Searching in the Toolbox\*\*](#)

selecting from current cursor to last Go Back marker, [\*\*06.57 Select from\*\*](#)

[the Current Cursor Location to the Last Go Back Marker](#)–[06.57 Select from the Current Cursor Location to the Last Go Back Marker](#), [06.57 Select from the Current Cursor Location to the Last Go Back Marker](#)

shortcuts, creating new, [03.26 Keyboard Shortcuts: Creating New Shortcuts](#)

show previous file versions, [04.12 Show Previous Versions](#)

showing hidden tool windows with Auto Hide Channel, [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#)–[03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#)

Solution Folders, [02.09 Using Solution Folders](#)–[02.11 Pin a Project to the Recent Projects List](#), [02.11 Pin a Project to the Recent Projects List](#)

tabs in Toolbox, [03.10 Working with Tabs in the Toolbox](#)

Tag Specific options, [06.61 Understanding Tag Specific Options](#)

Task Lists, creating custom tokens for, [C++–07.07 Create Code Shortcuts in the Task List](#), [07.06 Create Custom Tokens for the Task List](#), [07.07 Create Code Shortcuts in the Task List](#), [07.07 Create Code Shortcuts in the Task List](#)

text, replacing text with box selection, [06.28 Replacing Text with a Box Selection](#)

tool window animations, changing, [01.07 Change Tool Window Animations](#)  
Toolbars tab, customizing, [01.11 Customize Your Toolbars in Visual Studio 2010: Toolbars Tab](#)–[Custom Toolbars](#), [Custom Toolbars](#)

Toolbox, drag and drop into, [06.21 Drag and Drop Code into the Toolbox](#)–[06.21 Drag and Drop Code into the Toolbox](#), [06.21 Drag and Drop Code into the Toolbox](#), [06.21 Drag and Drop Code into the Toolbox](#)

Toolbox, expanding/collapsing all in, [03.05 Expand and Collapse All in the Toolbox](#)

tracepoints, setting in source code, [07.25 Setting a Tracepoint in Source Code](#)–[Continue execution](#), [Continue execution](#)

tracing, application/page level, [07.30 Application and Page Level Tracing](#)–[Combined Tracing](#), [Page Level Tracing](#), [Combined Tracing](#), [Combined Tracing](#)

tracking changes in Editor, [06.58 Track Changes in the Editor](#)–[06.59 Edit Read-Only Files](#), [06.58 Track Changes in the Editor](#), [06.59 Edit Read-Only Files](#)

using older frameworks with multi-targeting, [02.03 Using Older Frameworks with Multi-Targeting](#)

using smart tags from keyboards, [06.22 Using Smart Tags from the Keyboard](#)–[Remove Unused Usings](#), [06.22 Using Smart Tags from the Keyboard](#), [Remove Unused Usings](#)

using statements, organizing (C#), [Remove Unused Usings](#)

Visual Studio logging, [01.13 Visual Studio Logging](#)–[01.14 Visual Studio Safe Mode](#), [01.14 Visual Studio Safe Mode](#), [01.14 Visual Studio Safe Mode](#)

Visual Studio Online Help, table of contents in, [01.02 Getting Table of Contents in Visual Studio 2010 Online Help](#)

visualizers (Watch windows), [07.33 The Watch Window: Visualizers](#)–[07.34 The Watch Window: Refreshing Data](#), [07.34 The Watch Window: Refreshing Data](#)

Watch window, [07.31 The Watch Window: Watching and Changing Values](#)–[What Does It Do?](#), [What Does It Do?](#)

watches, adding from variable windows, [07.35 The Watch Window:](#)

[Adding Watches from Variable Windows](#)–[07.36 Create Folders in Class View](#), [07.36 Create Folders in Class View](#)

white space, viewing, [06.32 View White Space](#)–[06.33 Collapsing Your Code with Outlining](#), [06.33 Collapsing Your Code with Outlining](#)

window layouts, [03.08 Window Layouts: The Four Modes](#), [03.09 Window Layouts: Design, Debug, and Full Screen](#)

word completion, [06.20 Word Completion](#)

word wrap, [06.37 Understanding Word Wrap](#)–[06.38 Properties Window Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window Keyboard Shortcuts](#)

WPF Tree Visualizer, [07.45 Using the WPF Tree Visualizer](#)–[07.45 Using the WPF Tree Visualizer](#), [07.45 Using the WPF Tree Visualizer](#)

collapsing code, [06.35 Collapse to Definitions with Outlining](#)–[06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#), [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#), [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#)

collapse to definitions with outlining, [06.35 Collapse to Definitions with Outlining](#)–[06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#), [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#)

cut/copy/paste with outlining, [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#)

color schemes, changing, [01.09 Change Your Visual Studio Color Scheme](#)–[Changing Your Colors](#), [Changing Your Colors](#)–[Resetting the Colors](#), [Changing Your Colors](#), [Changing Your Colors](#), [Resetting the Colors](#), [Resetting the Colors](#)

colors, importing, [Changing Your Colors](#)–[Resetting the Colors](#), [Changing Your Colors](#), [Resetting the Colors](#), [Resetting the Colors](#)

obtaining fonts and colors, [01.09 Change Your Visual Studio Color](#)

Scheme–Changing Your Colors, Changing Your Colors resetting colors, Resetting the Colors colors, Visual Studio Color Theme Editor–08.02 Insert Images into Your Code, To Customize, 08.02 Insert Images into Your Code Color Theme Editor extension (Visual Studio), Visual Studio Color Theme Editor–08.02 Insert Images into Your Code, To Customize, 08.02 Insert Images into Your Code column ordering in tools window, 07.11 Ordering and Multicolumn Sorting in Tool Windows–07.12 Pin a DataTip to Source Code, Multicolumn Sorting, 07.12 Pin a DataTip to Source Code combo box, Zoom, Combo Box command prompt, 03.16 Command Prompt History–Wildcard Search, 03.17 Command Prompt Tab Completion–Click and drag, Wildcard Search, Wildcard Search, Finally, Click and drag history, 03.16 Command Prompt History–Wildcard Search, Wildcard Search tab completion, 03.17 Command Prompt Tab Completion–Click and drag, Wildcard Search, Finally, Click and drag Command Window, 01.08 Importing or Changing Your Environment Settings, 03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro, 03.33 Exporting Your Command Window Aliases and External Tools List, 03.34 Creating and Using a Macro aliases, exporting, 03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro, 03.33 Exporting Your Command Window Aliases and External Tools List, 03.34 Creating and Using a Macro aliases, setting, 01.08 Importing or Changing Your Environment Settings

commands, [01.12 Customize Your Toolbars in Visual Studio 2010: Commands Tab](#), [03.03 Cycle Through Your Open Tool Windows](#), [03.19 Understanding Commands: Simple Commands](#)–[03.20 Understanding Commands: Aliases](#), [03.20 Understanding Commands: Aliases](#)–[03.21 Understanding Commands: Arguments and Switches](#), [03.20 Understanding Commands: Aliases](#), [Create a New Alias](#), [03.21 Understanding Commands: Arguments and Switches](#), [/overwrite](#), [Example](#)

aliases, [03.20 Understanding Commands: Aliases](#)–[03.21 Understanding Commands: Arguments and Switches](#), [Create a New Alias](#), [03.21 Understanding Commands: Arguments and Switches](#)

Commands tab, customizing (toolbars), [01.12 Customize Your Toolbars in Visual Studio 2010: Commands Tab](#)

hidden Command tool window, [03.03 Cycle Through Your Open Tool Windows](#)

logging, [/overwrite](#), [Example](#)

simple, [03.19 Understanding Commands: Simple Commands](#)–[03.20 Understanding Commands: Aliases](#), [03.20 Understanding Commands: Aliases](#)

comments, [Modify Selection](#)

Comment/Uncomment items (Modify Selection), [Modify Selection](#)

Common Elements source files (VS Image Library), [Common Elements](#), [Common Elements](#)

Common tab/All tab in statement completion, [06.17 Move Between the Common Tab and All Tab in Statement Completion \(VB\)](#)

conditional breakpoints, [07.22 Set a Simple Breakpoint Condition](#), [Special Notes](#), [07.23 Set a Complex Breakpoint Condition](#)

controls, [Controls](#)–[Buttons](#), [Buttons](#), [Buttons](#), [Modify Selection](#), [Custom](#)

[Controls, Control box \(Visual Studio 2010 only\)](#)

[Control box \(Visual Studio 2010\), Control box \(Visual Studio 2010 only\)](#)

Controls area for modifying menus/toolbars, [Controls–Buttons, Buttons](#), [Buttons](#)

Controls dialog box (Commands tab), [Modify Selection](#)  
custom (Toolbox), [Custom Controls](#)

copying, [Copy \(Ctrl+C\)](#), [06.03 How to Keep from Accidentally Copying a Blank Line, Actions](#)

blank lines accidentally, [06.03 How to Keep from Accidentally Copying a Blank Line](#)

[Copy \(Ctrl+C\) command \(Find Symbol\)](#), [Copy \(Ctrl+C\)](#)

[Copy Exception Detail To The Clipboard link \(Exception Assistant\)](#),  
[Actions](#)

current and previous statements (Autos window), [Simple Example](#)

Current Project action, [05.02 Using Quick Find](#)

[Quick Find, 05.02 Using Quick Find](#)

Custom Component Set, [Custom Component Set–07.41 The Object Browser: Navigation and References](#), [07.41 The Object Browser: Navigation and References](#)

editing, [Custom Component Set–07.41 The Object Browser: Navigation and References](#), [07.41 The Object Browser: Navigation and References](#)

customizing, [Custom Toolbars](#), [Custom Toolbars](#), [Custom Toolbars](#), [02.15 Organizing Your Custom Item Templates](#), [02.15 Organizing Your Custom Item Templates](#), [02.16 Organizing Your Custom Project Templates](#), [Custom Controls](#), [04.13 Using Custom File Extension Associations–04.13 Using Custom File Extension Associations](#), [04.13 Using Custom File Extension Associations](#), [Display File Names Only](#), [05.17 Customize Results in Find In](#)

[Files Searches](#), [To Customize the Document Tab Well User Interface–08.21](#)  
[Create and Find Code Snippets](#), [To Customize the Document Tab Well User Interface](#), [08.21 Create and Find Code Snippets](#), [More Information](#), [08.23 Customize Visual Studio Using Windows PowerShell](#)–[To Customize](#), [To Use](#), [To Use](#), [To Customize](#), [More Information](#)

custom controls (Toolbox), [Custom Controls](#)

custom file extension associations, [04.13 Using Custom File Extension Associations](#)–[04.13 Using Custom File Extension Associations](#), [04.13 Using Custom File Extension Associations](#)

custom item templates, organizing, [02.15 Organizing Your Custom Item Templates](#), [02.15 Organizing Your Custom Item Templates](#), [02.16 Organizing Your Custom Project Templates](#)

Customize dialog box, [Custom Toolbars](#)

Document Tab Well user interface, [To Customize the Document Tab Well User Interface–08.21](#) [Create and Find Code Snippets](#), [To Customize the Document Tab Well User Interface](#), [08.21 Create and Find Code Snippets](#)

Find in Files search results, [05.17 Customize Results in Find In Files Searches](#)

GhostDoc extension (VS), [More Information](#)

StudioShell module, [More Information](#)

Toolbars tab, [Custom Toolbars](#), [Custom Toolbars](#)

Visual Studio with Windows PowerShell, [08.23 Customize Visual Studio Using Windows PowerShell](#)–[To Customize](#), [To Use](#), [To Use](#), [To Customize](#)

“Customize How Find In Files Results Are Displayed in the Find Results Window”, [Display File Names Only](#)

cycling through Clipboard Ring, [06.11 How to Cycle Through the Clipboard Ring](#)–[06.12 Using the Undo and Redo Stack](#), [06.12 Using the Undo and Redo](#)

## **Stack**

## D

data, refreshing (Watch window), [07.34 The Watch Window: Refreshing Data–Two threads, Two threads, Refreshing the data](#)

DataTips, [07.13 Create a Floating DataTip–07.14 Adding Comments to a DataTip](#), [07.13 Create a Floating DataTip](#), [07.13 Create a Floating DataTip](#), [07.14 Adding Comments to a DataTip–07.15 Use a DataTip to Edit a Value](#), [07.14 Adding Comments to a DataTip](#), [07.14 Adding Comments to a DataTip](#), [07.15 Use a DataTip to Edit a Value](#)

adding comments to, [07.14 Adding Comments to a DataTip–07.15 Use a DataTip to Edit a Value](#), [07.14 Adding Comments to a DataTip](#), [07.15 Use a DataTip to Edit a Value](#)

creating floating, [07.13 Create a Floating DataTip–07.14 Adding Comments to a DataTip](#), [07.13 Create a Floating DataTip](#), [07.14 Adding Comments to a DataTip](#)

pinning to source code, [07.13 Create a Floating DataTip](#)

debugging, [03.08 Window Layouts: The Four Modes](#), [Debug Mode](#), [07.08 Code Definition Window](#), [Prompt To Save All Changes](#), [Column Ordering](#), [Multicolumn Sorting](#), [07.25 Setting a Tracepoint in Source Code–Continue execution, Setting Tracepoints](#), [Continue execution](#), [Continue execution](#), [07.27 Run to Cursor](#), [07.28 Using the Exception Assistant–Actions](#), [Actions](#), [07.29 Use a Specific Port for the Development Server \(Web Applications\)](#), [07.30 Application and Page Level Tracing–Combined Tracing](#), [Page Level Tracing](#), [Combined Tracing](#), [Combined Tracing](#), [07.31 The Watch Window: Watching and Changing Values](#), [Type it in](#), [Add Watch](#), [07.32 Understanding QuickWatch–07.33 The Watch Window: Visualizers](#), [07.32 Understanding QuickWatch](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#), [Keyboard Mapping](#), [Keyboard Mapping](#), [Create a New Folder](#), [Removing Items from Folders](#), [Use a Previous Search](#), [07.38 Synchronize Your Class View](#), [07.42 The Exceptions Dialog Box–Special Case](#), [Special Case](#), [07.43](#)

[Setting a Breakpoint in the Call Stack Window](#)–[07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#), [07.44 Setting a Tracepoint in the Call Stack Window](#)–[07.45 Using the WPF Tree Visualizer](#), [07.44 Setting a Tracepoint in the Call Stack Window](#), [07.45 Using the WPF Tree Visualizer](#), [07.47 Changing Context in the Locals Window](#)–[Simple Example](#), [Debug Location Toolbar](#), [Debug Location Toolbar](#), [Stack Frame](#), [07.48 Understanding the Autos Window](#)–[Another Example](#), [Changing Values](#), [Simple Example](#), [Another Example](#)

application-level/page-level tracing, [07.30 Application and Page Level Tracing](#)–[Combined Tracing](#), [Page Level Tracing](#), [Combined Tracing](#), [Combined Tracing](#)

Autos window, [07.48 Understanding the Autos Window](#)–[Another Example](#), [Another Example](#)

Call Stack window, setting breakpoints in, [07.43 Setting a Breakpoint in the Call Stack Window](#)–[07.43 Setting a Breakpoint in the Call Stack Window](#), [07.43 Setting a Breakpoint in the Call Stack Window](#)

changing context in Locals window, [07.47 Changing Context in the Locals Window](#)–[Simple Example](#), [Debug Location Toolbar](#), [Stack Frame](#), [Changing Values](#), [Simple Example](#)

Class View, creating folders in, [Create a New Folder](#), [Removing Items from Folders](#)

Code Definition window, [07.08 Code Definition Window](#)

Debug Location toolbar, [Debug Location Toolbar](#)

Debug Mode (window layouts), [03.08 Window Layouts: The Four Modes](#), [Debug Mode](#)

Debug.AddWatch command, [Keyboard Mapping](#)

development server, using specific port for, [07.29 Use a Specific Port for the Development Server \(Web Applications\)](#)

Exception Assistant, [07.28 Using the Exception Assistant–Actions](#), [Actions](#)

Exceptions dialog box, [07.42 The Exceptions Dialog Box–Special Case](#), [Special Case](#)

multicolumn sorting, [Multicolumn Sorting](#)

navigating errors in Errors List, [Column Ordering](#)

QuickWatch, [07.32 Understanding QuickWatch](#)–[07.33 The Watch Window: Visualizers](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#)

Run To Cursor, [07.27 Run to Cursor](#)

saving changes before building, [Prompt To Save All Changes](#)

searching in Class View, [Use a Previous Search](#), [07.38 Synchronize Your Class View](#)

setting tracepoints in source code, [07.25 Setting a Tracepoint in Source Code–Continue execution](#), [Setting Tracepoints](#), [Continue execution](#), [Continue execution](#)

tracepoints, setting in Call Stack window, [07.44 Setting a Tracepoint in the Call Stack Window](#)–[07.45 Using the WPF Tree Visualizer](#), [07.44 Setting a Tracepoint in the Call Stack Window](#), [07.45 Using the WPF Tree Visualizer](#), [07.45 Using the WPF Tree Visualizer](#)

Watch window, [07.31 The Watch Window: Watching and Changing Values](#), [Type it in](#), [Add Watch](#), [07.32 Understanding QuickWatch](#)

watches, adding from variable windows, [Keyboard Mapping](#)

WPF Tree Visualizer, [07.45 Using the WPF Tree Visualizer](#)–[07.45 Using the WPF Tree Visualizer](#), [07.45 Using the WPF Tree Visualizer](#)

dedicated style sheets (CSS), [Dedicated Style Sheets](#)

default browsers, changing, [Changing the Default Browser](#)

default development settings, [01.10 Reset All Your Development Settings](#)

default item templates, reorganizing, [02.17 Reorganize the Default Item Templates](#)–[02.18 Reorganize the Default Project Templates](#), [02.17 Reorganize the Default Item Templates](#), [02.18 Reorganize the Default Project Templates](#), [02.18 Reorganize the Default Project Templates](#)

default New Project location, changing, [02.06 Change the Default New Project Location](#)–[02.07 Track Active Item in Solution Explorer](#), [02.06 Change the Default New Project Location](#), [02.07 Track Active Item in Solution Explorer](#)

default project templates, reorganizing, [02.18 Reorganize the Default Project Templates](#)–[02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.18 Reorganize the Default Project Templates](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)

default toolbars, [Custom Toolbars](#)

DefaultFileOpenLocation (registry), [04.11 Understanding the File Open Location](#)

DefaultTracepointMessage string value, [Change Default Message](#)

deleting, [Custom Toolbars](#), [Renaming Tabs](#), [Removing Browsers](#), [Remove Unused Usings](#)–[Sort Usings](#), [Remove Unused Usings](#), [Sort Usings](#), [Sort Usings](#), [Putting Items into Your Folder](#)

browsers, [Removing Browsers](#)

custom toolbars, [Custom Toolbars](#)

folders in Class View, [Putting Items into Your Folder](#)

tabs (Toolbox), [Renaming Tabs](#)

unused using directives, [Remove Unused Usings](#)–[Sort Usings](#), [Remove Unused Usings](#), [Sort Usings](#), [Sort Usings](#)

design, [03.09 Window Layouts: Design, Debug, and Full Screen](#), [03.24 Find Keyboard Shortcuts](#)–[03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)

Design Mode (window layouts), [03.09 Window Layouts: Design, Debug, and Full Screen](#)

Designer, viewing code in, [03.24 Find Keyboard Shortcuts](#)–[03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)

Design View, [03.08 Window Layouts: The Four Modes](#)

window layouts, [03.08 Window Layouts: The Four Modes](#)

detachable document windows, [04.03 Working with Documents on Multiple Monitors](#)

development servers, using specific port for, [07.29 Use a Specific Port for the Development Server \(Web Applications\)](#)

development settings, resetting, [01.10 Reset All Your Development Settings](#)–[Custom Toolbars](#), [01.10 Reset All Your Development Settings](#), [Custom Toolbars](#)

devenv.com command, [02.17 Reorganize the Default Item Templates](#), [02.18 Reorganize the Default Project Templates](#)

Display File Names Only (Find Results window), [Display File Names Only](#) /doc or /d switch, [Switches](#)

docking, [03.14 Moving Tool Windows Around with Your Keyboard](#), [04.03 Working with Documents on Multiple Monitors](#)

Dock As Tabbed Document option, [04.03 Working with Documents on Multiple Monitors](#)

**Dock menu, 03.14 Moving Tool Windows Around with Your Keyboard**  
**documents, 04.02 Recent Files–04.03 Working with Documents on Multiple Monitors, 04.03 Working with Documents on Multiple Monitors, 04.06 Open a File Location from the File Tab, 04.08 Using the IDE Navigator, 04.10 Closing Just the Selected Files You Want, 04.11 Understanding the File Open Location–04.12 Show Previous Versions, 04.11 Understanding the File Open Location, 04.12 Show Previous Versions, 04.13 Using Custom File Extension Associations–04.13 Using Custom File Extension Associations, 04.13 Using Custom File Extension Associations, 06.49 HTML Editor Tag Navigation, Save Changes To Open Documents Only, To Customize the Document Tab Well User Interface–08.21 Create and Find Code Snippets, 08.21 Create and Find Code Snippets, To Use**

closing only specified, **04.10 Closing Just the Selected Files You Want**  
**custom document extension associations, 04.13 Using Custom File Extension Associations–04.13 Using Custom File Extension Associations, 04.13 Using Custom File Extension Associations**

document open location, **04.11 Understanding the File Open Location–04.12 Show Previous Versions, 04.11 Understanding the File Open Location, 04.12 Show Previous Versions**

Document Tab Well user interface, **To Customize the Document Tab Well User Interface–08.21 Create and Find Code Snippets, 08.21 Create and Find Code Snippets**

Document This command, **To Use**

IDE navigator, **04.08 Using the IDE Navigator**

navigating open document windows, **04.06 Open a File Location from the File Tab**

recent, **04.02 Recent Files–04.03 Working with Documents on Multiple Monitors, 04.03 Working with Documents on Multiple Monitors**

saving changes to open, [Save Changes To Open Documents Only](#)

“Document Outline: Web Project”, [06.49 HTML Editor Tag Navigation](#)

drag and drop code into Toolbox, [06.21 Drag and Drop Code into the Toolbox](#)—[06.21 Drag and Drop Code into the Toolbox](#), [06.21 Drag and Drop Code into the Toolbox](#), [06.21 Drag and Drop Code into the Toolbox](#)

Dynamic Tab Wells, [To Customize the Document Tab Well User Interface](#)

# E

editing, [03.21 Understanding Commands: Arguments and Switches–Using the Arguments and Switches](#), [List Current Options](#), [Using the Arguments and Switches](#), [Custom Component Set \(Find Symbol\)](#), [Custom Component Set–07.41 The Object Browser: Navigation and References](#), [Edit Custom Component Set](#), [Edit Custom Component Set, 07.41 The Object Browser: Navigation and References](#)

[Custom Component Set](#), [Custom Component Set–07.41 The Object Browser: Navigation and References](#), [Edit Custom Component Set](#), [Edit Custom Component Set, 07.41 The Object Browser: Navigation and References](#)

[Edit Custom Component Set dialog box](#), [Custom Component Set \(Find Symbol\)](#)

[Edit.Find command](#), [03.21 Understanding Commands: Arguments and Switches–Using the Arguments and Switches](#), [List Current Options](#), [Using the Arguments and Switches](#)

[Editor](#), [Rearrange](#), [06.27 Jump Back to the Editor from Just About Anywhere](#), [08.03 Add Visual Guidelines to Your Code–08.05 Sync the Solution Explorer to the Current File](#), [To Use](#), [To Use](#), [08.05 Sync the Solution Explorer to the Current File](#), [Share Bad Code with the World](#), [08.12 Run VIM Commands in the Editor–08.14 Zoom Across All Files](#), [08.14 Zoom Across All Files](#)

[Editor Context Menu \(toolbars\)](#), [Rearrange](#)

[Editor Guideline extension \(VS\)](#), [08.03 Add Visual Guidelines to Your Code–08.05 Sync the Solution Explorer to the Current File](#), [To Use](#), [To Use](#), [08.05 Sync the Solution Explorer to the Current File](#)

[running VIM commands in](#), [08.12 Run VIM Commands in the Editor–08.14 Zoom Across All Files](#), [08.14 Zoom Across All Files](#)

using Emac commands in, [Share Bad Code with the World](#)

using Esc to return to, [06.27 Jump Back to the Editor from Just About Anywhere](#)

Emacs Commands extension (VS), [08.07 Use Emacs Commands in the Editor–Share Bad Code with the World](#), [To Uninstall](#), [Share Bad Code with the World](#)

embedded styles (HTML), [Embedded Styles–Exploring the Tag Specific Options Dialog Box](#), [Exploring the Tag Specific Options Dialog Box](#)

Enable Rich Client Visual Experience option, [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

environment settings, [01.03 Exporting Your Environment Settings–01.04 Remove Projects from the Recent Projects List](#), [01.03 Exporting Your Environment Settings](#), [01.04 Remove Projects from the Recent Projects List](#)

exporting, [01.03 Exporting Your Environment Settings–01.04 Remove Projects from the Recent Projects List](#), [01.03 Exporting Your Environment Settings](#), [01.04 Remove Projects from the Recent Projects List](#)

errors, [03.03 Cycle Through Your Open Tool Windows](#)

Error List tool window, [03.03 Cycle Through Your Open Tool Windows](#)

Esc for returning to Editor, [06.27 Jump Back to the Editor from Just About Anywhere](#)

Exception Assistant, [07.28 Using the Exception Assistant–Actions](#), [Actions Exceptions dialog box](#), [07.42 The Exceptions Dialog Box–07.43 Setting a Breakpoint in the Call Stack Window](#), [Special Case](#), [07.43 Setting a Breakpoint in the Call Stack Window](#)

exporting, [01.03 Exporting Your Environment Settings–01.04 Remove Projects from the Recent Projects List](#), [01.03 Exporting Your Environment](#)

Settings, 01.04 Remove Projects from the Recent Projects List, 01.10 Reset All Your Development Settings, 01.10 Reset All Your Development Settings, 01.15 The ResetSettings Switch, 02.13 Create Your Own Item Template, 02.13 Create Your Own Item Template, 02.14 Roll Your Own Project Template with the Export Template Wizard, 02.14 Roll Your Own Project Template with the Export Template Wizard, 02.14 Roll Your Own Project Template with the Export Template Wizard, 03.29 Export Your Window Layouts–03.31 Using External Tools, 03.29 Export Your Window Layouts, 03.29 Export Your Window Layouts, 03.31 Using External Tools, 03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro, 03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro, 03.33 Exporting Your Command Window Aliases and External Tools List, 03.33 Exporting Your Command Window Aliases and External Tools List, 03.34 Creating and Using a Macro, 03.34 Creating and Using a Macro, 03.34 Creating and Using a Macro, 07.26 Import and Export Breakpoints

breakpoints, 07.26 Import and Export Breakpoints

Command Window aliases, 03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro, 03.34 Creating and Using a Macro

environment settings, 01.03 Exporting Your Environment Settings–01.04 Remove Projects from the Recent Projects List, 01.03 Exporting Your Environment Settings, 01.04 Remove Projects from the Recent Projects List

Export Selected Environment Settings, 03.29 Export Your Window Layouts, 03.33 Exporting Your Command Window Aliases and External Tools List

Export Template Wizard, 02.13 Create Your Own Item Template, 02.13 Create Your Own Item Template, 02.14 Roll Your Own Project Template with the Export Template Wizard, 02.14 Roll Your Own Project Template with the Export Template Wizard

[with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

external tools list, [03.33 Exporting Your Command Window Aliases and External Tools List](#)–[03.34 Creating and Using a Macro](#), [03.33 Exporting Your Command Window Aliases and External Tools List](#), [03.34 Creating and Using a Macro](#), [03.34 Creating and Using a Macro](#)

Import and Export Settings Wizard, [01.10 Reset All Your Development Settings](#)

Window Layouts, [03.29 Export Your Window Layouts](#)–[03.31 Using External Tools](#), [03.29 Export Your Window Layouts](#), [03.31 Using External Tools](#)

“Export Your Window Layouts”, [01.15 The ResetSettings Switch](#)

“Exporting Your Environment Settings”, [01.10 Reset All Your Development Settings](#)

expressions, [Use](#), [05.16 Replace in Files: Tagged Expressions](#), [Example](#), [Example](#), [07.31 The Watch Window: Watching and Changing Values](#)–[What Does It Do?](#), [Type it in](#), [Add Watch](#), [07.32 Understanding QuickWatch](#), [What Does It Do?](#)

Expression Builder, [Use](#), [05.16 Replace in Files: Tagged Expressions](#)

tagged (Replace in Files), [Example](#), [Example](#)

watch, [07.31 The Watch Window: Watching and Changing Values](#)–[What Does It Do?](#), [Type it in](#), [Add Watch](#), [07.32 Understanding QuickWatch](#), [What Does It Do?](#)

extensions, [04.13 Using Custom File Extension Associations](#)–[04.13 Using Custom File Extension Associations](#), [04.13 Using Custom File Extension Associations](#)

file, [04.13 Using Custom File Extension Associations](#)–[04.13 Using Custom File Extension Associations](#), [04.13 Using Custom File Extension Associations](#)

## Associations

external tools, [01.08 Importing or Changing Your Environment Settings, 03.31 Using External Tools–Prompt For Arguments, Prompt For Arguments, 03.32 Create Keyboard Accelerators for External Tools, 03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro, 03.33 Exporting Your Command Window Aliases and External Tools List, 03.34 Creating and Using a Macro](#)

creating keyboard accelerators for, [03.32 Create Keyboard Accelerators for External Tools](#)

exporting list, [03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro, 03.33 Exporting Your Command Window Aliases and External Tools List, 03.34 Creating and Using a Macro](#)

External Tools List setting, [01.08 Importing or Changing Your Environment Settings](#)

running, [03.31 Using External Tools–Prompt For Arguments, Prompt For Arguments](#)

# F

files, [Design View](#), [Filename–03.29 Export Your Window Layouts](#), [03.29 Export Your Window Layouts](#), [Types of Files](#)–[Image Library Contents](#), [Image Library Contents](#), [Image Library Contents](#), [04.02 Recent Files](#)–[04.03 Working with Documents on Multiple Monitors](#), [04.02 Recent Files](#), [04.03 Working with Documents on Multiple Monitors](#), [04.06 Open a File Location from the File Tab](#), [04.07 Open the File Menu Drop-Down List from Your Keyboard](#), [04.11 Understanding the File Open Location](#)–[04.12 Show Previous Versions](#), [04.11 Understanding the File Open Location](#), [04.12 Show Previous Versions](#), [Replace in Files \(Ctrl+Shift+H\)](#), [Replace in Files \(Ctrl+Shift+H\)](#), [05.10 Find In Files: Result Options](#)–[Display File Names Only](#), [Display File Names Only](#), [Keep Modified Files Open After Replace All](#), [05.11 Replace In Files: Basic Options](#), [05.16 Replace in Files: Tagged Expressions](#), [08.05 Sync the Solution Explorer to the Current File](#)–[To Use](#), [To Use](#)

current, syncing Solution Explorer to, [08.05 Sync the Solution Explorer to the Current File](#)–[To Use](#), [To Use](#)

File menu, [04.02 Recent Files](#)

file open location, [04.11 Understanding the File Open Location](#)–[04.12 Show Previous Versions](#), [04.11 Understanding the File Open Location](#), [04.12 Show Previous Versions](#)

File View (window layouts), [Design View](#)

filename argument (logging), [Filename–03.29 Export Your Window Layouts](#), [03.29 Export Your Window Layouts](#)

Find In Files, [05.10 Find In Files: Result Options](#)–[Display File Names Only](#), [Display File Names Only](#), [Keep Modified Files Open After Replace All](#)

opening file menu drop-down list from keyboard, [04.07 Open the File Menu Drop-Down List from Your Keyboard](#)

opening location from file tab, [04.06 Open a File Location from the File Tab](#)

recent, [04.02 Recent Files](#)–[04.03 Working with Documents on Multiple Monitors](#), [04.03 Working with Documents on Multiple Monitors](#)

Replace in Files, [Replace in Files \(Ctrl+Shift+H\)](#), [Replace in Files \(Ctrl+Shift+H\)](#), [05.11 Replace In Files: Basic Options](#), [05.16 Replace in Files: Tagged Expressions](#)

Visual Studio Image Library, [Types of Files](#)–[Image Library Contents](#), [Image Library Contents](#), [Image Library Contents](#)

filters, setting for breakpoints, [07.24 Setting a Breakpoint Filter](#)–[07.25 Setting a Tracepoint in Source Code](#), [07.25 Setting a Tracepoint in Source Code](#), [07.25 Setting a Tracepoint in Source Code](#)

finding, [Basic Use](#), [Arguments and Switches](#), [03.22 Testing a Command](#), [Shortcuts](#), [Find Combo Box](#), [03.24 Find Keyboard Shortcuts](#)–[03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.24 Find Keyboard Shortcuts](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [05.01 Repeat Your Last Search](#), [Find Next](#), [05.03 Using a Simple Quick Replace](#), [05.03 Using a Simple Quick Replace](#), [05.06 Using the Find Combo Box Keyboard Shortcuts](#)–[05.07 Using Incremental Search](#), [Go To Line \(Ctrl+G\)](#), [05.07 Using Incremental Search](#), [05.09 Find In Files: Find Options](#), [Find What](#)–[Navigation](#), [Find What](#), [05.10 Find In Files: Result Options](#), [05.10 Find In Files: Result Options](#), [Find Results \[1,2\] Window](#), [Navigation](#), [Keep Modified Files Open After Replace All](#), [Find Options](#), [Find Next](#), [Find What](#), [Match](#), [Find All](#), [05.17 Customize Results in Find In Files Searches](#), [05.17 Customize Results in Find In Files Searches](#)

Find All option (Find Symbol), [Find All](#)

Find And Replace dialog box, [Basic Use](#)

**Find Combo box**, [03.22 Testing a Command](#), [Find Combo Box](#), [05.01 Repeat Your Last Search](#), [05.06 Using the Find Combo Box Keyboard Shortcuts](#)–[05.07 Using Incremental Search](#), [Go To Line \(Ctrl+G\)](#), [05.07 Using Incremental Search](#)

**Find In Files**, [05.09 Find In Files: Find Options](#), [Find What](#), [05.10 Find In Files: Result Options](#), [Keep Modified Files Open After Replace All](#)

**Find In Files** search results, customizing, [05.17 Customize Results in Find In Files Searches](#)

**Find Next button** (Quick Find), [Find Next](#)

**Find Next button** (Quick Replace), [05.03 Using a Simple Quick Replace](#)

**Find Next button** (Replace in Files), [Find Next](#)

**Find options** (Find Symbol), [Match](#)

**Find options** (Replace in Files), [Find Options](#)

**Find Result Format string** (registry), [05.17 Customize Results in Find In Files Searches](#)

**Find Results windows**, [Find Results \[1,2\] Window](#)

**Find What area** (Quick Replace), [05.03 Using a Simple Quick Replace](#)

**Find What** combo box, [Find What–Navigation](#), [05.10 Find In Files: Result Options](#), [Navigation](#)

**Find What** field, [Find What](#)

**finding keyboard shortcuts** (Visual Studio), [03.24 Find Keyboard Shortcuts](#)–[03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.24 Find Keyboard Shortcuts](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)

“**Find Command**” documentation, [Arguments and Switches](#)

“**Find Keyboard Shortcuts**”, [Shortcuts](#)

floating DataTips, [07.13 Create a Floating DataTip](#)–[07.14 Adding Comments to a DataTip](#), [07.13 Create a Floating DataTip](#), [07.14 Adding Comments to a DataTip](#)

floating tool window, docking to previous location, [03.02 Dock a Floating Tool Window Back to Its Previous Location](#)

folders, [02.09 Using Solution Folders](#), [07.36 Create Folders in Class View](#)–[Creating Subfolders](#), [Creating Subfolders](#), [Creating Subfolders](#)

creating in Class View, [07.36 Create Folders in Class View](#)–[Creating Subfolders](#), [Creating Subfolders](#), [Creating Subfolders](#)

Solution Folders, [02.09 Using Solution Folders](#)

fonts, importing, [On someone's computer](#), [On someone's computer](#)

Ford, Sara, [Share Bad Code with the World](#)

Foreground And Background Colors In A Console Window snippet, [06.45 Using the Code Snippets Manager](#)

forward slash (/) to collapse Toolbox, [03.05 Expand and Collapse All in the Toolbox](#)

frameworks, [02.03 Using Older Frameworks with Multi-Targeting](#), [Framework X / Silverlight X](#)

Framework X/Silverlight X option (Look In dialog), [Framework X / Silverlight X](#)

older, using with multi-targeting, [02.03 Using Older Frameworks with Multi-Targeting](#)

Friendly Name field (browsers), [Adding New Browsers](#)

Full Screen Mode (window layouts), [Full Screen Mode](#)

## G

GhostDoc tool (VS), [08.22 Document Your Code with Three Keystrokes](#)–[08.23 Customize Visual Studio Using Windows PowerShell, To Use, To Use](#), [08.23 Customize Visual Studio Using Windows PowerShell](#)

global actions, Undo/Redo, [06.13 Undo and Redo Global Actions](#)

Go Back markers, [06.56 Navigate Backward and Navigate Forward Using Go Back Markers](#), [06.56 Navigate Backward and Navigate Forward Using Go Back Markers](#), [06.57 Select from the Current Cursor Location to the Last Go Back Marker](#), [06.57 Select from the Current Cursor Location to the Last Go Back Marker](#)

Go To Declaration (Ctrl+F12), [Go To Declaration \(Ctrl+F12\)](#)

Go To Definition (F12), [Go To Definition \(F12\)](#)

Go To Definition for CSS, [05.12 Go To Definition for Cascading Style Sheets](#)

Go To File (Find/Command box), [Go To File \(Ctrl+Shift+G\)](#)

Go To Line (Find/Command box), [Go To Line \(Ctrl+G\)](#)

Go To Reference (Shift+F12), [Go To Reference \(Shift+F12\)–Browse Definition, Browse Definition](#)

Gopp, Benjamin, [Control Zooming with a Slider Using the ZoomEditorMargin Extension](#)

Granger, Chris, [08.14 Zoom Across All Files](#)

Guide Diamond, [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#)–[03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#), [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#), [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#)

guidelines, adding to code, [08.03 Add Visual Guidelines to Your Code](#)–[08.05](#)

**Sync the Solution Explorer to the Current File, To Use, 08.05 Sync the Solution Explorer to the Current File**

# H

Harrington, Paul, [01.13 Visual Studio Logging](#), [08.03 Add Visual Guidelines to Your Code](#)

Has Changed condition (breakpoints), [07.22 Set a Simple Breakpoint Condition–Special Notes](#), [Has Changed](#), [Special Notes](#)

Help Library Manager, [01.02 Getting Table of Contents in Visual Studio 2010 Online Help](#)

Help Online area (Exception Assistant), [Help Online–Actions](#), [Actions](#)

Hide External Items option (Navigate To), [05.13 How to Use Navigate To Hide Selection \(code\)](#), [06.34 Using Hide Selection–06.35 Collapse to Definitions with Outlining](#), [06.34 Using Hide Selection](#), [06.35 Collapse to Definitions with Outlining](#)

highlighting, reference, [06.14 How to Use Reference Highlighting–06.16 Invoke Statement Completion](#), [Turning it Off](#), [06.16 Invoke Statement Completion](#)

history, command prompt, [03.16 Command Prompt History–Wildcard Search](#), [03.17 Command Prompt Tab Completion](#), [Wildcard Search](#)

Hit Count, breakpoint, [07.20 Breakpoint Hit Count–07.21 Set a Breakpoint on a Function](#), [Break When The Hit Count Is Equal To](#), [Break When The Hit Count Is Greater Than Or Equal To](#), [07.21 Set a Breakpoint on a Function](#)

Horst, Bill, [Unwind The Call Stack On Unhandled Exceptions](#)

HTML (Hypertext Markup Language), [06.26 Change the Default View in the HTML Editor–06.28 Replacing Text with a Box Selection](#), [06.27 Jump Back to the Editor from Just About Anywhere](#), [06.28 Replacing Text with a Box Selection](#), [06.43 HTML Code Snippets–06.44 JavaScript Code Snippets](#), [06.44 JavaScript Code Snippets](#), [06.49 HTML Editor Tag Navigation–06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as](#)

## Errors, 07.33 The Watch Window: Visualizers

code snippets, [06.43 HTML Code Snippets](#)–[06.44 JavaScript Code Snippets](#), [06.44 JavaScript Code Snippets](#)

editor, changing default view in, [06.26 Change the Default View in the HTML Editor](#)–[06.28 Replacing Text with a Box Selection](#), [06.27 Jump Back to the Editor from Just About Anywhere](#), [06.28 Replacing Text with a Box Selection](#)

HTML Editor tag navigation, [06.49 HTML Editor Tag Navigation](#)–[06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#)

HTML Visualizers, [07.33 The Watch Window: Visualizers](#)

|  
Icon Image field (Export Template Wizard), [02.13 Create Your Own Item Template](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

icons, refresh, [Refresh Icons–07.35 The Watch Window: Adding Watches from Variable Windows](#), [07.35 The Watch Window: Adding Watches from Variable Windows](#)

IDE navigator, [04.08 Using the IDE Navigator–Active tool windows](#), [04.08 Using the IDE Navigator, Active tool windows](#)

IIS (Internet Information Services), [02.04 Create Web Application or Virtual Directory in IIS–02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#)

illustrations, [01.01 Running Multiple Versions of Visual Studio Side-By-Side](#), [01.02 Getting Table of Contents in Visual Studio 2010 Online Help–01.03 Exporting Your Environment Settings](#), [Using Classic View](#), [Using Classic View–01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings–01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings](#), [01.04 Remove Projects from the Recent Projects List](#), [01.05 AutoRecover–01.05 AutoRecover](#), [01.05 AutoRecover](#), [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#), [01.07 Change Tool Window Animations](#), [01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#), [Seeing What You Like–Changing Your Colors](#), [On the Studio Styles site](#), [Changing Your Colors](#), [Changing Your Colors](#), [Changing Your Colors](#), [Resetting the Colors](#), [Custom Toolbars](#), [Modify Selection](#), [01.15 The ResetSettings Switch–Two Different Machines](#), [Two Different Machines](#), [02.02 Recent](#)

[Project Templates in the New Project Dialog Box](#), [02.03 Using Older Frameworks with Multi-Targeting](#), [02.04 Create Web Application or Virtual Directory in IIS](#), [02.11 Pin a Project to the Recent Projects List](#), [02.13 Create Your Own Item Template](#), [02.13 Create Your Own Item Template](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.16 Organizing Your Custom Project Templates](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)–[02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#), [03.05 Expand and Collapse All in the Toolbox](#), [03.08 Window Layouts: The Four Modes](#), [Design Mode](#), [Debug Mode](#), [Adding New Browsers](#), [Removing Browsers](#), [03.19 Understanding Commands: Simple Commands](#)–[03.20 Understanding Commands: Aliases](#), [03.19 Understanding Commands: Simple Commands](#), [03.20 Understanding Commands: Aliases](#), [03.20 Understanding Commands: Aliases](#), [Basic Use](#), [03.24 Find Keyboard Shortcuts](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#)–[03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [Reset](#)–[Example](#), [/overwrite](#)–[03.29 Export Your Window Layouts](#), [Example](#), [Example](#), [03.29 Export Your Window Layouts](#), [03.33 Exporting Your Command Window Aliases and External Tools List](#)–[03.34 Creating and Using a Macro](#), [03.34 Creating and Using a Macro](#), [03.34 Creating and Using a Macro](#), [Types of Files](#)–[Using the Images](#), [Image Library Contents](#), [Actions](#), [Objects](#), [Using the Images](#), [04.02 Recent Files](#)–[04.03 Working with Documents on Multiple Monitors](#), [04.03 Working with Documents on Multiple Monitors](#), [04.05 Close the Current Document Window](#), [04.07 Open the File Menu Drop-Down List from Your Keyboard](#), [04.08 Using the IDE Navigator](#), [04.10 Closing Just the Selected Files You Want](#)–[04.11 Understanding the File Open Location](#), [04.10 Closing Just the](#)

[Selected Files You Want](#), [04.11 Understanding the File Open Location](#), [04.11 Understanding the File Open Location](#), [05.02 Using Quick Find](#), [05.02 Using Quick Find](#), [Find Options](#)–[Regular expressions](#), [Regular expressions](#), [Buttons](#), [05.03 Using a Simple Quick Replace](#)–[05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.03 Using a Simple Quick Replace](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.07 Using Incremental Search](#)–[05.08 Search the Currently Selected String Without the Find Window](#), [05.08 Search the Currently Selected String Without the Find Window](#), [05.09 Find In Files: Find Options](#), [05.10 Find In Files: Result Options](#), [Execution](#), [Find What](#), [05.15 Find Symbol Results Shortcuts](#)–[Browse Definition](#), [Go To Reference \(Shift+F12\)](#), [Browse Definition](#), [06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)](#)–[06.16 Invoke Statement Completion](#), [Moving](#), [06.16 Invoke Statement Completion](#), [06.18 Using Parameter Information](#), [06.21 Drag and Drop Code into the Toolbox](#)–[06.21 Drag and Drop Code into the Toolbox](#), [06.22 Using Smart Tags from the Keyboard](#)–[Remove Unused Usings](#), [Remove Unused Usings](#), [06.25 Toggle Designer](#), [06.28 Replacing Text with a Box Selection](#), [06.28 Replacing Text with a Box Selection](#), [06.30 Pasting a Single Selection into a Box Selection](#), [06.30 Pasting a Single Selection into a Box Selection](#)–[06.31 Using Zero-Length Box Selection](#), [06.31 Using Zero-Length Box Selection](#), [06.31 Using Zero-Length Box Selection](#), [Click Anywhere in Area \(Menu Item\)](#), [06.37 Understanding Word Wrap](#)–[06.38 Properties Window Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window Keyboard Shortcuts](#), [VB](#), [06.41 Surround with a Code Snippet](#), [06.43 HTML Code Snippets](#), [06.49 HTML Editor Tag Navigation](#)–[06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#), [06.55 Understanding the Navigation Stack](#)–[06.55 Understanding the Navigation Stack](#), [06.55 Understanding the Navigation Stack](#), [07.03 Adding Labels to Breakpoints](#), [07.08 Code Definition Window](#), [07.13 Create a Floating DataTip](#), [07.18 Using the Call Hierarchy](#)–[07.18 Using the Call](#)

Hierarchy, 07.18 Using the Call Hierarchy, 07.19 Searching Breakpoints–07.19 Searching Breakpoints, 07.19 Searching Breakpoints, 07.20 Breakpoint Hit Count–Break When The Hit Count Is A Multiple Of, Break When The Hit Count Is A Multiple Of, Breakpoints Window, 07.23 Set a Complex Breakpoint Condition, 07.25 Setting a Tracepoint in Source Code, Continue execution, 07.27 Run to Cursor, 07.28 Using the Exception Assistant–Actions, Actions, Watch Expressions, 07.32 Understanding QuickWatch–07.33 The Watch Window: Visualizers, What Does It Do?, 07.33 The Watch Window: Visualizers–07.34 The Watch Window: Refreshing Data, 07.33 The Watch Window: Visualizers, 07.34 The Watch Window: Refreshing Data, View.ClassViewSearch Command, 07.39 The Misnamed and Misunderstood Object Browser–07.40 The Object Browser: Setting the Browsing Scope, 07.39 The Misnamed and Misunderstood Object Browser, 07.40 The Object Browser: Setting the Browsing Scope, 07.41 The Object Browser: Navigation and References, 07.42 The Exceptions Dialog Box, 07.45 Using the WPF Tree Visualizer–07.45 Using the WPF Tree Visualizer, 07.45 Using the WPF Tree Visualizer, Stack Frame, Stack Frame, 07.48 Understanding the Autos Window, Image Insertion–To Save, Image Insertion, To Save, To Install–08.04 Get More IntelliSense in Your XAML Editor, To Use, 08.04 Get More IntelliSense in Your XAML Editor, Solution Explorer Tools, AllMargins–To Uninstall, To Uninstall, 08.19 Create Regular Expressions Within Your Code, 08.21 Create and Find Code Snippets

/ResetSettings switch, 01.15 The ResetSettings Switch–Two Different Machines, Two Different Machines

4 modes of window layouts, 03.08 Window Layouts: The Four Modes

AllMargins extension, AllMargins–To Uninstall, To Uninstall

animating environment tools, 01.07 Change Tool Window Animations

AutoRecover, 01.05 AutoRecover–01.05 AutoRecover, 01.05 AutoRecover

Autos window, 07.48 Understanding the Autos Window

box selections, pasting content between, [06.30 Pasting a Single Selection into a Box Selection](#)

box selections, pasting single selection into, [06.30 Pasting a Single Selection into a Box Selection](#)–[06.31 Using Zero-Length Box Selection](#), [06.31 Using Zero-Length Box Selection](#), [06.31 Using Zero-Length Box Selection](#)

box selections, replacing text with, [06.28 Replacing Text with a Box Selection](#)

breakpoint Hit Count, [07.20 Breakpoint Hit Count](#)–[Break When The Hit Count Is A Multiple Of](#), [Break When The Hit Count Is A Multiple Of](#)

breakpoint windows, opening, [07.03 Adding Labels to Breakpoints](#)

breakpoints, setting on functions, [Breakpoints Window](#)

browsers for web development, [Adding New Browsers](#), [Removing Browsers](#)

Call Hierarchy, [07.18 Using the Call Hierarchy](#)–[07.18 Using the Call Hierarchy](#), [07.18 Using the Call Hierarchy](#)

Class View, searching in, [View.ClassViewSearch Command](#)

Classic view (VS 2010 Online Help), [Using Classic View](#)–[01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings](#)

Code Definition Window, [07.08 Code Definition Window](#)

Code Snippets, inserting, [VB](#)

code snippets, surrounding existing code with, [06.41 Surround with a Code Snippet](#)

Command Window, [03.20 Understanding Commands: Aliases](#)

Commands tab, customizing (toolbars), [Modify Selection](#)

conditional breakpoints, setting complex, [07.23 Set a Complex Breakpoint](#)

## Condition

Debug Mode (windows layouts), [Debug Mode](#)

default view in HTML editor, changing, [06.28 Replacing Text with a Box Selection](#)

Design and Source views in web projects, [06.25 Toggle Designer](#)

Design Mode, [Design Mode](#)

document window, closing current, [04.05 Close the Current Document Window](#)

Editor Guidelines extension (VS), [To Install–08.04 Get More IntelliSense in Your XAML Editor](#), [To Use, 08.04 Get More IntelliSense in Your XAML Editor](#)

Exception Assistant, [07.28 Using the Exception Assistant–Actions, Actions](#)

Exceptions dialog box, [07.42 The Exceptions Dialog Box](#)

Export Template Wizard, [02.13 Create Your Own Item Template, 02.14 Roll Your Own Project Template with the Export Template Wizard](#)

exporting Command Window Aliases/External Tools List, [03.33 Exporting Your Command Window Aliases and External Tools List–03.34 Creating and Using a Macro](#), [03.34 Creating and Using a Macro, 03.34 Creating and Using a Macro](#)

exporting environment settings, [01.03 Exporting Your Environment Settings, 01.03 Exporting Your Environment Settings](#)

file menu drop-down list, opening from keyboard, [04.07 Open the File Menu Drop-Down List from Your Keyboard](#)

file open location, [04.11 Understanding the File Open Location](#)

files, closing only specified, [04.10 Closing Just the Selected Files You Want–04.11 Understanding the File Open Location, 04.10 Closing Just the](#)

[Selected Files You Want](#), [04.11 Understanding the File Open Location files, recent](#), [04.02 Recent Files–04.03 Working with Documents on Multiple Monitors](#), [04.03 Working with Documents on Multiple Monitors](#)

Find And Replace dialog box, [Basic Use](#)

Find in Files, [05.09 Find In Files: Find Options](#), [05.10 Find In Files: Result Options](#)

Find Symbol, [Find What](#)

Find Symbols results shortcuts, [05.15 Find Symbol Results Shortcuts](#)–[Browse Definition](#), [Go To Reference \(Shift+F12\)](#), [Browse Definition](#)

finding keyboard shortcuts, [03.24 Find Keyboard Shortcuts](#)

floating DataTips, [07.13 Create a Floating DataTip](#)

Guide Diamond, [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#)

Help Library Manager, [Using Classic View](#)

HTML code snippets, [06.43 HTML Code Snippets](#)

HTML Editor tag navigation, [06.49 HTML Editor Tag Navigation–06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#)

IDE navigator, [04.08 Using the IDE Navigator](#)

Image Insertion tool (Visual Studio), [Image Insertion–To Save](#), [Image Insertion, To Save](#)

Import And Export Settings Wizard, [01.03 Exporting Your Environment Settings–01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings](#)

importing colors, [Changing Your Colors, Resetting the Colors](#)

importing/changing environment settings, [01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#)

incremental search, [05.07 Using Incremental Search](#)–[05.08 Search the Currently Selected String Without the Find Window](#), [05.08 Search the Currently Selected String Without the Find Window](#)

item templates, [02.13 Create Your Own Item Template](#)

Keyboard Mapping Schemes, [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)

keyboard shortcuts, resetting, [Reset](#)–[Example](#), [Example](#)

Local Internet Information Server dialog, [02.04 Create Web Application or Virtual Directory in IIS](#)

Locals window, changing context in, [Stack Frame](#), [Stack Frame](#)

logging commands, [/overwrite](#)–[03.29 Export Your Window Layouts](#), [Example](#), [03.29 Export Your Window Layouts](#)

multiple versions of Visual Studio, running side-by-side, [01.01 Running Multiple Versions of Visual Studio Side-By-Side](#)

navigation stack, [06.55 Understanding the Navigation Stack](#)–[06.55 Understanding the Navigation Stack](#), [06.55 Understanding the Navigation Stack](#)

New Project dialog box, [02.02 Recent Project Templates in the New Project Dialog Box](#), [02.03 Using Older Frameworks with Multi-Targeting](#), [02.16 Organizing Your Custom Project Templates](#)

Object Browser overview, [07.39 The Misnamed and Misunderstood Object Browser](#)–[07.40 The Object Browser: Setting the Browsing Scope](#), [07.39 The Misnamed and Misunderstood Object Browser](#), [07.40 The](#)

## Object Browser: Setting the Browsing Scope

Object Browser scope settings, [07.41 The Object Browser: Navigation and References](#)

outlining, collapsing code with, [Click Anywhere in Area \(Menu Item\)](#)

parameter information, [06.18 Using Parameter Information](#)

Quick Find, [05.02 Using Quick Find](#), [05.02 Using Quick Find](#), [Find Options–Regular expressions](#), [Regular expressions](#), [Buttons](#)

Quick Replace, [05.03 Using a Simple Quick Replace](#)–[05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.03 Using a Simple Quick Replace](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#)

Quickwatch, [07.32 Understanding QuickWatch](#)–[07.33 The Watch Window: Visualizers](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#)

Recent Projects list, [02.11 Pin a Project to the Recent Projects List](#)

Recent Projects list, removing projects from, [01.04 Remove Projects from the Recent Projects List](#)

Regex Editor, [08.19 Create Regular Expressions Within Your Code](#)

Replace in Files basic options, [Execution](#)

Run To Cursor, [07.27 Run to Cursor](#)

searching breakpoints, [07.19 Searching Breakpoints](#)–[07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#)

selecting/moving between matching braces, [06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)](#)–[06.16 Invoke Statement Completion](#), [Moving](#), [06.16 Invoke Statement Completion](#)

shortcuts, creating new, [03.26 Keyboard Shortcuts: Creating New](#)

[Shortcuts](#)–[03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#)

simple commands, [03.19 Understanding Commands: Simple Commands](#)–[03.20 Understanding Commands: Aliases](#), [03.19 Understanding Commands: Simple Commands](#), [03.20 Understanding Commands: Aliases](#)

smart tags, using from keyboards, [06.22 Using Smart Tags from the Keyboard](#)–[Remove Unused Usings](#), [Remove Unused Usings](#)

Snippet Designer, [08.21 Create and Find Code Snippets](#)

Solution Explorer Tools extension, [Solution Explorer Tools](#)

templates in New Project/Item dialogs, changing, [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)–[02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)

Toolbars tab, customizing, [Custom Toolbars](#)

Toolbox, [03.05 Expand and Collapse All in the Toolbox](#)

Toolbox, drag and drop code into, [06.21 Drag and Drop Code into the Toolbox](#)–[06.21 Drag and Drop Code into the Toolbox](#), [06.21 Drag and Drop Code into the Toolbox](#)

tracepoints, setting in source code, [07.25 Setting a Tracepoint in Source Code](#), [Continue execution](#)

Visual Studio 2010 Online Help, table of contents in, [01.02 Getting Table of Contents in Visual Studio 2010 Online Help](#)–[01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings](#)

Visual Studio 2010, changing visual experience in, [01.06 Improving](#)

## **Performance by Changing the Visual Experience in Visual Studio 2010**

Visual Studio color schemes, changing, [Seeing What You Like–Changing Your Colors](#), [On the Studio Styles site](#), [Changing Your Colors](#), [Changing Your Colors](#)

Visual Studio Image Library, [Types of Files–Using the Images](#), [Image Library Contents](#), [Actions](#), [Objects](#), [Using the Images](#)

visualizers, [07.33 The Watch Window: Visualizers](#)–[07.34 The Watch Window: Refreshing Data](#), [07.34 The Watch Window: Refreshing Data](#)

watch expressions, [Watch Expressions](#)

word wrap, [06.37 Understanding Word Wrap](#)–[06.38 Properties Window Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window Keyboard Shortcuts](#)

WPF Tree Visualizer, [07.45 Using the WPF Tree Visualizer](#)–[07.45 Using the WPF Tree Visualizer](#), [07.45 Using the WPF Tree Visualizer](#)

images, [08.02 Insert Images into Your Code](#)–[To Save](#), [To Save](#)

Image Insertion tool, [08.02 Insert Images into Your Code](#)–[To Save](#), [To Save](#)

Immediate Window, [Immediate Window](#)

importing, [01.03 Exporting Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#), [01.09 Change Your Visual Studio Color Scheme](#), [On someone's computer](#), [Changing Your Colors](#), [Resetting the Colors](#), [03.29 Export Your Window Layouts](#), [07.26 Import and Export Breakpoints](#)

breakpoints, [07.26 Import and Export Breakpoints](#)

colors, [01.09 Change Your Visual Studio Color Scheme](#), [On someone's computer](#), [Resetting the Colors](#)

Import and Export settings, [01.08 Importing or Changing Your Environment Settings](#)

Import and Export Settings Wizard, [01.03 Exporting Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#)

Import Selected Environment Settings option, [Changing Your Colors](#) importing or changing environment settings, [01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#), [03.29 Export Your Window Layouts](#)

Include Sub-Folders option (Find What combo box), [Include sub-folders](#)

includes operations, [05.13 How to Use Navigate To](#)

incremental search, [05.07 Using Incremental Search](#)–[05.08 Search the Currently Selected String Without the Find Window](#), [05.07 Using Incremental Search](#), [05.08 Search the Currently Selected String Without the Find Window](#)

Initial Directory, [03.31 Using External Tools](#)

Insert Documents To The Right Of Existing Tabs option, [04.01 Insert Documents to the Right of Existing Tabs](#)

installing, [01.01 Running Multiple Versions of Visual Studio Side-By-Side](#), [Introducing Visual Studio Extensions](#)–[Installing Through Xcopy](#), [Installing from the Extension Manager](#), [Installing from the Visual Studio Gallery](#), [Installing Through Xcopy](#), [Win7 Taskbar Extension](#)–[Triple Click](#), [Triple Click](#)

Visual Studio extensions, [Introducing Visual Studio Extensions](#)–[Installing Through Xcopy](#), [Installing from the Extension Manager](#), [Installing from the Visual Studio Gallery](#), [Installing Through Xcopy](#)

Win7 Taskbar Extension, [Win7 Taskbar Extension](#)–[Triple Click](#), [Triple Click](#)

Click

“Installing Visual Studio Versions Side-by-Side”, [01.01 Running Multiple Versions of Visual Studio Side-By-Side](#)

Intellisense, [Breakpoints Window](#)

Use IntelliSense To Verify The Function Name, [Breakpoints Window](#)

invoke statement completion, [06.16 Invoke Statement Completion–06.17 Move Between the Common Tab and All Tab in Statement Completion \(VB\)](#), [06.16 Invoke Statement Completion](#), [06.17 Move Between the Common Tab and All Tab in Statement Completion \(VB\)](#)

Is True condition (breakpoints), [07.22 Set a Simple Breakpoint Condition–Special Notes](#), [Special Notes](#)

item templates, [02.15 Organizing Your Custom Item Templates–02.16 Organizing Your Custom Project Templates](#), [02.15 Organizing Your Custom Item Templates](#), [02.16 Organizing Your Custom Project Templates–02.17 Reorganize the Default Item Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.17 Reorganize the Default Item Templates–02.18 Reorganize the Default Project Templates](#), [02.17 Reorganize the Default Item Templates](#), [02.18 Reorganize the Default Project Templates](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes–02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)

custom, organizing, [02.15 Organizing Your Custom Item Templates–02.16 Organizing Your Custom Project Templates](#), [02.15 Organizing Your Custom Item Templates](#), [02.16 Organizing Your Custom Project Templates–02.17 Reorganize the Default Item Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.17 Reorganize the Default Item Templates](#)

default, reorganizing, [02.17 Reorganize the Default Item](#)

Templates–02.18 Reorganize the Default Project Templates, 02.18

Reorganize the Default Project Templates

in Items dialog, changing, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes–02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes

items, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes–02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, Adding Items–Renaming Tabs, Renaming Tabs

adding to custom tabs (Toolbox), Adding Items–Renaming Tabs, Renaming Tabs

New Items dialog, changing templates in, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes–02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes

## J

JavaScript code snippets, [06.44 JavaScript Code Snippets](#)–[06.45 Using the Code Snippets Manager](#), [06.45 Using the Code Snippets Manager](#), [06.45 Using the Code Snippets Manager](#)

Johnson, Matthew, [08.01 Create Themes Using All Visual Studio Elements](#)

JScript, [06.53 Using JScript Libraries in Other JScript Files](#)

libraries, using in JScript files, [06.53 Using JScript Libraries in Other JScript Files](#)

# K

Keep Modified Files Open After Replace All option, [Replace in Files \(Ctrl+Shift+H\)](#), [Keep Modified Files Open After Replace All](#), [Result Options](#) keyboard accelerators, [03.32 Create Keyboard Accelerators for External Tools](#)–[03.32 Create Keyboard Accelerators for External Tools](#), [03.32 Create Keyboard Accelerators for External Tools](#), [03.32 Create Keyboard Accelerators for External Tools](#)

keyboard shortcuts, [02.04 Create Web Application or Virtual Directory in IIS](#)–[02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#), [02.09 Using Solution Folders](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [03.14 Moving Tool Windows Around with Your Keyboard](#), [03.14 Moving Tool Windows Around with Your Keyboard](#), [03.15 Keyboard Access to a Tool Window's Toolbar](#), [03.19 Understanding Commands: Simple Commands](#), [03.23 Understanding Commands: Running Commands](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)–[03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [Reset](#), [03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#), [03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#)–[Example](#), [/overwrite](#), [Example](#), [04.05 Close the Current Document Window](#), [04.13 Using Custom File Extension Associations](#), [05.02 Using Quick Find](#), [05.06 Using the Find Combo Box Keyboard Shortcuts](#), [05.07 Using Incremental Search](#), [05.09 Find In Files: Find Options](#), [05.10 Find In Files: Result Options](#), [05.13 How to Use Navigate To](#)–[05.14 Understanding Find Symbol](#), [05.14 Understanding Find Symbol](#)–[05.15 Find Symbol Results Shortcuts](#), [05.14 Understanding Find Symbol](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#), [06.18 Using Parameter Information](#), [06.23 Organize Using Statements \(C# Only\)](#), [06.28 Replacing Text with a Box Selection](#), [06.35 Collapse to Definitions with](#)

[Outlining](#), [06.37 Understanding Word Wrap](#)–[06.38 Properties Window](#)  
[Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window](#) [Keyboard Shortcuts](#), [06.38 Properties Window](#) [Keyboard Shortcuts](#),  
[Working with Categories](#), [06.45 Using the Code Snippets Manager](#), [06.47 Format the Current Document or Selection \(Web\)](#), [06.55 Understanding the Navigation Stack](#), [06.61 Understanding Tag Specific Options](#), [07.08 Code Definition Window](#), [07.27 Run to Cursor](#), [07.31 The Watch Window: Watching and Changing Values](#), [07.32 Understanding QuickWatch](#)–[07.33 The Watch Window: Visualizers](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#), [07.42 The Exceptions Dialog Box](#), [07.48 Understanding the Autos Window](#), [Visual Studio Keyboard Shortcut Posters](#)

[Additional Keyboard Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)–[03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#)

[Autos window](#), [07.48 Understanding the Autos Window](#)

[Code Definition window](#), [07.08 Code Definition Window](#)

[Code Snippets Manager](#), [06.45 Using the Code Snippets Manager](#)

[Collapse To Definitions with outlining](#), [06.35 Collapse to Definitions with Outlining](#)

[commands](#), [running](#), [03.23 Understanding Commands: Running Commands](#)  
[current document window](#), [closing](#), [04.05 Close the Current Document Window](#)

[Dock menu](#), [03.14 Moving Tool Windows Around with Your Keyboard](#)

[Exceptions dialog box](#), [07.42 The Exceptions Dialog Box](#)

[Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

file extension associations, custom, [04.13 Using Custom File Extension Associations](#)

Find Combo box, [05.06 Using the Find Combo Box Keyboard Shortcuts](#)

Find In Files, [05.09 Find In Files: Find Options](#), [05.10 Find In Files: Result Options](#)

Find Symbol, [05.14 Understanding Find Symbol](#)–[05.15 Find Symbol Results Shortcuts](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#)

for using smart tags, [06.23 Organize Using Statements \(C# Only\)](#)

HTML, formatting documents/selections for, [06.47 Format the Current Document or Selection \(Web\)](#)

incremental search, [05.07 Using Incremental Search](#)

moving tool windows with, [03.14 Moving Tool Windows Around with Your Keyboard](#)

Navigate To dialog, [05.13 How to Use Navigate To](#)–[05.14 Understanding Find Symbol](#), [05.14 Understanding Find Symbol](#)

navigation stack, [06.55 Understanding the Navigation Stack](#)

parameter information, [06.18 Using Parameter Information](#)

posters, [Visual Studio Keyboard Shortcut Posters](#)

Properties window, [06.38 Properties Window Keyboard Shortcuts](#), [Working with Categories](#)

Quick Find, [05.02 Using Quick Find](#)

QuickWatch, [07.32 Understanding QuickWatch](#)–[07.33 The Watch Window: Visualizers](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#)

Run To Cursor, [07.27 Run to Cursor](#)

shortcuts, creating new, [03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#)

shortcuts, resetting all, [03.27 Keyboard Shortcuts: Reset All Your Shortcuts–Example](#), [/overwrite, Example](#)

Solution Folders, [02.09 Using Solution Folders](#)

Tag Specific options, [06.61 Understanding Tag Specific Options](#)

text, replacing with box selection, [06.28 Replacing Text with a Box Selection](#)

tool window toolbar, access to, [03.15 Keyboard Access to a Tool Window's Toolbar](#)

Watch window, [07.31 The Watch Window: Watching and Changing Values](#)

Web Applications/Virtual Directories, creating in IIS, [02.04 Create Web Application or Virtual Directory in IIS](#)–[02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#)

word wrap, [06.37 Understanding Word Wrap](#)–[06.38 Properties Window Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window Keyboard Shortcuts](#)

“Keyboard Shortcuts Reset All Your Shortcuts”, [Reset](#)

“Keyboard Shortcuts: Creating New Shortcuts”, [03.19 Understanding Commands: Simple Commands](#)

keywords, Intellisense, [06.06 Using the New IntelliSense: Keywords](#)–[06.07 Using the New IntelliSense: Pascal Case](#), [06.07 Using the New IntelliSense: Pascal Case](#), [06.07 Using the New IntelliSense: Pascal Case](#)

Kurata, Deborah, [Multiple Views](#)

## L

labels, adding to breakpoints, [07.03 Adding Labels to Breakpoints–07.03 Adding Labels to Breakpoints](#), [07.03 Adding Labels to Breakpoints](#), [07.03 Adding Labels to Breakpoints](#)

Lightweight view (VS 2010 Online Help), [Using Classic View](#)

lines, [06.03 How to Keep from Accidentally Copying a Blank Line](#), [06.09 Insert a Blank Line Above or Below the Current Line](#)

blank, accidentally copying, [06.03 How to Keep from Accidentally Copying a Blank Line](#)

blank, inserting above/below current line, [06.09 Insert a Blank Line Above or Below the Current Line](#)

loading Visual Studio, [01.13 Visual Studio Logging–01.14 Visual Studio Safe Mode](#), [01.14 Visual Studio Safe Mode](#), [01.14 Visual Studio Safe Mode](#)

Locals window, [07.35 The Watch Window: Adding Watches from Variable Windows](#), [07.47 Changing Context in the Locals Window](#), [Debug Location Toolbar](#), [Stack Frame](#), [07.48 Understanding the Autos Window](#)

Location variables (customizing search results), [Variables](#)

logging (Visual Studio), [01.13 Visual Studio Logging](#), [01.13 Visual Studio Logging](#), [03.28 Understanding Commands: Logging Commands](#), [/on /off, Example](#)

Look at These File Types option (Find What combo box), [Look in](#)

Look In area (Quick Find), [Look In–Use](#), [Use](#)

Look In dialog (Find Symbol), [Look In–Search Results](#), [Look In References](#), [Search Results](#)

Look in drop-down list (Find What combo box), [Look in](#)

# M

Macro Explorer, [03.34 Creating and Using a Macro](#)

macros, creating/using, [03.34 Creating and Using a Macro](#)–[03.34 Creating and Using a Macro](#), [03.34 Creating and Using a Macro](#), [03.34 Creating and Using a Macro](#)

Manela, Matt, [08.21 Create and Find Code Snippets](#)

Mapping Schemes, Keyboard, [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#)–[03.26 Keyboard Shortcuts: Creating New Shortcuts](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.25 Keyboard Shortcuts: Additional Mapping Schemes](#), [03.26 Keyboard Shortcuts: Creating New Shortcuts](#)

/markall or /m switch, [Switches](#)

Match Case option, [Find What](#)

Find What combo box, [Find What](#)

Match option (Find Symbol), [Match](#)

Match Whole Word option, [Find What](#)

Find What combo box, [Find What](#)

matching braces, moving/selecting between, [06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)](#)–[06.16 Invoke Statement Completion](#), [Selecting](#), [06.16 Invoke Statement Completion](#)

Members list, [06.48 Using the Navigation Bar](#)–[06.50 Format HTML on Paste](#), [06.50 Format HTML on Paste](#)

menu/command references, [02.07 Track Active Item in Solution Explorer](#), [02.09 Using Solution Folders](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [03.04 Closing Tool Windows](#), [03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#), [04.02 Recent Files](#), [04.05 Close the](#)

Current Document Window, 04.05 Close the Current Document Window,  
04.11 Understanding the File Open Location, Finding Things–05.02 Using Quick Find, 05.02 Using Quick Find, 05.02 Using Quick Find, 05.02 Using Quick Find, 05.06 Using the Find Combo Box Keyboard Shortcuts, 05.07 Using Incremental Search, 05.09 Find In Files: Find Options, 05.10 Find In Files: Result Options, 05.13 How to Use Navigate To, 05.14 Understanding Find Symbol–05.15 Find Symbol Results Shortcuts, Find What, Custom Component Set (Find Symbol), 05.15 Find Symbol Results Shortcuts, 06.18 Using Parameter Information, 06.35 Collapse to Definitions with Outlining, 06.37 Understanding Word Wrap–06.38 Properties Window Keyboard Shortcuts, 06.37 Understanding Word Wrap, 06.38 Properties Window Keyboard Shortcuts, 06.38 Properties Window Keyboard Shortcuts, 06.45 Using the Code Snippets Manager, 06.55 Understanding the Navigation Stack, 06.58 Track Changes in the Editor–06.59 Edit Read-Only Files, 06.59 Edit Read-Only Files, 06.61 Understanding Tag Specific Options, 07.08 Code Definition Window, 07.20 Breakpoint Hit Count–07.21 Set a Breakpoint on a Function, 07.21 Set a Breakpoint on a Function, 07.27 Run to Cursor, 07.31 The Watch Window: Watching and Changing Values, 07.32 Understanding QuickWatch–07.33 The Watch Window: Visualizers, What Does It Do?, 07.33 The Watch Window: Visualizers, 07.42 The Exceptions Dialog Box, 07.48 Understanding the Autos Window

Autos window, 07.48 Understanding the Autos Window

breakpoint Hit Count, 07.20 Breakpoint Hit Count–07.21 Set a Breakpoint on a Function, 07.21 Set a Breakpoint on a Function

Code Definition window, 07.08 Code Definition Window

Code Snippets Manager, 06.45 Using the Code Snippets Manager

Collapse To Definitions with outlining, 06.35 Collapse to Definitions with Outlining

document windows, closing current, 04.05 Close the Current Document Window

Editor, tracking changes in, [06.58 Track Changes in the Editor](#)–[06.59 Edit Read-Only Files](#), [06.59 Edit Read-Only Files](#)

Exceptions dialog box, [07.42 The Exceptions Dialog Box](#)

Export Template Wizard, [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

file location, opening from file tab, [04.05 Close the Current Document Window](#)

file open location, [04.11 Understanding the File Open Location](#)

Find Combo box keyboard shortcuts, [05.06 Using the Find Combo Box Keyboard Shortcuts](#)

Find In Files, [05.09 Find In Files: Find Options](#), [05.10 Find In Files: Result Options](#)

Find Symbol, [05.14 Understanding Find Symbol](#)–[05.15 Find Symbol Results Shortcuts](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#)

incremental search, [05.07 Using Incremental Search](#)

keyboard shortcuts, resetting all, [03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#)

Navigate To dialog, [05.13 How to Use Navigate To](#)

navigation stack, [06.55 Understanding the Navigation Stack](#)

parameter information, [06.18 Using Parameter Information](#)

Properties window, [06.38 Properties Window Keyboard Shortcuts](#)

Quick Find, [05.02 Using Quick Find](#)

QuickWatch, [07.32 Understanding QuickWatch](#)–[07.33 The Watch Window: Visualizers](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#)

recent files, [04.02 Recent Files](#)

Run To Cursor, [07.27 Run to Cursor](#)

Solution Explorer, tracking active items in, [02.07 Track Active Item in Solution Explorer](#)

Solution Folders, [02.09 Using Solution Folders](#)

Tag Specific options, [06.61 Understanding Tag Specific Options](#)

tool windows, closing, [03.04 Closing Tool Windows](#)

Visual Studio, repeating last search in, [Finding Things](#)–[05.02 Using Quick Find](#), [05.02 Using Quick Find](#), [05.02 Using Quick Find](#)

Watch window, [07.31 The Watch Window: Watching and Changing Values](#)

word wrap, [06.37 Understanding Word Wrap](#)–[06.38 Properties Window Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window Keyboard Shortcuts](#)

menus, rearranging, [Rearrange](#)

Microsoft namespace, [Sort Usings](#)

Microsoft WindowsClient.NET site, [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

minus sign (-) to collapse area of code, [Minus Sign](#)–[06.34 Using Hide Selection](#), [06.34 Using Hide Selection](#), [06.34 Using Hide Selection](#)

Misc category (Properties window), [Working with Categories](#)

Modify Selection area (customize dialog box), [Modify Selection](#)–[01.13 Visual Studio Logging](#), [Modify Selection](#), [Modify Selection](#), [01.13 Visual Studio Logging](#)

Modify Selection option (toolbars), [Custom Toolbars](#)

monitors, multiple documents on, [04.03 Working with Documents on Multiple Monitors](#)–[04.03 Working with Documents on Multiple Monitors](#), [04.03](#)

## Working with Documents on Multiple Monitors

mouse wheel, zooming in/out of text with, [06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel](#)–[06.02 Zoom In or Out of Text in the Editor](#), [06.02 Zoom In or Out of Text in the Editor](#), [06.02 Zoom In or Out of Text in the Editor](#)

moving, [06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)](#)–[06.16 Invoke Statement Completion](#), [Selecting](#), [06.16 Invoke Statement Completion](#), [06.17 Move Between the Common Tab and All Tab in Statement Completion \(VB\)](#)

and selecting between matching braces, [06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)](#)–[06.16 Invoke Statement Completion](#), [Selecting](#), [06.16 Invoke Statement Completion](#)

between Common tab and All tab in statement completion, [06.17 Move Between the Common Tab and All Tab in Statement Completion \(VB\)](#)

multi-targeting, using older frameworks with, [02.03 Using Older Frameworks with Multi-Targeting](#)

multicolumn sorting, [Multicolumn Sorting](#)

multiple monitors, documents on, [04.03 Working with Documents on Multiple Monitors](#)–[04.03 Working with Documents on Multiple Monitors](#), [04.03 Working with Documents on Multiple Monitors](#)

multiple Startup Projects, [02.05 Multiple Startup Projects](#)–[02.06 Change the Default New Project Location](#), [02.05 Multiple Startup Projects](#), [02.05 Multiple Startup Projects](#), [02.06 Change the Default New Project Location](#)

multiple versions of VS, running side-by-side, [01.01 Running Multiple Versions of Visual Studio Side-By-Side](#)

multiple views of same document, [04.09 Multiple Views of the Same Document](#)–[04.10 Closing Just the Selected Files You Want](#), [Multiple Views](#), [04.10 Closing Just the Selected Files You Want](#)

**My Solution option (Look In dialog), My Solution**

# N

n minutes/n days options (AutoRecover), [01.05 AutoRecover namespaces](#), System, [Sort Usings](#) navigating, [04.08 Using the IDE Navigator](#), [Find Results \[1,2\] Window](#), [05.13 How to Use Navigate To](#), [05.13 How to Use Navigate To](#), [06.14 How to Use Reference Highlighting](#), [06.49 HTML Editor Tag Navigation](#)–[06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#), [06.55 Understanding the Navigation Stack](#)

Find Results list, [Find Results \[1,2\] Window](#)

HTML Editor tag navigation, [06.49 HTML Editor Tag Navigation](#)–[06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#)

IDE navigator, [04.08 Using the IDE Navigator](#)

Navigate To dialog, [05.13 How to Use Navigate To](#), [05.13 How to Use Navigate To](#)

navigation stack, [06.55 Understanding the Navigation Stack](#)

reference highlighting and, [06.14 How to Use Reference Highlighting](#)

.NET Framework, older versions of, [02.03 Using Older Frameworks with Multi-Targeting](#)

New Item dialog box, [02.17 Reorganize the Default Item Templates](#), [02.17 Reorganize the Default Item Templates](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)

New Project location, changing default of, [02.06 Change the Default New Project Location](#)–[02.07 Track Active Item in Solution Explorer](#), [02.07 Track Active Item in Solution Explorer](#)

New Projects dialog, [02.01 Search for Project Templates in the New Project Dialog Box](#), [02.02 Recent Project Templates in the New Project Dialog Box](#), [02.12 Creating Temporary Projects](#), [02.16 Organizing Your Custom Project Templates](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)

# O

Object Browser, 03.11 Using Additional Browsers for Web Development–Browser Window Size, Browser Window Size, Browser Window Size, Custom Component Set (Object Browser), Browse Definition–05.16 Replace in Files: Tagged Expressions, 05.16 Replace in Files: Tagged Expressions, 05.16 Replace in Files: Tagged Expressions, 05.16 Replace in Files: Tagged Expressions, Browse–07.41 The Object Browser: Navigation and References, Custom Component Set, Edit Custom Component Set, Edit Custom Component Set, 07.41 The Object Browser: Navigation and References–References, 07.41 The Object Browser: Navigation and References, Navigation, References

Browse Definition command, Browse Definition–05.16 Replace in Files: Tagged Expressions, 05.16 Replace in Files: Tagged Expressions, 05.16 Replace in Files: Tagged Expressions

navigation and references, 07.41 The Object Browser: Navigation and References–References, Navigation, References

setting browsing scope, Browse–07.41 The Object Browser: Navigation and References, Custom Component Set, Edit Custom Component Set, Edit Custom Component Set, 07.41 The Object Browser: Navigation and References

using for web development, 03.11 Using Additional Browsers for Web Development–Browser Window Size, Browser Window Size, Browser Window Size

“The Object Browser: Browsing Scope”, Custom Component Set (Object Browser)

Object images (VS Image Library), Objects

/on /off argument (logging), /on /off

Online Help, table of contents in (Visual Studio 2010), 01.02 Getting Table of Contents in Visual Studio 2010 Online Help–01.03 Exporting Your

[Environment Settings](#), [Online Help](#), [Using Classic View](#), [01.03 Exporting Your Environment Settings](#)

[Open Data Protocol Visualizer extension \(VS\)](#), [08.11 Visualize OData in a Graphical View](#)–[To Use](#), [More Information](#), [To Use](#)

[Open Packaging Convention](#), [Inside a .vsix File](#)

[opening](#), [04.11 Understanding the File Open Location](#)

[Open File Using Directory Of Currently Active Document option](#), [04.11 Understanding the File Open Location](#)

[ordering](#), [column \(tool windows\)](#), [07.11 Ordering and Multicolumn Sorting in Tool Windows](#)–[07.12 Pin a DataTip to Source Code](#), [Multicolumn Sorting](#), [07.12 Pin a DataTip to Source Code](#)

[Organize Usings menu \(editor\)](#), [06.23 Organize Using Statements \(C# Only\)](#)  
[outlining](#), [06.35 Collapse to Definitions with Outlining](#)–[06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#), [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#)

[Collapse to Definitions with](#), [06.35 Collapse to Definitions with Outlining](#)–[06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#), [06.36 Cut, Copy, and Paste Collapsed Code with Outlining](#)

[output files \(Visual Studio Image Library\)](#), [03.35 Visual Studio Image Library](#)

[Output Location \(Export Template Wizard\)](#), [02.13 Create Your Own Item Template](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

[Output window option \(Initial Directory\)](#), [Use Output Window](#)

[OverviewMargin extension \(VS\)](#), [AllMargins](#)–[To Install](#), [To Use](#), [08.17 Build Projects from the Windows 7 Taskbar](#), [To Install](#)

[/overwrite argument \(logging\)](#), [/overwrite](#)

# P

page-level tracing, [07.30 Application and Page Level Tracing–Combined Tracing, Attributes, Page Level Tracing, Combined Tracing, Combined Tracing](#)

Papadimoulis, Alex, [Share Bad Code with the World](#)

parameter information, [06.18 Using Parameter Information](#)–[06.20 Word Completion, 06.20 Word Completion](#)

Parsons, Jared, [08.12 Run VIM Commands in the Editor](#)–[08.14 Zoom Across All Files, 08.14 Zoom Across All Files](#)

Pascal Case, [05.13 How to Use Navigate To, 06.07 Using the New IntelliSense: Pascal Case](#)

pasting, [06.31 Using Zero-Length Box Selection](#)

single selection into box selection, [06.31 Using Zero-Length Box Selection](#)

performance, improving by changing visual experience (VS 2010), [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)–[01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010, 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010, 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

pinning, [07.12 Pin a DataTip to Source Code](#)–[07.13 Create a Floating DataTip, 07.13 Create a Floating DataTip, To Customize the Document Tab Well User Interface](#)

DataTips to source code, [07.12 Pin a DataTip to Source Code](#)–[07.13 Create a Floating DataTip, 07.13 Create a Floating DataTip](#)

tabs, [To Customize the Document Tab Well User Interface](#)

posters of keyboard shortcuts, [Visual Studio Keyboard Shortcut Posters](#)

**Power Commands extension (VS)**, [To Use](#)–[To Use](#), [08.07 Use Emacs Commands in the Editor](#), [To Use](#)

**PowerConsole extension (VS)**, [08.10 Run Windows PowerShell Within the IDE](#)–[To Use](#), [To Use](#)

**PowerShell (Windows)**, [08.23 Customize Visual Studio Using Windows PowerShell](#)–[To Customize](#), [StudioShell](#), [To Use](#), [To Customize](#)

**Prefix option (Find Symbol)**, [Prefix](#)

**Presentation Zoom extension (VS)**, [Universal Zoom](#), [Presentation Zoom](#)

**Preview Image field (Export Template Wizard)**, [02.13 Create Your Own Item Template](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

**Process combo box (Locals window)**, [Process](#)

**Productivity Power Tools (VS)**, [08.20 Get More Productivity Tools in the IDE](#)–[08.21 Create and Find Code Snippets](#), [To Customize the Document Tab Well User Interface](#), [To Customize the Document Tab Well User Interface](#), [08.21 Create and Find Code Snippets](#)

project templates, [02.14 Roll Your Own Project Template with the Export Template Wizard](#)–[02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.16 Organizing Your Custom Project Templates](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)–[02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)

creating with Export Template Wizard, [02.14 Roll Your Own Project](#)

Template with the Export Template Wizard–02.14 Roll Your Own Project Template with the Export Template Wizard, 02.14 Roll Your Own Project Template with the Export Template Wizard, 02.14 Roll Your Own Project Template with the Export Template Wizard, 02.14 Roll Your Own Project Template with the Export Template Wizard, 02.14 Roll Your Own Project Template with the Export Template Wizard, 02.14 Roll Your Own Project Template with the Export Template Wizard

custom, organizing, 02.16 Organizing Your Custom Project Templates in New Project dialog, changing, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes–02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes

projects, 01.04 Remove Projects from the Recent Projects List, 02.12 Creating Temporary Projects–02.13 Create Your Own Item Template, 02.13 Create Your Own Item Template, 02.17 Reorganize the Default Item Templates–02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, 02.18 Reorganize the Default Project Templates, 02.18 Reorganize the Default Project Templates, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes

creating temporary, 02.12 Creating Temporary Projects–02.13 Create Your Own Item Template, 02.13 Create Your Own Item Template default templates, reorganizing, 02.17 Reorganize the Default Item Templates–02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, 02.18 Reorganize the Default Project Templates, 02.18 Reorganize the Default Project Templates, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes, 02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes

removing from Recent Projects list, 01.04 Remove Projects from the Recent Projects List

properties, [\*\*02.10 Navigating Property Tabs in the Project Properties\*\*](#), [\*\*06.38 Properties Window Keyboard Shortcuts–Working with Categories\*\*](#), [\*\*Working with Categories\*\*](#)

project, navigating property tabs in, [\*\*02.10 Navigating Property Tabs in the Project Properties\*\*](#)

Properties window keyboard shortcuts, [\*\*06.38 Properties Window Keyboard Shortcuts–Working with Categories\*\*](#), [\*\*Working with Categories\*\*](#)

Pugh, David, [\*\*08.15 View Code Blocks Using Vertical Lines\*\*](#), [\*\*08.16 Get a Bird's-Eye View of Your Code in an Editor Margin\*\*](#)

# Q

question mark (?) for wildcard searches, [Wildcard Search](#)

Quick Find, [Using the Arguments and Switches](#)–[Using the Arguments and Switches](#), [Using the Arguments and Switches](#), [Using the Arguments and Switches](#), [Find What](#), [Find Options](#)–[05.03 Using a Simple Quick Replace](#), [Find Options](#), [Regular expressions](#), [Buttons](#), [Buttons](#), [05.03 Using a Simple Quick Replace](#)

buttons, [Buttons](#)

dialog box, [Using the Arguments and Switches](#)–[Using the Arguments and Switches](#), [Using the Arguments and Switches](#), [Using the Arguments and Switches](#), [Using the Arguments and Switches](#)

Find Options area, [Find Options](#)–[05.03 Using a Simple Quick Replace](#), [Regular expressions](#), [Buttons](#), [05.03 Using a Simple Quick Replace](#)

Find What field, [Find What](#)

Look In area, [Find Options](#)

Quick Info option, [06.19 Using Quick Info](#)–[06.21 Drag and Drop Code into the Toolbox](#), [06.21 Drag and Drop Code into the Toolbox](#)

Quick Replace, [05.03 Using a Simple Quick Replace](#), [05.03 Using a Simple Quick Replace](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.05 Undo Quick Replace and Replace in Files](#), [Replace in Files \(Ctrl+Shift+H\)](#)

QuickWatch, [Type it in](#), [07.35 The Watch Window: Adding Watches from Variable Windows](#)

setting watch expressions with, [Type it in](#)

window, [07.35 The Watch Window: Adding Watches from Variable Windows](#)

quotes, inserting when typing attribute values, [\*\*06.46 Insert Quotes When Typing Attribute Values\*\*](#)

# R

read-only files, editing, [06.59 Edit Read-Only Files–Dedicated Style Sheets](#), [Edit In-Memory](#), [Dedicated Style Sheets](#)

recent files, [04.02 Recent Files–04.03 Working with Documents on Multiple Monitors](#), [04.03 Working with Documents on Multiple Monitors](#)

Recent Project templates in New Project dialog, [Good News](#)

Redo/Undo global actions, [06.13 Undo and Redo Global Actions](#)

Redo/Undo stack, [06.12 Using the Undo and Redo Stack–06.13 Undo and Redo Global Actions](#), [06.13 Undo and Redo Global Actions](#)

reference highlighting, [06.14 How to Use Reference Highlighting–06.16 Invoke Statement Completion](#), [06.16 Invoke Statement Completion](#)

references, Object Browser, [07.41 The Object Browser: Navigation and References](#)–[References](#), [References](#), [References](#), [References](#)

refreshing data (Watch window), [07.34 The Watch Window: Refreshing Data–Two threads](#), [Two threads](#), [Refreshing the data](#)

Regex Editor extension (VS), [Regex Editor–To Use](#), [To Use](#), [To Use](#)

registry, editing, [05.17 Customize Results in Find In Files Searches](#), [Change Default Message](#), [To Use](#), [To Customize](#)

Regular Expressions, [Look in](#), [08.19 Create Regular Expressions Within Your Code–To Use](#), [To Use](#), [To Use](#)

creating within code, [08.19 Create Regular Expressions Within Your Code–To Use](#), [To Use](#), [To Use](#)

Find What combo box and, [Look in](#)

Remote Desktop, improving performance over, [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

renaming code snippets, [06.21 Drag and Drop Code into the Toolbox](#)

repeating last search in VS, [05.01 Repeat Your Last Search](#)–[05.02 Using Quick Find](#), [05.02 Using Quick Find](#)

Replace All buttons (Quick Replace), [05.03 Using a Simple Quick Replace](#)

Replace buttons (Replace in Files), [Replace With](#)–[05.12 Go To Definition for Cascading Style Sheets](#), [Replace All](#), [05.12 Go To Definition for Cascading Style Sheets](#)

Replace in Files operation, [05.11 Replace In Files: Basic Options](#)–[05.12 Go To Definition for Cascading Style Sheets](#), [05.12 Go To Definition for Cascading Style Sheets](#), [05.12 Go To Definition for Cascading Style Sheets](#)

basic options, [05.11 Replace In Files: Basic Options](#)–[05.12 Go To Definition for Cascading Style Sheets](#), [05.12 Go To Definition for Cascading Style Sheets](#), [05.12 Go To Definition for Cascading Style Sheets](#)

Replace With area (Quick Replace), [05.03 Using a Simple Quick Replace](#)

Replace With field (Replace in Files), [Replace With](#), [Replace](#) resetting, [Modify Selection](#), [01.15 The ResetSettings Switch](#)–[Same Machine](#), [Same Machine](#), [Reset](#)

Reset All option (Modify Selection), [Modify Selection](#)

Reset button (tools), [Reset](#)

ResetSettings switch (Visual Studio), [01.15 The ResetSettings Switch](#)–[Same Machine](#), [Same Machine](#)

Result options, [05.10 Find In Files: Result Options](#), [Keep Modified Files Open After Replace All](#)

Find in Files, [05.10 Find In Files: Result Options](#), [Keep Modified Files Open After Replace All](#)

results shortcuts (Find Symbol), [05.15 Find Symbol Results Shortcuts](#)–[05.16 Replace in Files: Tagged Expressions](#), [Go To Reference \(Shift+F12\)](#), [Clear](#)

All, 05.16 Replace in Files: Tagged Expressions

**Richards, Noah, 08.18 Triple-Click to Select an Entire Line**

Run command (Find/Command box), 05.06 Using the Find Combo Box

Keyboard Shortcuts

**Run To Cursor, 07.27 Run to Cursor**

running commands, 03.23 Understanding Commands: Running Commands–03.24 Find Keyboard Shortcuts, Command Window, 03.24 Find Keyboard Shortcuts, 03.24 Find Keyboard Shortcuts

# S

safe mode (Visual Studio), [01.14 Visual Studio Safe Mode](#)

saving, [01.05 AutoRecover](#), [01.10 Reset All Your Development Settings](#), [To Save](#)

current settings, [01.10 Reset All Your Development Settings](#)

images in .resx files, [To Save](#)

Save AutoRecover Information Every check box, [01.05 AutoRecover](#)  
Script Editor, [04.13 Using Custom File Extension Associations](#)

ScriptFree view (VS 2010 Online Help), [Using Classic View](#)

Scollable Tab Wells, [To Customize the Document Tab Well User Interface](#)  
searching, [02.01 Search for Project Templates in the New Project Dialog Box](#)–[02.03 Using Older Frameworks with Multi-Targeting](#), [02.02 Recent Project Templates in the New Project Dialog Box](#), [02.03 Using Older Frameworks with Multi-Targeting](#), [Simple Search–Wildcard Search](#), [Wildcard Search](#), [Search up](#), [05.03 Using a Simple Quick Replace](#)–[05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.03 Using a Simple Quick Replace](#), [05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#), [05.05 Undo Quick Replace and Replace in Files–Run Command \(Ctrl+ /\)](#), [Quick Replace \(Ctrl+H\)](#), [Replace in Files \(Ctrl+Shift+H\)](#), [Replace in Files \(Ctrl+Shift+H\)](#), [Replace in Files \(Ctrl+Shift+H\)](#), [05.06 Using the Find Combo Box Keyboard Shortcuts](#), [Run Command \(Ctrl+ /\)](#), [05.07 Using Incremental Search](#), [05.09 Find In Files: Find Options](#), [05.10 Find In Files: Result Options](#), [05.10 Find In Files: Result Options](#), [Keep Modified Files Open After Replace All](#), [05.11 Replace In Files: Basic Options](#), [05.13 How to Use Navigate To](#), [05.14 Understanding Find Symbol](#)–[05.15 Find Symbol Results Shortcuts](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [Search Results](#)–[05.15 Find Symbol Results Shortcuts](#),

[Search Results](#), [05.15 Find Symbol Results Shortcuts](#), [05.15 Find Symbol Results Shortcuts](#), [Go To Declaration \(Ctrl+F12\)](#), [Go To Reference \(Shift+F12\)](#), [Clear All](#), [05.16 Replace in Files: Tagged Expressions](#), [05.17 Customize Results in Find In Files Searches](#)–[Char](#), [Char](#), [07.19 Searching Breakpoints](#)–[07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#), [07.37 Search in Class View](#)–[07.38 Synchronize Your Class View](#), [07.38 Synchronize Your Class View](#)

breakpoints, [07.19 Searching Breakpoints](#)–[07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#), [07.19 Searching Breakpoints](#)

Find Combo box keyboard shortcuts, [05.06 Using the Find Combo Box Keyboard Shortcuts](#)

Find in Files, [05.09 Find In Files: Find Options](#), [05.10 Find In Files: Result Options](#), [05.10 Find In Files: Result Options](#), [Keep Modified Files Open After Replace All](#)

Find Symbol, [05.14 Understanding Find Symbol](#)–[05.15 Find Symbol Results Shortcuts](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#)

Find Symbol results shortcuts, [Go To Declaration \(Ctrl+F12\)](#), [Go To Reference \(Shift+F12\)](#), [Clear All](#)

for project templates in New Project dialog, [02.01 Search for Project Templates in the New Project Dialog Box](#)–[02.03 Using Older Frameworks with Multi-Targeting](#), [02.02 Recent Project Templates in the New Project Dialog Box](#), [02.03 Using Older Frameworks with Multi-Targeting](#)

in Class View, [07.37 Search in Class View](#)–[07.38 Synchronize Your Class View](#), [07.38 Synchronize Your Class View](#)

incremental search, [05.07 Using Incremental Search](#)

Navigate To dialog, [05.13 How to Use Navigate To](#)

Quick Replace, [05.03 Using a Simple Quick Replace](#)–[05.04 Hide the](#)

[Quick Find and Quick Replace Tool Window After the First Match, 05.03](#)  
[Using a Simple Quick Replace, 05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match, 05.04 Hide the Quick Find and Quick Replace Tool Window After the First Match](#)

[Replace in Files, Replace in Files \(Ctrl+Shift+H\), Replace in Files \(Ctrl+Shift+H\), 05.11 Replace In Files: Basic Options, 05.16 Replace in Files: Tagged Expressions](#)

[Search Results \(Find Symbol\), Search Results–05.15 Find Symbol Results Shortcuts, Search Results, 05.15 Find Symbol Results Shortcuts](#)

[search results, customizing \(Find In Files\), 05.17 Customize Results in Find In Files Searches–Char, Char](#)

[Search Up option \(Quick Find\), Search up](#)

[simple searches for files, Simple Search–Wildcard Search, Wildcard Search](#)

[Undo Quick Replace and Replace in Files, 05.05 Undo Quick Replace and Replace in Files–Run Command \(Ctrl+/, Quick Replace \(Ctrl+H\), Replace in Files \(Ctrl+Shift+H\), Run Command \(Ctrl+/\)](#)

[selecting, Modify Selection, 06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)–06.16 Invoke Statement Completion, Moving, 06.16 Invoke Statement Completion, 06.47 Format the Current Document or Selection \(Web\)–06.48 Using the Navigation Bar, 06.48 Using the Navigation Bar](#)

[and moving between matching braces, 06.15 Moving or Selecting Between Matching Braces \(C++, C# Only\)–06.16 Invoke Statement Completion, Moving, 06.16 Invoke Statement Completion](#)

[formatting for HTML selections, 06.47 Format the Current Document or Selection \(Web\)–06.48 Using the Navigation Bar, 06.48 Using the Navigation Bar](#)

Selection Comment/Selection Uncomment:, [Modify Selection](#)

Server projects, launching, [02.05 Multiple Startup Projects](#) settings, [01.03 Exporting Your Environment Settings](#)–[01.04 Remove Projects from the Recent Projects List](#), [01.04 Remove Projects from the Recent Projects List](#), [01.07 Change Tool Window Animations](#)–[01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#), [Set a Breakpoint \(F9\)](#), [07.44 Setting a Tracepoint in the Call Stack Window](#), [07.44 Setting a Tracepoint in the Call Stack Window](#)

environment, exporting, [01.03 Exporting Your Environment Settings](#)–[01.04 Remove Projects from the Recent Projects List](#), [01.04 Remove Projects from the Recent Projects List](#)

environment, importing/changing, [01.07 Change Tool Window Animations](#)–[01.08 Importing or Changing Your Environment Settings](#), [01.08 Importing or Changing Your Environment Settings](#)

Set a Breakpoint (Find/Command box), [Set a Breakpoint \(F9\)](#)

“Setting a Breakpoint in the Call Stack Window”, [07.44 Setting a Tracepoint in the Call Stack Window](#)

“Setting a Tracepoint in Source Code”, [07.44 Setting a Tracepoint in the Call Stack Window](#)

sharing tokens, [Sharing Tokens](#)

Shifflett, Karl, [08.04 Get More IntelliSense in Your XAML Editor](#) shortcuts, [03.26 Keyboard Shortcuts: Creating New Shortcuts](#)–[03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#), [03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#)

creating new, [03.26 Keyboard Shortcuts: Creating New Shortcuts](#)–[03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#), [03.27 Keyboard Shortcuts: Reset All Your Shortcuts](#)

Show Commands Containing area, [03.19 Understanding Commands: Simple Commands](#)

Show Previous Versions option (documents), [04.12 Show Previous Versions](#)–[04.12 Show Previous Versions](#), [04.12 Show Previous Versions](#), [04.12 Show Previous Versions](#)

Sitnikov, Dmitry, [08.17 Build Projects from the Windows 7 Taskbar](#)

smart tags, using from keyboard, [06.22 Using Smart Tags from the Keyboard](#)–[Remove Unused Usings](#), [Remove Unused Usings](#)

snippets, code, [06.40 Inserting Code Snippets](#)–[06.41 Surround with a Code Snippet](#), [06.41 Surround with a Code Snippet](#), [06.42 Using Code Snippets](#)–[06.43 HTML Code Snippets](#), [06.43 HTML Code Snippets](#), [Snippet Designer](#)–[To Use](#), [More Information](#), [To Use](#)

inserting, [06.40 Inserting Code Snippets](#)–[06.41 Surround with a Code Snippet](#), [06.41 Surround with a Code Snippet](#)

Snippet Designer extension (VS), [Snippet Designer](#)–[To Use](#), [More Information](#), [To Use](#)

using, [06.42 Using Code Snippets](#)–[06.43 HTML Code Snippets](#), [06.43 HTML Code Snippets](#)

Solution Explorer, [02.05 Multiple Startup Projects](#), [08.05 Sync the Solution Explorer to the Current File](#)–[To Use](#), [Solution Explorer Tools](#), [To Use](#)

Properties button, [02.05 Multiple Startup Projects](#)

Solution Explorer Tools extension (VS), [08.05 Sync the Solution Explorer to the Current File](#)–[To Use](#), [Solution Explorer Tools](#), [To Use](#)

Solution Folders, [02.09 Using Solution Folders](#)–[02.11 Pin a Project to the Recent Projects List](#), [Removing Solution Folders](#), [02.11 Pin a Project to the Recent Projects List](#)

sorting, [Multicolumn Sorting](#)

**multicolumn sorting**, [Multicolumn Sorting](#)

Source and Design views in web projects, [06.24 Switch Between Design and Source in Web Projects](#)–[06.26 Change the Default View in the HTML Editor](#), [06.25 Toggle Designer](#), [06.26 Change the Default View in the HTML Editor](#) source files (Visual Studio Image Library), [03.35 Visual Studio Image Library](#) Spell Checker extension (VS), [08.13 Check Spelling in Your Code](#)–[Control Zooming with a Slider Using the ZoomEditorMargin Extension](#), [Control Zooming with a Slider Using the ZoomEditorMargin Extension](#)

**Split view**, [Split View](#)

stack frame (Locals window), [Stack Frame](#)

Startup Projects, multiple, [02.05 Multiple Startup Projects](#)–[02.06 Change the Default New Project Location](#), [02.05 Multiple Startup Projects](#), [02.05 Multiple Startup Projects](#), [02.06 Change the Default New Project Location](#) statements, using, [06.23 Organize Using Statements \(C# Only\)](#)–[Split View](#), [Remove Unused Usings](#), [Sort Usings](#), [Split View](#)

**Status Bar**, [03.06 Searching in the Toolbox](#)

StructureAdornment extension (VS), [08.15 View Code Blocks Using Vertical Lines](#)–[To Use](#), [To Customize](#), [AllMargins](#), [To Use](#)

StudioShell extension (VS), [08.23 Customize Visual Studio Using Windows PowerShell](#)–[To Customize](#), [StudioShell](#), [To Use](#), [To Customize](#)

style sheets, dedicated (CSS), [06.60 Choosing CSS Versions](#)

styles, embedded (HTML), [Embedded Styles](#)–[Exploring the Tag Specific Options Dialog Box](#), [Exploring the Tag Specific Options Dialog Box](#)

Substring option (Find Symbol), [Substring](#)

switches and arguments (commands), [03.21 Understanding Commands: Arguments and Switches](#)–[03.22 Testing a Command](#), [Arguments and](#)

Switches, Switches, 03.22 Testing a Command

symbols, 05.16 Replace in Files: Tagged Expressions

Find Symbols results shortcuts, 05.16 Replace in Files: Tagged Expressions

synchronizing Class View, 07.38 Synchronize Your Class View

syntax for logging, 03.28 Understanding Commands: Logging Commands

System menu, 03.14 Moving Tool Windows Around with Your Keyboard

System namespaces, Sort Usings

# T

table of contents in Visual Studio 2010 Online Help, [01.02 Getting Table of Contents in Visual Studio 2010 Online Help](#)–[01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings](#)

tabs, [06.21 Drag and Drop Code into the Toolbox](#)

organizing in Toolbox, [06.21 Drag and Drop Code into the Toolbox](#)

Tag Specific Options dialog box, [Tree view](#), [Tree view](#), [Default Settings](#), [Client HTML Tags](#)–[Closing tag](#), [ASP.NET Controls](#), [New Tag](#), [New Folder](#), [Delete](#), [Per tag formatting](#), [Closing tag](#)–[Enable outlining for tag](#), [Closing tag](#), [Line breaks](#), [Enable outlining for tag](#), [Minimum lines](#), [Tag foreground](#), [Bold](#)

[ASP.NET Controls](#), [ASP.NET Controls](#)

[Bold setting](#), [Bold](#)

[Client HTML Tags](#), [Client HTML Tags](#)–[Closing tag](#), [Delete](#), [Closing tag](#) closing tag option, [Closing tag](#)–[Enable outlining for tag](#), [Enable outlining for tag](#), [Enable outlining for tag](#)

[Default Settings](#), [Default Settings](#)

[Enable Outlining for Tag](#), [Enable outlining for tag](#)

[Indent Contents](#), [Per tag formatting](#)

[Line Breaks](#), [Line breaks](#)

[Minimum lines setting](#), [Enable outlining for tag](#)

[New Folder option](#), [New Folder](#)

[New Tag option](#), [New Tag](#)

[Outlining in Code Editor](#), [Enable outlining for tag](#)

**Per Tag Colorization, [Tag foreground](#)**

**Per Tag Formatting, [Tree view](#)**

**Tag Background/Foreground settings, [Minimum lines](#)**

**Tree view, [Tree view](#)**

tagged expressions (Replace in Files), [05.16 Replace in Files: Tagged Expressions](#)–[05.17 Customize Results in Find In Files Searches](#), [05.16 Replace in Files: Tagged Expressions](#), [Example](#), [Example](#), [05.17 Customize Results in Find In Files Searches](#)

tags, [06.49 HTML Editor Tag Navigation](#)–[06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#)

HTML Editor tag navigation, [06.49 HTML Editor Tag Navigation](#)–[06.51 Display HTML/CSS Warnings as Errors](#), [06.51 Display HTML/CSS Warnings as Errors](#)

templates, [Projects and Items](#)–[02.03 Using Older Frameworks with Multi-Targeting](#), [02.02 Recent Project Templates in the New Project Dialog Box](#), [02.03 Using Older Frameworks with Multi-Targeting](#), [02.13 Create Your Own Item Template](#)–[02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.13 Create Your Own Item Template](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#)–[02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.15 Organizing Your Custom Item Templates](#)–[02.16 Organizing Your Custom Project Templates](#), [02.15](#)

[Organizing Your Custom Item Templates](#), [02.15 Organizing Your Custom Item Templates](#), [02.16 Organizing Your Custom Project Templates](#)–[02.17 Reorganize the Default Item Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.17 Reorganize the Default Item Templates](#)–[02.18 Reorganize the Default Project Templates](#), [02.17 Reorganize the Default Item Templates](#), [02.17 Reorganize the Default Item Templates](#)–[02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.18 Reorganize the Default Project Templates](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)

custom item, organizing, 02.15 Organizing Your Custom Item  
Templates–02.16 Organizing Your Custom Project Templates, 02.15  
Organizing Your Custom Item Templates, 02.15 Organizing Your Custom  
Item Templates, 02.16 Organizing Your Custom Project Templates, 02.16  
Organizing Your Custom Project Templates

**default item, reorganizing, 02.17 Reorganize the Default Item Templates–02.18 Reorganize the Default Project Templates, 02.18 Reorganize the Default Project Templates**

**default project, reorganizing, [02.17 Reorganize the Default Item Templates](#)**–**[02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#), [02.19 Change the Templates that Appear in the New Project or Item Dialog Boxes](#)**

## [Template](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

project, creating with Export Template Wizard, [02.14 Roll Your Own Project Template with the Export Template Wizard](#)–[02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

project, organizing custom, [02.16 Organizing Your Custom Project Templates](#)–[02.17 Reorganize the Default Item Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.16 Organizing Your Custom Project Templates](#), [02.17 Reorganize the Default Item Templates](#)

project, searching for, [Projects and Items](#)–[02.03 Using Older Frameworks with Multi-Targeting](#), [02.02 Recent Project Templates in the New Project Dialog Box](#), [02.03 Using Older Frameworks with Multi-Targeting](#)

temporary projects, creating, [02.12 Creating Temporary Projects](#)–[02.13 Create Your Own Item Template](#), [02.12 Creating Temporary Projects](#), [02.13 Create Your Own Item Template](#)

testing commands, [03.22 Testing a Command](#)–[Shortcuts](#), [03.22 Testing a Command](#), [03.23 Understanding Commands: Running Commands, Shortcuts](#)

text, [06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel](#)–[06.03 How to Keep from Accidentally Copying a Blank Line](#), [06.02 Zoom In or Out of Text in the Editor, Keyboard](#), [06.03 How to Keep from Accidentally Copying a Blank Line](#), [07.33 The Watch Window: Visualizers](#)

Text Visualizers, [07.33 The Watch Window: Visualizers](#)

zoom in/out with Editor, [06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel](#)–[06.03 How to Keep from Accidentally Copying a Blank Line](#), [06.02 Zoom In or Out of Text in the Editor, Keyboard](#), [06.03 How to](#)

## Keep from Accidentally Copying a Blank Line

themes, creating with VS Color Theme Editor, [08.01 Create Themes Using All Visual Studio Elements](#)–[08.02 Insert Images into Your Code](#), [To Use](#), [To Customize](#), [08.02 Insert Images into Your Code](#)

Thread combo (Locals window), [Thread](#)

Threads window, [07.24 Setting a Breakpoint Filter](#)

Title Bar, [03.02 Dock a Floating Tool Window Back to Its Previous Location](#)

To, Quan, [Inside a .vsix File](#)

TODO comments in Task List, [07.05 TODO Comments in the Task List](#)–[07.06 Create Custom Tokens for the Task List](#), [07.06 Create Custom Tokens for the Task List](#), [07.06 Create Custom Tokens for the Task List](#)

toggling Design view, [06.25 Toggle Designer](#)

tokens, creating custom for Task Lists, [07.06 Create Custom Tokens for the Task List](#)–[07.07 Create Code Shortcuts in the Task List](#), [07.06 Create Custom Tokens for the Task List](#), [07.07 Create Code Shortcuts in the Task List](#)

Toolbox, [Adding Items](#), [Deleting Tabs](#), [03.30 Stop the Toolbox from Auto-Populating from the Solution](#), [03.30 Stop the Toolbox from Auto-Populating from the Solution](#)

context menu, [03.30 Stop the Toolbox from Auto-Populating from the Solution](#)

stopping from auto-populating, [03.30 Stop the Toolbox from Auto-Populating from the Solution](#)

tabs in, [Adding Items](#), [Deleting Tabs](#)

tools, [03.31 Using External Tools–Prompt For Arguments](#), [03.31 Using External Tools](#), [Prompt For Arguments](#), [03.33 Exporting Your Command Window Aliases and External Tools List](#)

exporting external tools list, [03.33 Exporting Your Command Window Aliases and External Tools List](#)

running external, [03.31 Using External Tools–Prompt For Arguments](#), [03.31 Using External Tools, Prompt For Arguments](#)

tools window, [03.03 Cycle Through Your Open Tool Windows](#), [03.12 Auto-Hide All Tool Windows](#)–[03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.15 Keyboard Access to a Tool Window's Toolbar](#)

keyboard access to toolbar, [03.15 Keyboard Access to a Tool Window's Toolbar](#)

open, cycling through, [03.03 Cycle Through Your Open Tool Windows](#) showing hidden with Auto Hide Channel, [03.12 Auto-Hide All Tool Windows](#)–[03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#), [03.13 Showing Hidden Tool Windows with the Auto Hide Channel](#)

tracepoints, [07.44 Setting a Tracepoint in the Call Stack Window](#)–[07.45 Using the WPF Tree Visualizer](#), [07.45 Using the WPF Tree Visualizer](#)

setting in Call Stackwindow, [07.44 Setting a Tracepoint in the Call Stack Window](#)–[07.45 Using the WPF Tree Visualizer](#), [07.45 Using the WPF Tree Visualizer](#)

Triple Click extension (VS), [08.18 Triple-Click to Select an Entire Line–To Use](#), [To Use](#)

Troubleshooting Tips area (Exception Assistant), [Troubleshooting Tips](#)

two threads icon, [Two threads](#)

type-ahead functionality, [04.07 Open the File Menu Drop-Down List from Your Keyboard](#)

type-ahead selection support in Solution Explorer, [02.08 Type-Ahead](#)

[Selection Support in Solution Explorer](#)–[02.08 Type-Ahead Selection Support in Solution Explorer](#), [02.08 Type-Ahead Selection Support in Solution Explorer](#), [02.08 Type-Ahead Selection Support in Solution Explorer](#)

# U

uncommenting/commenting code in web pages, [06.08 Comment and Uncomment in Web Pages](#)–[06.09 Insert a Blank Line Above or Below the Current Line](#), [06.09 Insert a Blank Line Above or Below the Current Line](#), [06.09 Insert a Blank Line Above or Below the Current Line](#)

Undo Quick Replace/Replace in Files, [05.05 Undo Quick Replace and Replace in Files](#)–[Run Command \(Ctrl+ /\)](#), [Quick Replace \(Ctrl+H\)](#), [Replace in Files \(Ctrl+Shift+H\)](#), [Find \(Ctrl+D\)](#), [Run Command \(Ctrl+ /\)](#)

Undo/Redo global actions, [06.13 Undo and Redo Global Actions](#)

Undo/Redo stack, [06.12 Using the Undo and Redo Stack](#)–[06.13 Undo and Redo Global Actions](#), [06.13 Undo and Redo Global Actions](#)

undocking single tool windows, [03.18 Undock and Dock a Single Tool Window in a Group](#)–[Click and drag](#), [Click and drag](#)

unhandled exceptions, unwinding call stacks on, [Unwind The Call Stack On Unhandled Exceptions](#)

Unicode, [Treat Output As Unicode](#)

universal zoom, [Universal Zoom](#)

UnresolvedMergeConflict token, [07.06 Create Custom Tokens for the Task List](#)

updating JScript Intellisense, [06.52 Updating JScript IntelliSense](#)

Use Defaults option (color schemes), [Resetting the Colors](#)

Use Hardware Graphics Acceleration If Available option, [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

Use option, [Find What](#)

Find What combo box, [Find What](#)

using statements (C#), [06.23 Organize Using Statements \(C# Only\)](#)–[Split](#)

[View](#), [Remove Unused Usings](#), [Sort Usings](#), [Split View](#)

# V

variables, [07.35 The Watch Window: Adding Watches from Variable Windows](#)—[07.36 Create Folders in Class View, Watch \[1, 2, 3, 4\] Window](#), [07.36 Create Folders in Class View](#)

Variable windows, adding watches from, [07.35 The Watch Window: Adding Watches from Variable Windows](#)—[07.36 Create Folders in Class View, Watch \[1, 2, 3, 4\] Window](#), [07.36 Create Folders in Class View](#)

Vertical Tab Well, [To Customize the Document Tab Well User Interface](#) views, [Using Classic View](#)—[01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings](#), [03.08 Window Layouts: The Four Modes](#), [Viewing Assigned Aliases](#), [06.32 View White Space](#)—[06.33 Collapsing Your Code with Outlining](#), [06.32 View White Space](#), [06.33 Collapsing Your Code with Outlining](#), [View.ClassViewSearch Command](#)—[07.38 Synchronize Your Class View](#), [07.38 Synchronize Your Class View](#), [07.38 Synchronize Your Class View](#), [AllMargins](#)

Classic (Visual Studio 2010 Online Help), [Using Classic View](#)—[01.03 Exporting Your Environment Settings](#), [01.03 Exporting Your Environment Settings](#)

[View.ClassViewSearch command](#), [View.ClassViewSearch Command](#)—[07.38 Synchronize Your Class View](#), [07.38 Synchronize Your Class View](#), [07.38 Synchronize Your Class View](#)

viewing assigned aliases (commands), [Viewing Assigned Aliases](#)

viewing white space, [06.32 View White Space](#)—[06.33 Collapsing Your Code with Outlining](#), [06.32 View White Space](#), [06.33 Collapsing Your Code with Outlining](#)

window layout, [03.08 Window Layouts: The Four Modes](#)

“View Code Blocks Using Vertical Lines”, [AllMargins](#)

Virtual Directories, creating in IIS, [02.04 Create Web Application or Virtual Directory in IIS](#)  
[02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#)

Visual Basic, [Special Note for VB Users in Visual Studio 2010](#), [VB Shows Three Statements on Either Side](#)

Autos window and, [VB Shows Three Statements on Either Side](#)

users in VS 2010, [Special Note for VB Users in Visual Studio 2010](#)

Visual Basic 6 keyboard shortcuts, [05.06 Using the Find Combo Box Keyboard Shortcuts](#), [05.07 Using Incremental Search](#), [05.10 Find In Files: Result Options](#), [06.18 Using Parameter Information](#), [06.37 Understanding Word Wrap](#)–[06.38 Properties Window Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window Keyboard Shortcuts](#), [06.55 Understanding the Navigation Stack](#), [07.27 Run to Cursor](#)

Find Combo box keyboard shortcuts, [05.06 Using the Find Combo Box Keyboard Shortcuts](#)

Find In Files (Result options), [05.10 Find In Files: Result Options](#)

incremental search, [05.07 Using Incremental Search](#)

navigation stack, [06.55 Understanding the Navigation Stack](#)

parameter information, [06.18 Using Parameter Information](#)

Run To Cursor, [07.27 Run to Cursor](#)

word wrap, [06.37 Understanding Word Wrap](#)–[06.38 Properties Window Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window Keyboard Shortcuts](#)

Visual C# 2005 keyboard shortcuts, [02.07 Track Active Item in Solution Explorer](#)–[02.08 Type-Ahead Selection Support in Solution Explorer](#), [02.08 Type-Ahead Selection Support in Solution Explorer](#), [02.08 Type-Ahead](#)

Selection Support in Solution Explorer, 05.06 Using the Find Combo Box Keyboard Shortcuts, 06.18 Using Parameter Information, 06.35 Collapse to Definitions with Outlining, 06.37 Understanding Word Wrap–06.38 Properties Window Keyboard Shortcuts, 06.37 Understanding Word Wrap, 06.38 Properties Window Keyboard Shortcuts, 06.38 Properties Window Keyboard Shortcuts, 07.08 Code Definition Window, 07.31 The Watch Window: Watching and Changing Values, 07.32 Understanding QuickWatch–07.33 The Watch Window: Visualizers, What Does It Do?, 07.33 The Watch Window: Visualizers, 07.42 The Exceptions Dialog Box, 07.48 Understanding the Autos Window

Autos window, 07.48 Understanding the Autos Window

Code Definition window, 07.08 Code Definition Window

Collapse To Definitions with outlining, 06.35 Collapse to Definitions with Outlining

Exceptions dialog box, 07.42 The Exceptions Dialog Box

Find Combo box keyboard shortcuts, 05.06 Using the Find Combo Box Keyboard Shortcuts

parameter information, 06.18 Using Parameter Information

Properties window, 06.38 Properties Window Keyboard Shortcuts

QuickWatch, 07.32 Understanding QuickWatch–07.33 The Watch Window: Visualizers, What Does It Do?, 07.33 The Watch Window: Visualizers

Solution Explorer, type-ahead selection support in, 02.07 Track Active Item in Solution Explorer–02.08 Type-Ahead Selection Support in Solution Explorer, 02.08 Type-Ahead Selection Support in Solution Explorer, 02.08 Type-Ahead Selection Support in Solution Explorer

Watch window, 07.31 The Watch Window: Watching and Changing Values

word wrap, 06.37 Understanding Word Wrap–06.38 Properties Window

## [Keyboard Shortcuts](#), [06.37 Understanding Word Wrap](#), [06.38 Properties Window Keyboard Shortcuts](#)

Visual C++ 2 keyboard shortcuts, [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#), [05.02 Using Quick Find](#), [05.06 Using the Find Combo Box Keyboard Shortcuts](#), [05.14 Understanding Find Symbol–05.15 Find Symbol Results Shortcuts](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#), [06.35 Collapse to Definitions with Outlining](#), [06.38 Properties Window Keyboard Shortcuts](#), [06.45 Using the Code Snippets Manager](#), [06.55 Understanding the Navigation Stack](#), [07.27 Run to Cursor](#)

Code Snippets Manager, [06.45 Using the Code Snippets Manager](#)

Collapse To Definitions with outlining, [06.35 Collapse to Definitions with Outlining](#)

Find Combo box keyboard shortcuts, [05.06 Using the Find Combo Box Keyboard Shortcuts](#)

Find Symbol, [05.14 Understanding Find Symbol–05.15 Find Symbol Results Shortcuts](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#)

navigation stack, [06.55 Understanding the Navigation Stack](#)

Properties window, [06.38 Properties Window Keyboard Shortcuts](#)

Quick Find, [05.02 Using Quick Find](#)

rearranging windows in Visual Studio 2010 with Guide Diamond, [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#)

Run To Cursor, [07.27 Run to Cursor](#)

Visual C++ 6 keyboard shortcuts, [05.06 Using the Find Combo Box Keyboard Shortcuts](#), [05.14 Understanding Find Symbol–05.15 Find Symbol Results Shortcuts](#), [Find What](#), [Custom Component Set \(Find Symbol\)](#), [05.15 Find Symbol Results Shortcuts](#), [06.38 Properties Window Keyboard Shortcuts](#),

## 06.55 Understanding the Navigation Stack, 07.08 Code Definition Window

**Code Definition window, 07.08 Code Definition Window**

**Find Combo box keyboard shortcuts, 05.06 Using the Find Combo Box Keyboard Shortcuts**

**Find Symbol, 05.14 Understanding Find Symbol–05.15 Find Symbol Results Shortcuts, Find What, Custom Component Set (Find Symbol), 05.15 Find Symbol Results Shortcuts**

**navigation stack, 06.55 Understanding the Navigation Stack**

**Properties window, 06.38 Properties Window Keyboard Shortcuts**

**visual experience, changing in VS 2010, 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010–01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010, 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010, 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010**

**Visual Studio, 01.01 Running Multiple Versions of Visual Studio Side-By-Side, 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010, 01.13 Visual Studio Logging–01.14 Visual Studio Safe Mode, 01.14 Visual Studio Safe Mode, 01.14 Visual Studio Safe Mode, 01.15 The ResetSettings Switch–Same Machine, Same Machine, 02.18 Reorganize the Default Project Templates, 03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond–03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond, 03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond, 03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond, 03.21 Understanding Commands: Arguments and Switches, Types of Files–Using the Images, Image Library Contents, Actions, Using the Images, Installing from the Visual Studio Gallery–Installing Through Xcopy, Installing from the Visual Studio Gallery, Installing Through Xcopy**

changing visual experience in, [01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010](#)

Guide Diamond, rearranging windows with, [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#)–[03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#), [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#), [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#), [03.01 Rearrange Windows in Visual Studio 2010 Using the Guide Diamond](#)

Image Library, [Types of Files](#)–[Using the Images](#), [Image Library Contents](#), [Actions](#), [Using the Images](#)

in safe mode, [01.14 Visual Studio Safe Mode](#)

logging, [01.13 Visual Studio Logging](#)–[01.14 Visual Studio Safe Mode](#), [01.14 Visual Studio Safe Mode](#)

project templates stored in, [02.18 Reorganize the Default Project Templates](#)

ResetSettings switch, [01.15 The ResetSettings Switch](#)–[Same Machine](#), [Same Machine](#)

running multiple versions side-by-side, [01.01 Running Multiple Versions of Visual Studio Side-By-Side](#)

Visual Studio Gallery, [Installing from the Visual Studio Gallery](#)–[Installing Through Xcopy](#), [Installing from the Visual Studio Gallery](#), [Installing Through Xcopy](#)

“Visual Studio Commands with Arguments”, [03.21 Understanding Commands: Arguments and Switches](#)

Visual Studio 6 keyboard shortcuts, [05.06 Using the Find Combo Box](#) [Keyboard Shortcuts](#), [05.07 Using Incremental Search](#), [05.09 Find In Files: Find Options](#), [06.18 Using Parameter Information](#), [06.35 Collapse to Definitions with Outlining](#), [06.38 Properties Window Keyboard Shortcuts](#)

**Collapse To Definitions with outlining, [06.35 Collapse to Definitions with Outlining](#)**

**Find Combo box keyboard shortcuts, [05.06 Using the Find Combo Box Keyboard Shortcuts](#)**

**Find In Files, [05.09 Find In Files: Find Options](#)**

**incremental search, [05.07 Using Incremental Search](#)**

**parameter information, [06.18 Using Parameter Information](#)**

**Properties window, [06.38 Properties Window Keyboard Shortcuts](#)**

Visual Studio extensions, [Introducing Visual Studio Extensions](#), [Installing from the Extension Manager](#), [Installing from the Visual Studio Gallery–Installing Through Xcopy](#), [Installing from the Visual Studio Gallery](#), [Installing Through Xcopy](#), [Installing Through Xcopy](#), [Disabling an Extension](#), [Uninstalling an Extension–To Use](#), [Resources for Developing Extensions](#), [To Use](#), [08.02 Insert Images into Your Code–To Save](#), [To Save](#), [To Use–Control Zooming with a Slider Using the ZoomEditorMargin Extension](#), [Presentation Zoom](#), [Control Zooming with a Slider Using the ZoomEditorMargin Extension](#), [Control Zooming with a Slider Using the ZoomEditorMargin Extension–Structure Adornment](#), [Control Zooming with a Slider Using the ZoomEditorMargin Extension–Structure Adornment](#), [Structure Adornment](#), [Structure Adornment](#), [AllMargins–To Install](#), [AllMargins](#), [To Use–To Install](#), [Win7 Taskbar Extension](#), [To Install](#), [To Install](#), [Triple Click–To Use](#), [To Use](#), [08.20 Get More Productivity Tools in the IDE–08.21 Create and Find Code Snippets](#), [08.21 Create and Find Code Snippets](#)

AllMargins extension, [AllMargins–To Install](#), [To Install](#)

disabling, [Disabling an Extension](#)

**Image Insertion tool, [08.02 Insert Images into Your Code–To Save](#), [To Save](#)**

installing from Extension Manager, [Installing from the Extension Manager](#)

installing from Visual Studio Gallery, [Installing from the Visual Studio Gallery](#)–[Installing Through Xcopy](#), [Installing from the Visual Studio Gallery](#), [Installing Through Xcopy](#)

installing through Xcopy, [Installing Through Xcopy](#)

overview, [Introducing Visual Studio Extensions](#)

OverviewMargin extension, [To Use](#)–[To Install](#), [To Install](#)

Presentation Zoom extension, [Control Zooming with a Slider Using the ZoomEditorMargin Extension](#)–[Structure Adornment](#), [Structure Adornment](#)

Productivity Power Tools, [08.20 Get More Productivity Tools in the IDE](#)–[08.21 Create and Find Code Snippets](#), [08.21 Create and Find Code Snippets](#)

resources for developing, [Resources for Developing Extensions](#)

Spell Checker extension, [To Use](#)–[Control Zooming with a Slider Using the ZoomEditorMargin Extension](#), [Presentation Zoom](#), [Control Zooming with a Slider Using the ZoomEditorMargin Extension](#)

StructureAdornment extension, [AllMargins](#)

Triple Click extension, [Triple Click](#)–[To Use](#), [To Use](#)

uninstalling, [Uninstalling an Extension](#)–[To Use](#), [To Use](#)

Win7 Taskbar Extension, [Win7 Taskbar Extension](#)

ZoomEditorMargin extension, [Control Zooming with a Slider Using the ZoomEditorMargin Extension](#)–[Structure Adornment](#), [Structure Adornment](#)

visualizers, [07.33 The Watch Window: Visualizers](#)–[07.34 The Watch Window: Refreshing Data](#), [07.34 The Watch Window: Refreshing Data](#)

.vsix files, [Inside a .vsix File](#)

.vssettings files, [Changing Your Colors](#), [03.29 Export Your Window Layouts](#)

VsVim extension (VS), [08.12 Run VIM Commands in the Editor](#)–[08.14 Zoom](#)

**Across All Files, Spell Checker, 08.14 Zoom Across All Files**

## W

Watch windows, [Column Ordering](#), [07.31 The Watch Window: Watching and Changing Values–What Does It Do?](#), [Changing Values](#), [07.32 Understanding QuickWatch](#), [What Does It Do?](#), [07.33 The Watch Window: Visualizers](#)–[07.34 The Watch Window: Refreshing Data](#), [07.34 The Watch Window: Refreshing Data](#)

column ordering and, [Column Ordering](#)

visualizers and, [07.33 The Watch Window: Visualizers](#)–[07.34 The Watch Window: Refreshing Data](#), [07.34 The Watch Window: Refreshing Data](#)

watching and changing values, [07.31 The Watch Window: Watching and Changing Values–What Does It Do?](#), [Changing Values](#), [07.32 Understanding QuickWatch](#), [What Does It Do?](#)

Web Applications, creating in IIS, [02.04 Create Web Application or Virtual Directory in IIS](#)–[02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#), [02.04 Create Web Application or Virtual Directory in IIS](#)

web development, different browsers for, [03.11 Using Additional Browsers for Web Development](#)–[Browser Window Size](#), [Browser Window Size](#)

web pages, commenting/uncommenting code in, [06.08 Comment and Uncomment in Web Pages](#)–[06.09 Insert a Blank Line Above or Below the Current Line](#), [06.09 Insert a Blank Line Above or Below the Current Line](#)

web projects, switching between Design and Source views in, [06.24 Switch Between Design and Source in Web Projects](#)–[06.26 Change the Default View in the HTML Editor](#), [06.25 Toggle Designer](#), [06.26 Change the Default View in the HTML Editor](#)

websites, for downloading, [Universal Zoom](#)

“Presentation Zoom” extension, [Universal Zoom](#)

websites, for further information, [01.13 Visual Studio Logging](#), [02.14 Roll Your Own Project Template with the Export Template Wizard](#), [03.19 Understanding Commands: Simple Commands](#), [Regular expressions](#), [06.38 Properties Window Keyboard Shortcuts](#), [06.49 HTML Editor Tag Navigation](#), [Attributes](#), [Combined Tracing](#), [07.48 Understanding the Autos Window](#), [Installing from the Visual Studio Gallery](#), [Inside a .vsix File](#), [Inside a .vsix File](#), [Uninstalling an Extension](#), [To Use](#), [More Information](#), [More Information](#), [08.19 Create Regular Expressions Within Your Code](#), [More Information](#)

.vsix file, [Inside a .vsix File](#)

Autos window, [07.48 Understanding the Autos Window](#)

Emacs keyboard shortcuts, [More Information](#)

Image Insertion tool, customizing, [To Use](#)

MSDN documentation, [03.19 Understanding Commands: Simple Commands](#)

Open Packaging Convention, [Inside a .vsix File](#)

Power Console extension (VS), [More Information](#)

Properties Window, [06.38 Properties Window Keyboard Shortcuts](#)

Regex Editor, [08.19 Create Regular Expressions Within Your Code](#)

Regular Expressions, [Regular expressions](#)

StudioShell module, [More Information](#)

trace element settings, [Attributes](#)

tracing, [Combined Tracing](#)

Visual Studio extensions, development resources for, [Uninstalling an Extension](#)

Visual Studio Gallery, [Installing from the Visual Studio Gallery](#)

Visual Studio, documentation for, [01.13 Visual Studio Logging](#)

“Export Template Wizard” documentation, [02.14 Roll Your Own Project Template with the Export Template Wizard](#)

“HTML Editor Tag Navigation in Visual Web Developer”, [06.49 HTML Editor Tag Navigation](#)

When Breakpoint Is Hit dialog box, [Setting Tracepoints](#)

white space, [06.09 Insert a Blank Line Above or Below the Current Line](#)

adding extra in code, [06.09 Insert a Blank Line Above or Below the Current Line](#)

Whole Word option (Find Symbol), [Whole Word](#)

/wild or /l switch, [Switches](#)

wildcard searches, [Wildcards](#), [Wildcards](#)

Win7 Taskbar Extension (VS), [08.17 Build Projects from the Windows 7 Taskbar—Triple Click](#), [To Uninstall](#), [Triple Click](#)

Window Layouts, [03.08 Window Layouts: The Four Modes](#)–[03.09 Window Layouts: Design, Debug, and Full Screen](#), [03.09 Window Layouts: Design, Debug, and Full Screen](#), [03.09 Window Layouts: Design, Debug, and Full Screen](#), [Design Mode](#), [Full Screen Mode](#)

Design Mode, [Design Mode](#)

four modes of, [03.08 Window Layouts: The Four Modes](#)–[03.09 Window Layouts: Design, Debug, and Full Screen](#), [03.09 Window Layouts: Design, Debug, and Full Screen](#), [03.09 Window Layouts: Design, Debug, and Full Screen](#), [03.09 Window Layouts: Design, Debug, and Full Screen](#)

Full Screen Mode, [Full Screen Mode](#)

windows, [03.03 Cycle Through Your Open Tool Windows](#), [Browser Window Size](#), [03.15 Keyboard Access to a Tool Window’s Toolbar](#), [Command Window](#), [Immediate Window](#), [05.08 Search the Currently Selected String Without the Find Window](#), [07.08 Code Definition Window](#), [Breakpoints Window](#), [07.35](#)

## The Watch Window: Adding Watches from Variable Windows

Breakpoints, Breakpoints Window

browser window size, Browser Window Size

Code Definition window, 07.08 Code Definition Window

Command Window, Command Window

Immediate Window, Immediate Window

moving tool windows with keyboard, 03.15 Keyboard Access to a Tool Window's Toolbar

QuickWatch window, 07.35 The Watch Window: Adding Watches from Variable Windows

searching currently selected string without Find window, 05.08 Search the Currently Selected String Without the Find Window

tool windows, cycling through open, 03.03 Cycle Through Your Open Tool Windows

Windows, Microsoft, 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010, 08.23 Customize Visual Studio Using Windows PowerShell—To Customize, To Customize

Windows PowerShell, 08.23 Customize Visual Studio Using Windows PowerShell—To Customize, To Customize

Windows Presentation Foundation (WPF), 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010

words, 06.37 Understanding Word Wrap—06.38 Properties Window Keyboard Shortcuts, 06.37 Understanding Word Wrap, 06.38 Properties Window Keyboard Shortcuts

word wrap, 06.37 Understanding Word Wrap—06.38 Properties Window Keyboard Shortcuts, 06.37 Understanding Word Wrap, 06.38 Properties Window Keyboard Shortcuts

**WPF Tree Visualizer, 07.45 Using the WPF Tree Visualizer–07.45 Using the WPF Tree Visualizer, 07.45 Using the WPF Tree Visualizer, 07.45 Using the WPF Tree Visualizer, 07.45 Using the WPF Tree Visualizer**

**WPFPerf tool (Windows SDK), 01.06 Improving Performance by Changing the Visual Experience in Visual Studio 2010**

# X

XAML Intellisense Presenter extension (VS), [08.04 Get More IntelliSense in Your XAML Editor](#)–[08.05 Sync the Solution Explorer to the Current File, To Use](#), [08.05 Sync the Solution Explorer to the Current File](#)

Xcopy, installing VS extensions through, [Installing Through Xcopy](#)

XML, exporting DataTips as, [07.17 Import and Export DataTips](#)

Xu, Jianchun, [08.10 Run Windows PowerShell Within the IDE](#)

## Z

[zero-length box selections, 06.31 Using Zero-Length Box Selection–06.32 View White Space, 06.31 Using Zero-Length Box Selection, 06.32 View White Space](#)

[zoom in/out of text with Editor, 06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel–06.03 How to Keep from Accidentally Copying a Blank Line, 06.01 Zoom In or Out of Text in the Editor Using the Mouse Wheel, Keyboard, 06.03 How to Keep from Accidentally Copying a Blank Line](#)

[ZoomEditorMargin extension \(VS\), Control Zooming with a Slider Using the ZoomEditorMargin Extension–Structure Adornment, Structure Adornment, Structure Adornment](#)

# **Appendix B. Additional Tips**

## **Additional Tips from Chapter 1**

### **AX.01 Getting Help Samples**

<b>WINDOWS</b>	Alt,H, L
<b>MENU</b>	Help   Samples
<b>COMMAND</b>	Help.Samples
<b>VERSIONS</b>	2008,2010
<b>CODE</b>	vstipEnv0002

This is another one of those things that is always there but that developers tend to forget about. You can get sample code from within Visual Studio itself. Just select Help | Samples from your menu bar.

What you see next is version-specific—for example, in Visual Studio 2010, you will see the page shown below:

Visual Studio 2010 Samples X

URL: C:\Program Files\Microsoft Visual Studio 10.0\Samples\1033\ReadMe.htm

# Microsoft Visual Studio 2010 Samples

Explore Visual Studio samples to get up-to-speed quickly with the Visual Studio 2010!

## MSDN Online Samples

The most up-to-date samples are always available online through the [Visual Studio 2010 Samples](#) page.

Be sure to check back often for the latest updates to samples, along with new samples as they are released!

## Samples on Disk

Samples for Visual Basic, Visual C#, Visual C++ and more are available in local .zip files in the location where you installed Visual Studio:

**Note:** To build the samples, you must extract the .zip files to a folder to which you have write access.

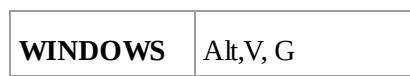
1. Select the .zip file containing the samples you want to open from the [local Samples folder](#).
2. Right-click the .zip file and then click **Extract All...** to extract the samples to a folder of your choice, such as **My Documents** or your **Desktop**.

Click the Local Samples Folder link, and you see the folder shown in the following image:

Name	Date modified
LanguageSamples	1/24/2011 9:55 PM
CSharpSamples.zip	3/19/2010 11:34 AM
ReadMe.htm	9/30/2009 9:19 PM
TeamTest Samples.zip	3/19/2010 8:05 AM
VBSamples.zip	3/19/2010 11:34 AM
VC2010Samples.zip	3/19/2010 11:34 AM

From here, you just unzip the code samples for the language you are interested in.

## AX.02 Make the Start Page Go Away



<b>MENU</b>	View   Start Page
<b>COMMAND</b>	View.StartPage
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0002

Does the Start Page appearing every time you start Visual Studio annoy you? You can set up Visual Studio so that the Start Page doesn't load when you start Visual Studio. Just look in the lower-left corner, and clear the Show Page On Startup check box.

From now on, the Start Page shows up only when you want it to.



## AX.03 Bringing Back the Start Page

<b>WINDOWS</b>	Alt,V, G
<b>MENU</b>	View   Start Page
<b>COMMAND</b>	View.StartPage
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipTool0001

You might have noticed that when you open up a Solution or Project, the Start Page goes away. This is the new default behavior in Visual Studio 2010.

To change it, look in the lower-left corner of the new and improved Start Page, and clear the Close Page After Project Load check box. From now on, the Start Page sticks around until you close it yourself.

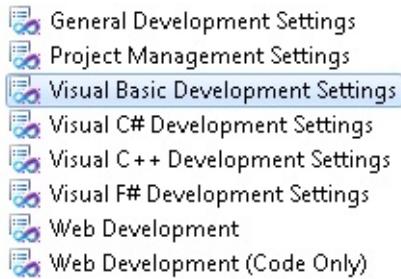


## AX.04 Show All Settings with Visual Basic

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options
<b>COMMAND</b>	Tools.Options

<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	VB
<b>CODE</b>	vstipEnv0042

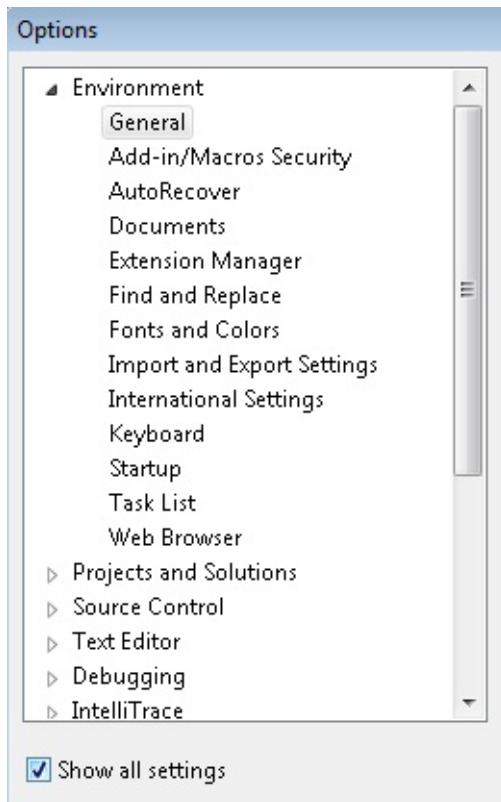
Did you choose the Visual Basic settings during your install?



If so, you might notice, when you open the Tools | Options menu, that the options are not all there:



To bring them back, just select Show All Settings at the bottom-left of the dialog box, and all the available settings will reappear:



## AX.05 Find Your Development Settings

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0020

Development settings determine quite a bit when you use Visual Studio. For example, they determine how you see the installed templates and what options you see initially in the Tools | Options dialog box. You probably remember the following choices when you installed Visual Studio:

- General Development Settings
- Project Management Settings
- Visual Basic Development Settings
- Visual C# Development Settings
- Visual C++ Development Settings
- Visual F# Development Settings
- Web Development
- Web Development (Code Only)

If you happen to forget what choice you made, you can quickly get a reminder by going to the registry key  
`HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\<version>\Profile.`

### WARNING

Don't make any changes in the Registry for this tip; just view the data.

So, for Visual Studio 2010, the path would be HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\10.0\Profile, and then you would look at the “LastResetSettingsFile” string:

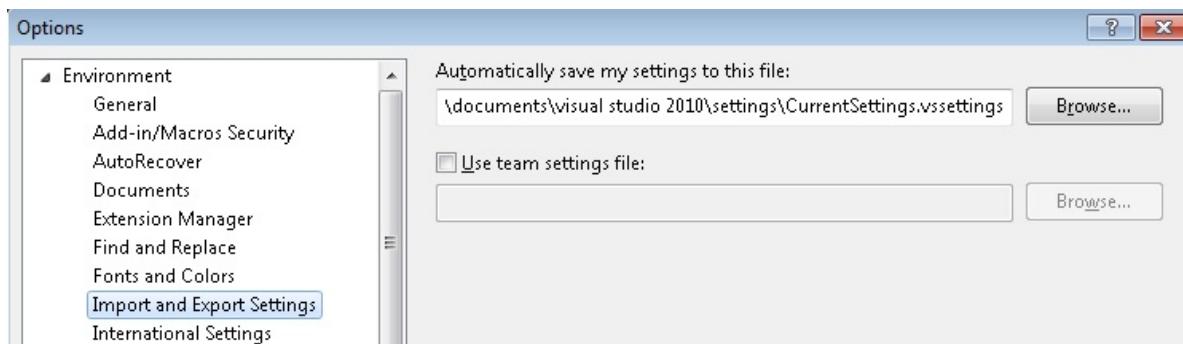
 LastResetSettingsFile REG\_SZ %vsspv\_vs\_install\_directory%\Common7\IDE\Profiles\General.vssettings

Assuming you haven't reset the settings via Tools | Import And Export Settings lately, this value should have the name of the settings file you chose at first launch. In my case, as shown in the preceding graphic, I chose the “General Development Settings” (General.vssettings) option.

## AX.06 Settings Automatically Saved On Exit

<b>WINDOWS</b>	Alt,T, I
<b>MENU</b>	Tools   Options   Import and Export Settings
<b>COMMAND</b>	Tools.ImportandExportSettings
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0026

Visual Studio automatically saves your settings every time it is closed. To see where this file is located (or to change the location), go to Tools | Options | Import And Export Settings and locate the Automatically Save My Settings To This File area:



The nice thing is that you can use this to “undo” any changes you have made during a session. So if you made some changes to Visual Studio but don't want to keep them, you can import the CurrentSettings.vssettings (default name) file to bring

back your settings from the last time Visual Studio was closed.

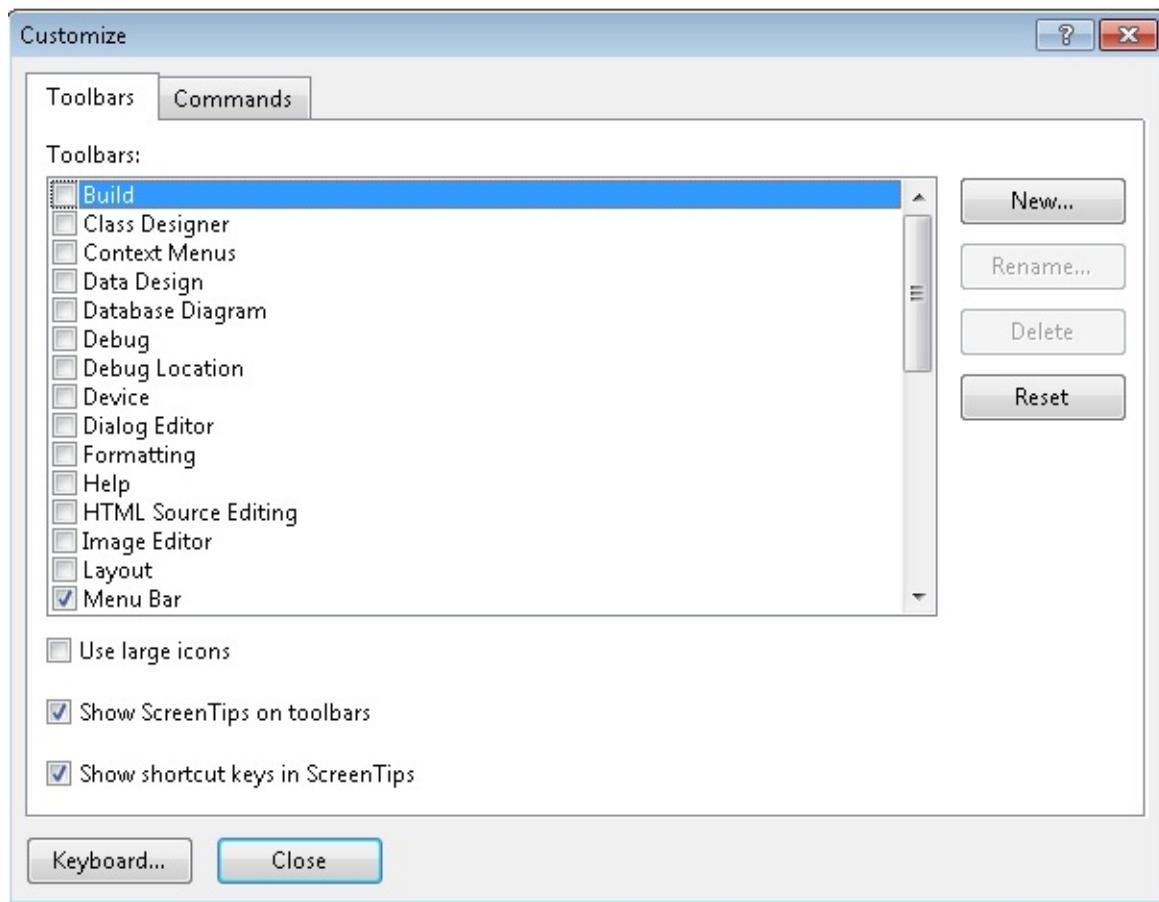
## AX.07 Customize Your Toolbars in Visual Studio 2008: Toolbars Tab

<b>WINDOWS</b>	Alt,T, C
<b>MENU</b>	Tools   Customize
<b>COMMAND</b>	Tools.Customize
<b>VERSIONS</b>	2008
<b>CODE</b>	vstipEnv0032

You can customize any toolbar in Visual Studio 2008. Just click the drop-down arrow to the right of any toolbar, click Add Or Remove Buttons, and choose Customize:

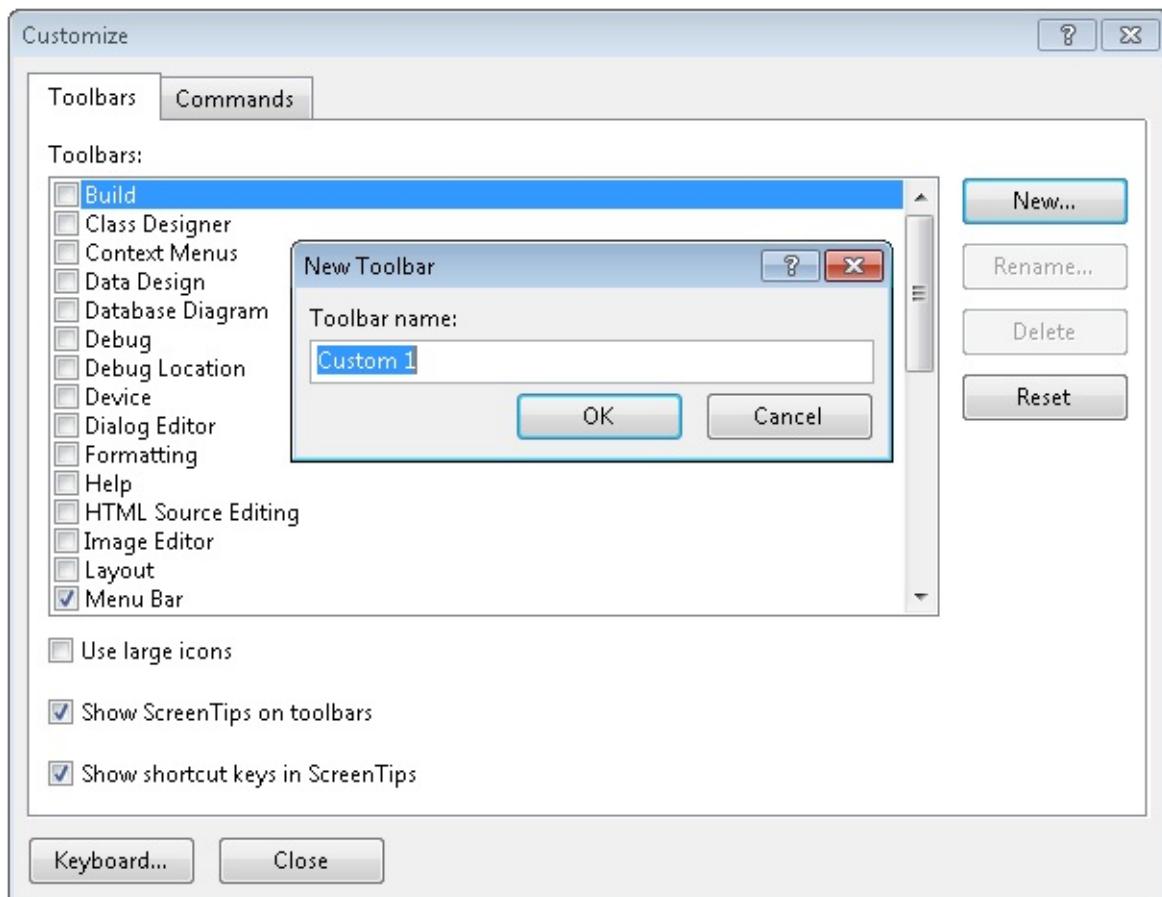


Whichever option you choose opens the Customize dialog box:

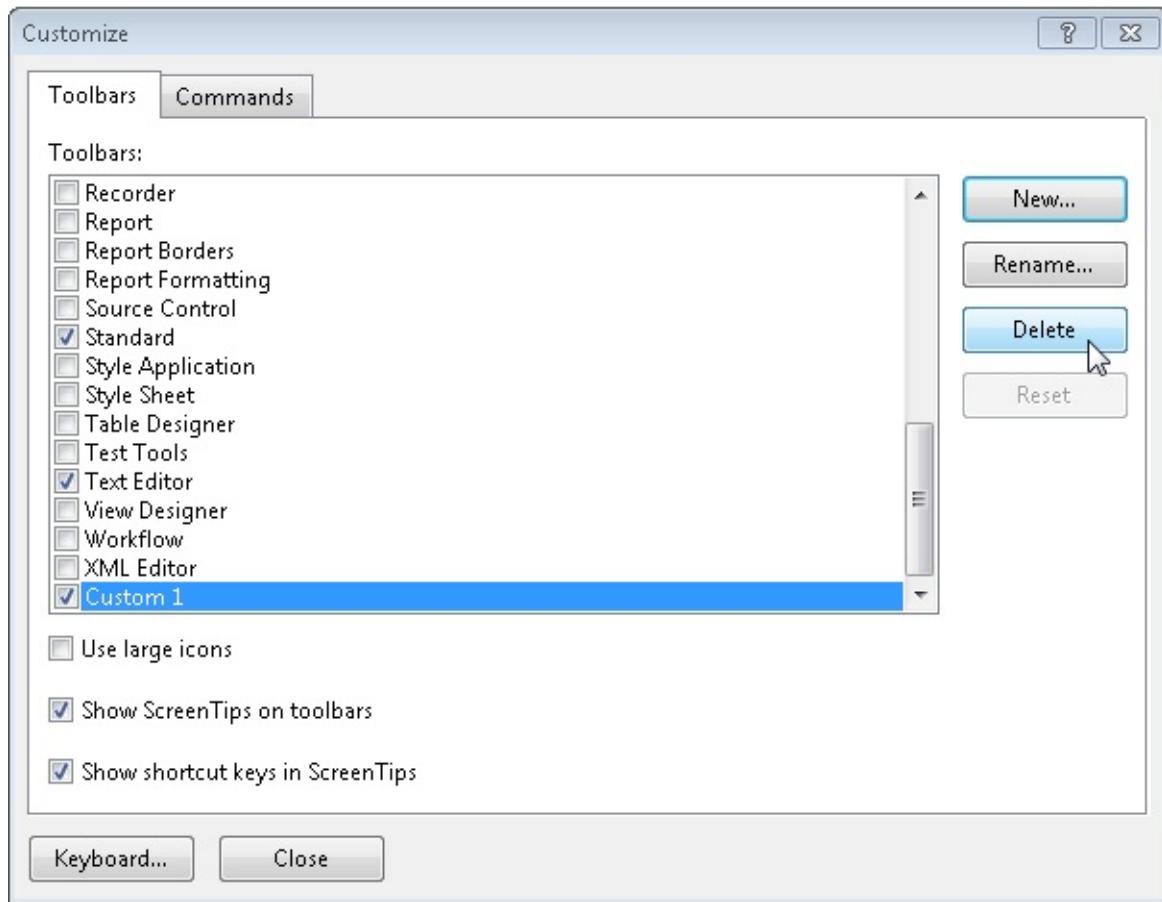


## Custom Toolbars

Notice that the Toolbars tab lists all the available toolbars. When you click New to create a customized toolbar, you are prompted to give the new toolbar a name:

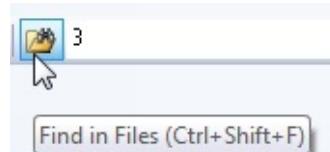


After you name it, you can delete the custom toolbar by clicking Delete, or you can rename it by clicking Rename:



Additionally, you can take advantage of the following options at the bottom of the dialog box:

- **Use Large Icons** Shows larger icons on toolbars in your environment.
- **Show ScreenTips On Toolbars** Indicates whether you see a tooltip for the toolbar items:

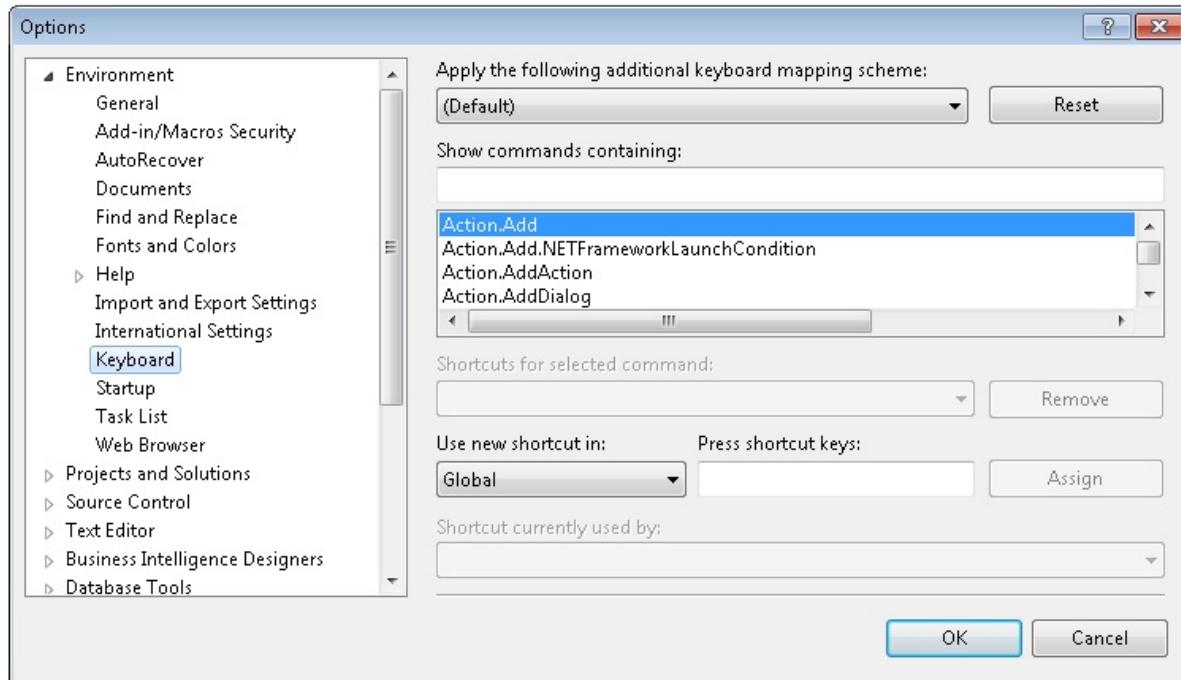


- **Show Shortcut Keys In ScreenTips** Shows the shortcut key combinations in the tooltips for items that have them.



Clicking the Keyboard button at the bottom of the Customize dialog box is just the

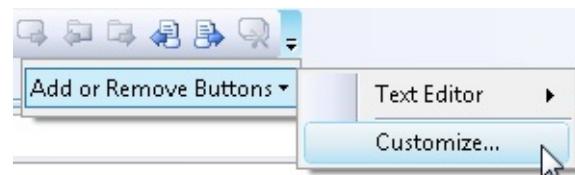
same as going to Tools | Options | Keyboard (see vstipTool0063, [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), on page 127 for details):



## AX.08 Customize Your Toolbars in Visual Studio 2008: Commands Tab

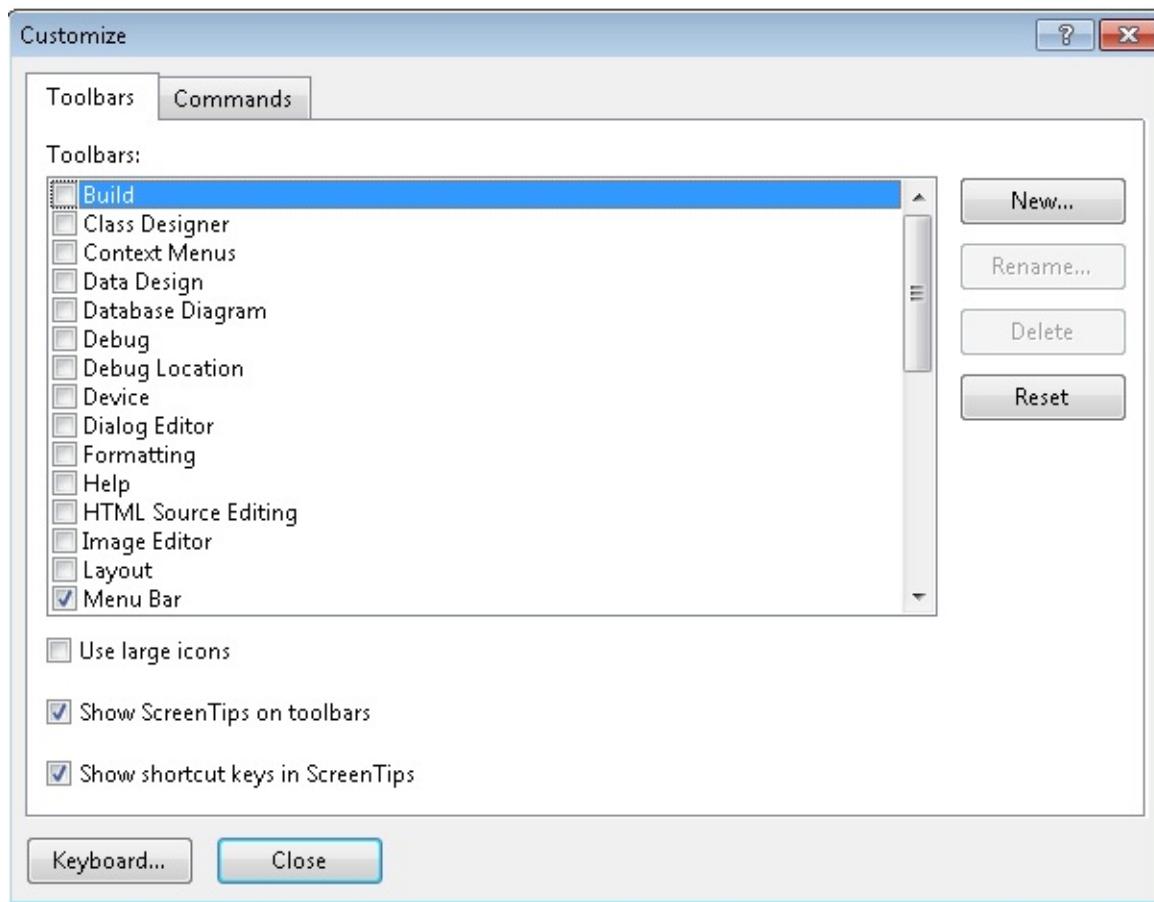
<b>WINDOWS</b>	Alt,T, C
<b>MENU</b>	Tools   Customize
<b>COMMAND</b>	Tools.Customize
<b>VERSIONS</b>	2008
<b>CODE</b>	vstipEnv0033

You can customize any toolbar in Visual Studio 2008. Just click the drop-down arrow to the right of any toolbar, click Add Or Remove Buttons, and choose Customize:



Alternatively, you can go to Tools | Customize on the menu bar. Whichever option

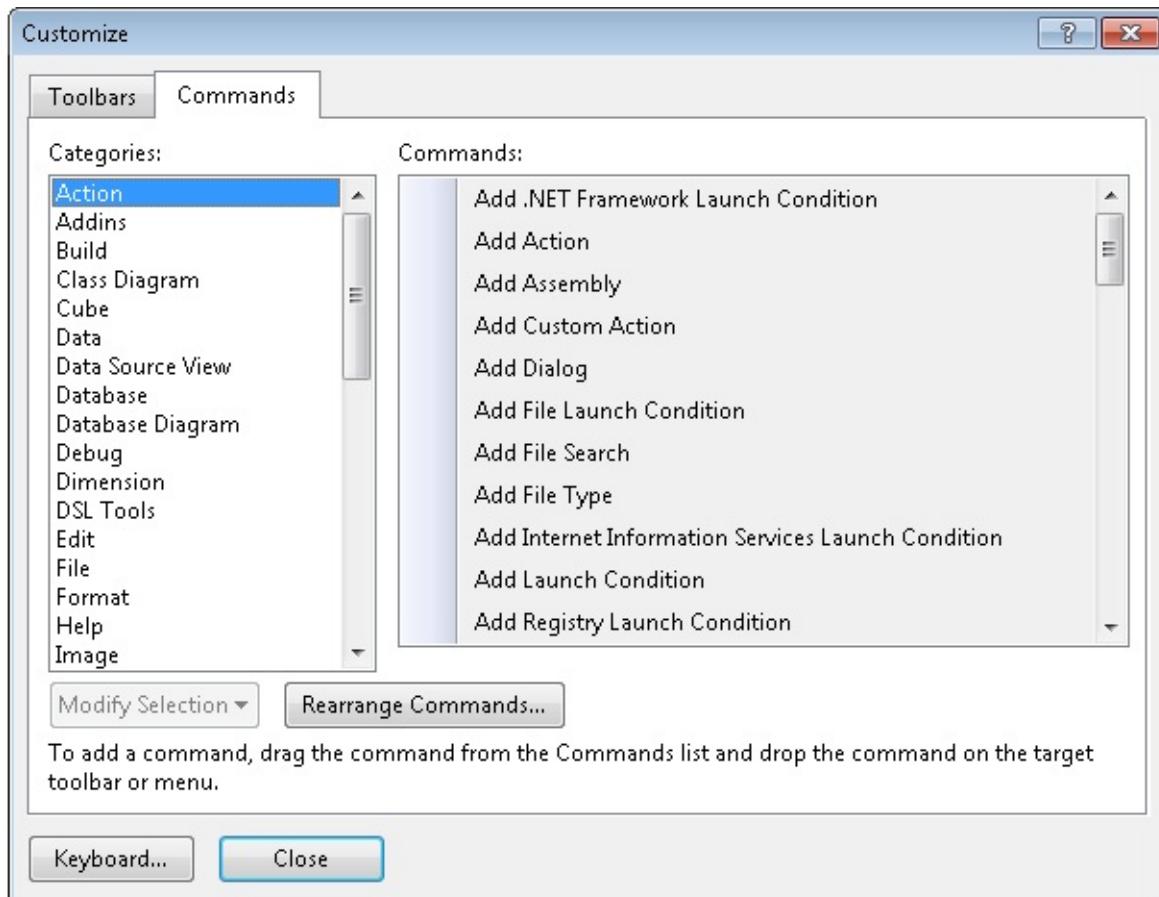
you choose opens the Customize dialog box:



In this case, let's look at the Commands tab:

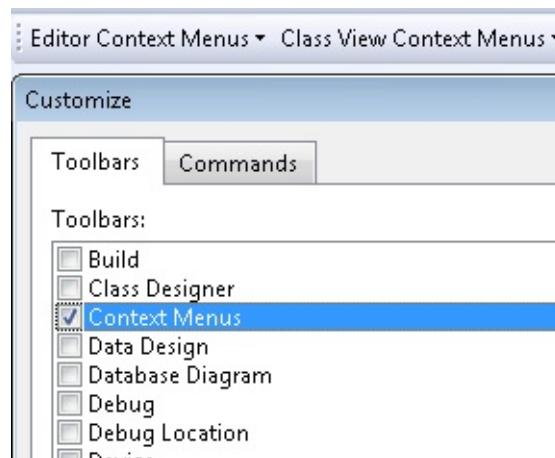
**NOTE**

To learn more about the Toolbars tab, see vtipEnv0032, [AX.07 Customize Your Toolbars in Visual Studio 2008: Toolbars Tab](#), on page A12.



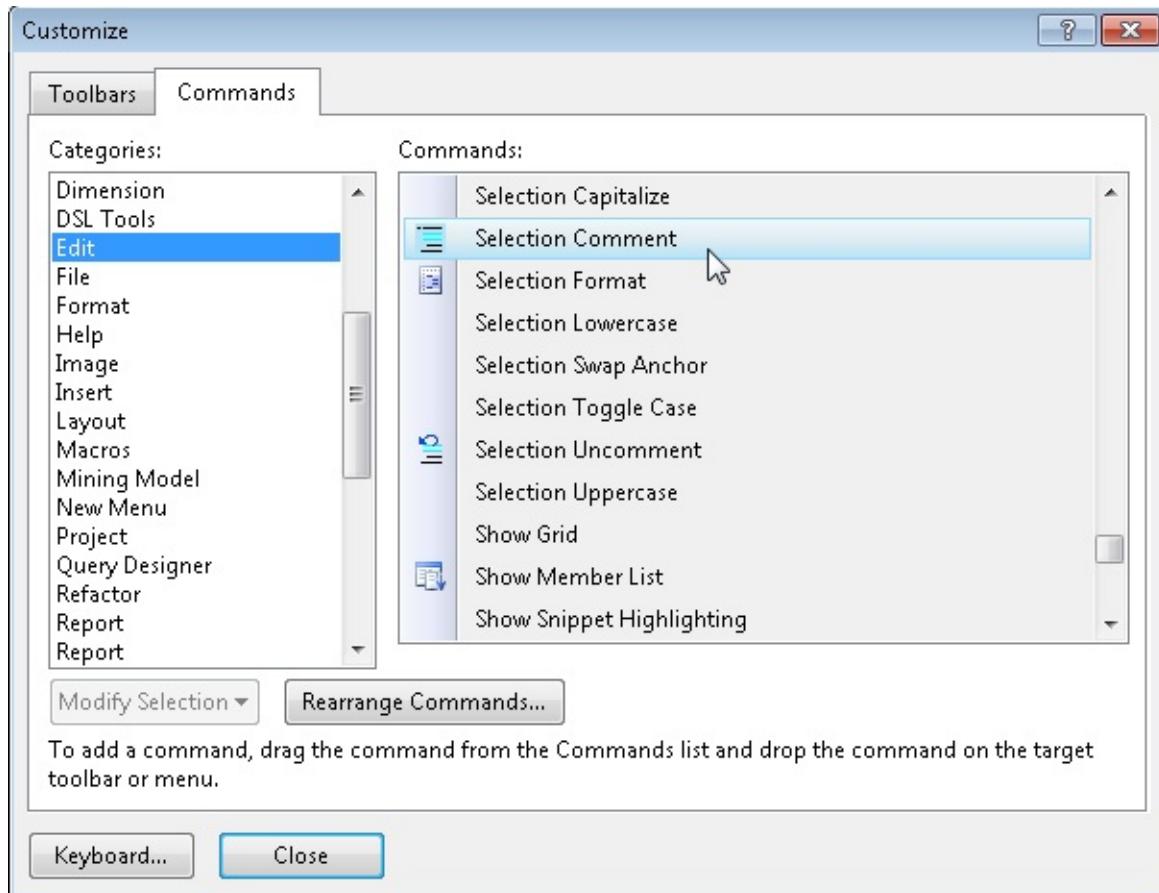
The best way to learn how to customize menus and toolbars is to work through an example. For our purposes, we want to add the ability to select some code, right-click, and comment or uncomment the code.

First, we have to go back to the Toolbars tab and pick the menu or toolbar we want to modify. For our example, let's choose Context Menus:

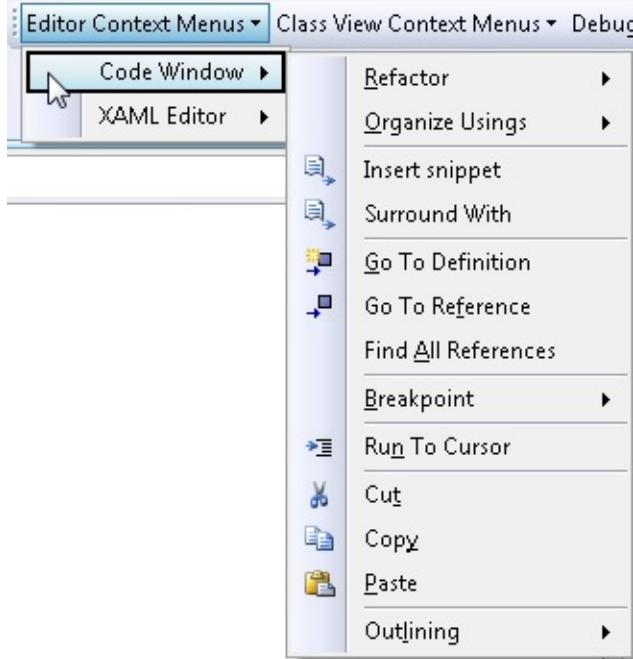


Now we click the Commands tab and locate the items we want to add. In this

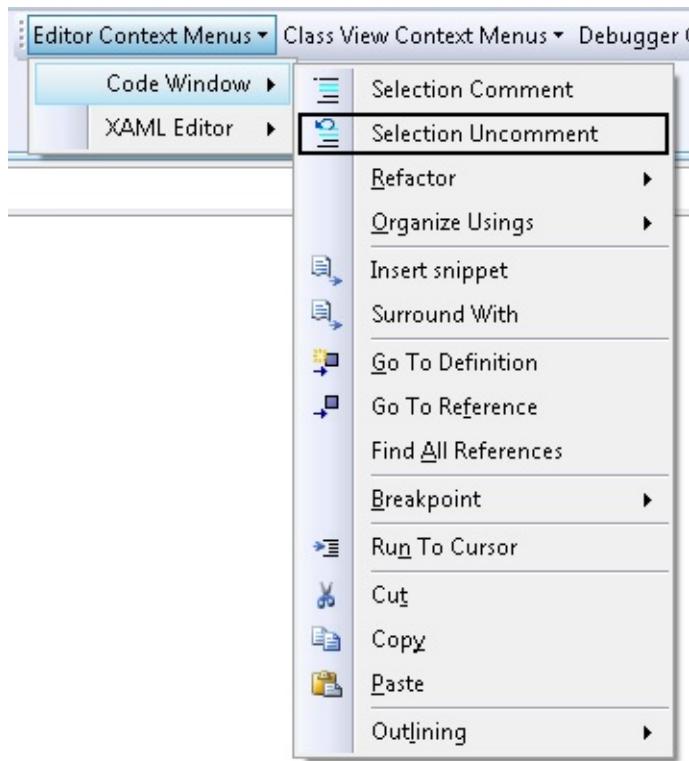
case, we dig a bit and find Selection Comment and Selection Uncomment in the Edit category:



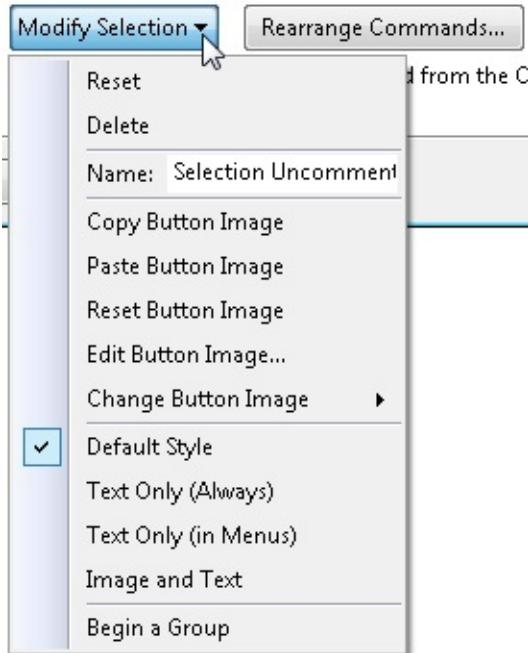
Now we have to see which menu we want to modify. Let's navigate to Editor Context Menus | Code Window to see where we want our items to go:



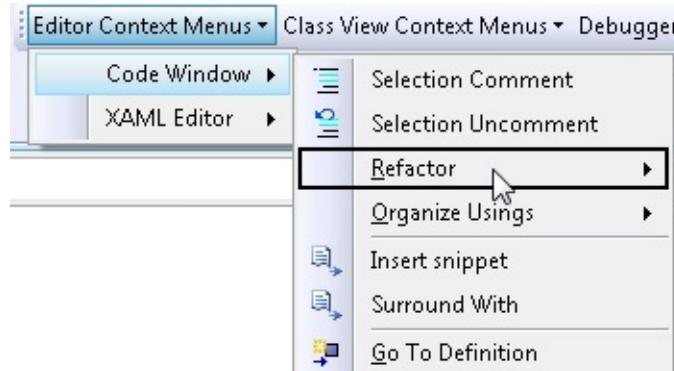
Now we drag the items where we want them on the menu:



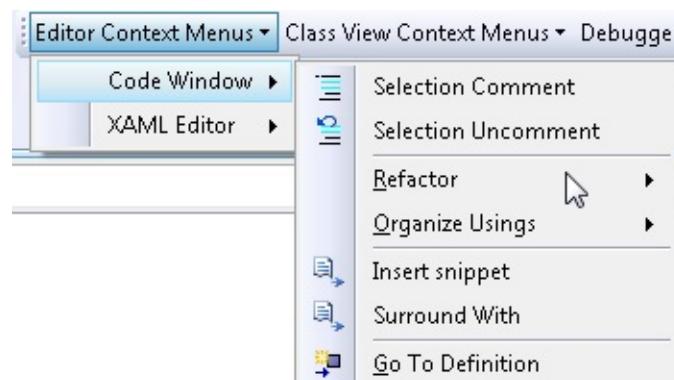
If we want, we can click Modify Selection to reset, delete, modify the name, modify the button image, change the way the item is displayed, or begin a new group:



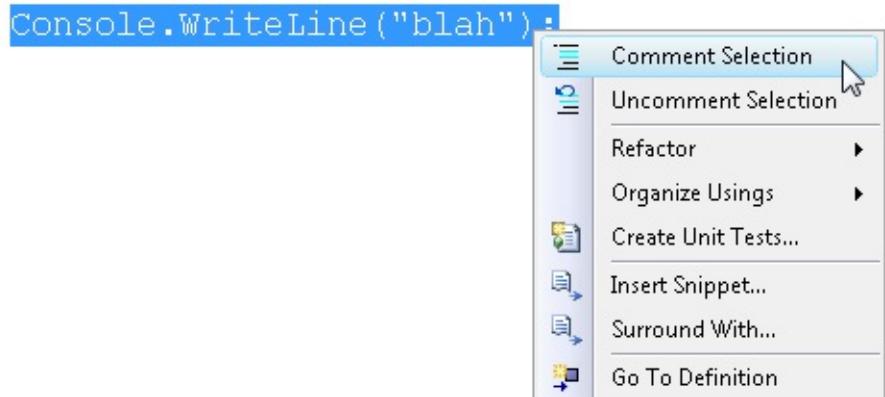
We would like the new buttons to be in their own group, so we click the item just below where we want our group to be:



Now we click Modify Selection and choose Begin A Group to get a new group line:



Close the Customize dialog box, select some code, and right-click to see whether our items show up:



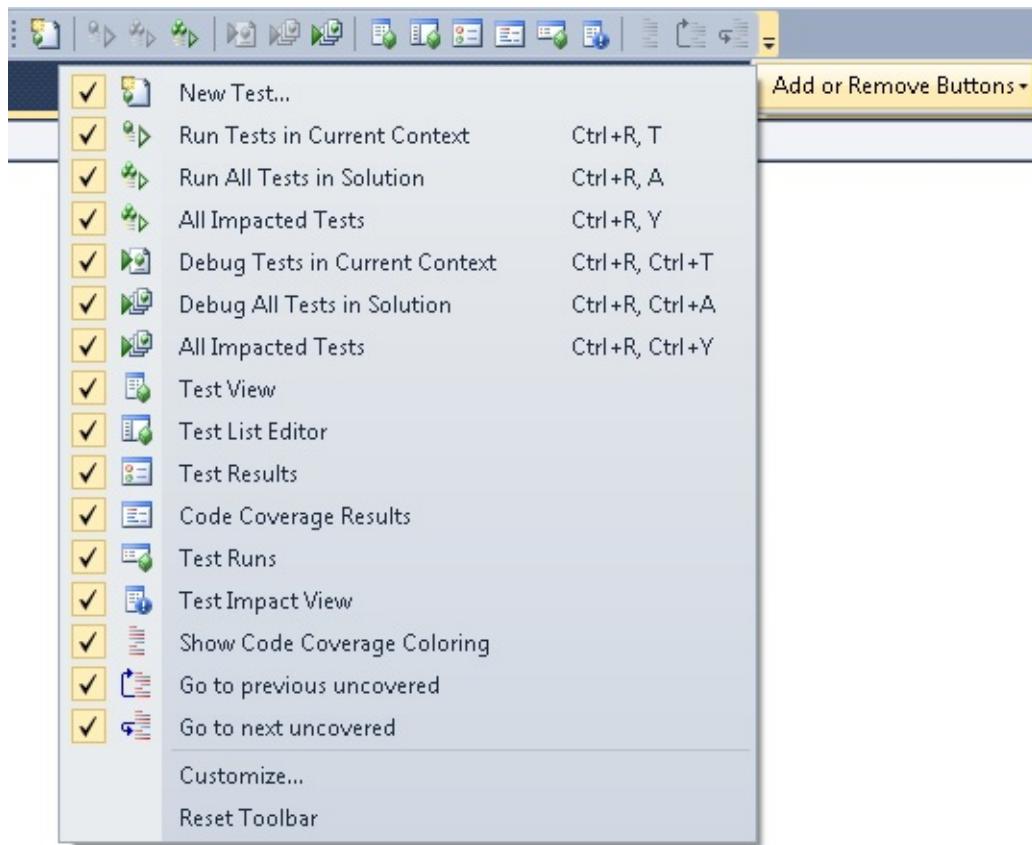
## AX.09 Hide or Show Default Buttons on a Toolbar

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0028

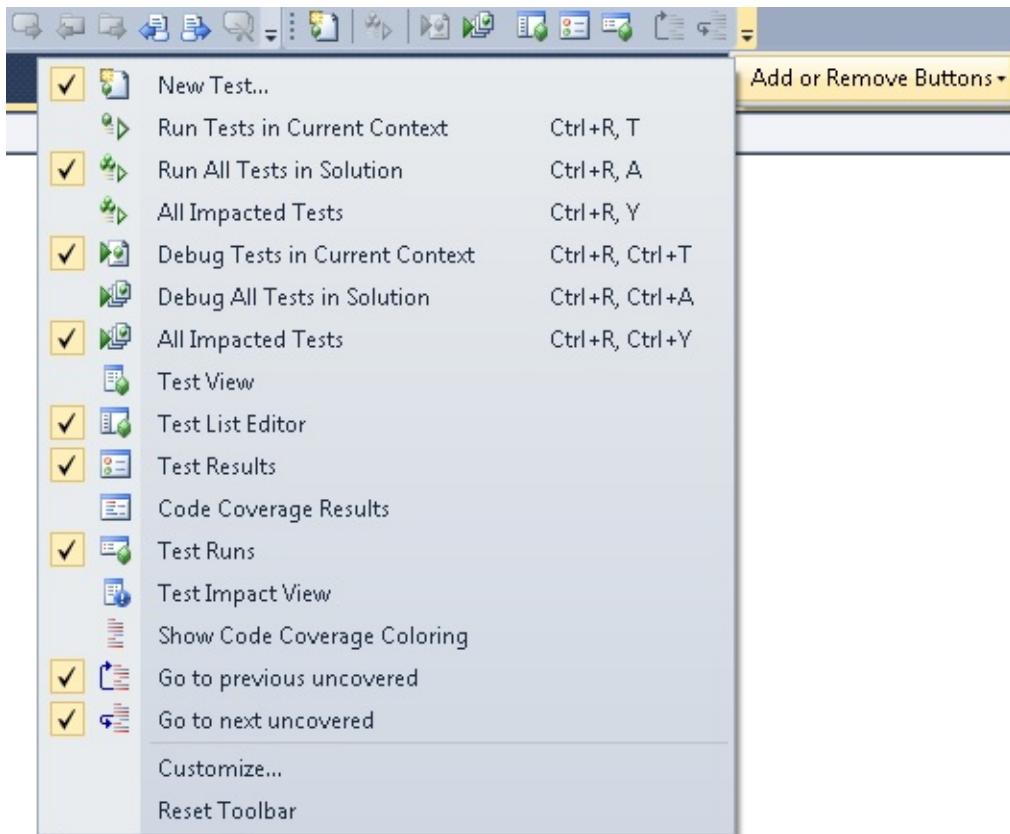
You can pick and choose which default buttons you want to show on any toolbar. Just click the drop-down arrow to the right of the toolbar:



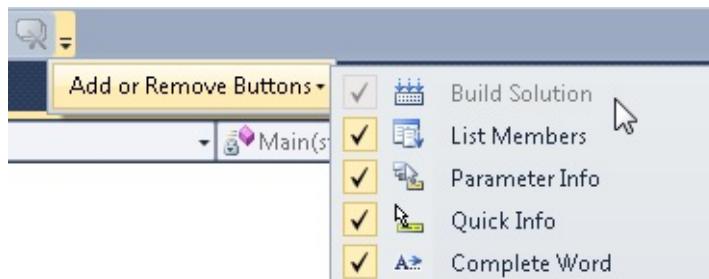
Then click Add or Remove Buttons:



As you can see, this gives you a list of the buttons currently being viewed. You can uncheck some of them to create a customized look for your toolbar:



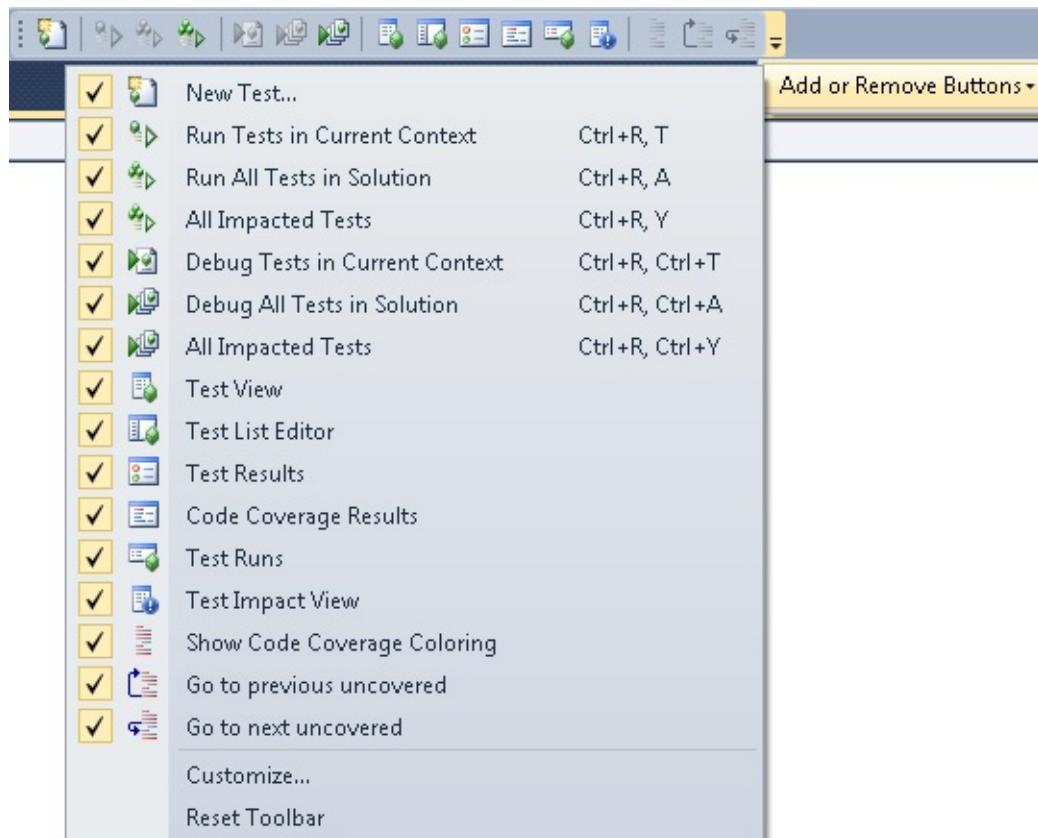
Unfortunately, if you have added customized buttons, they can't be hidden in this way:



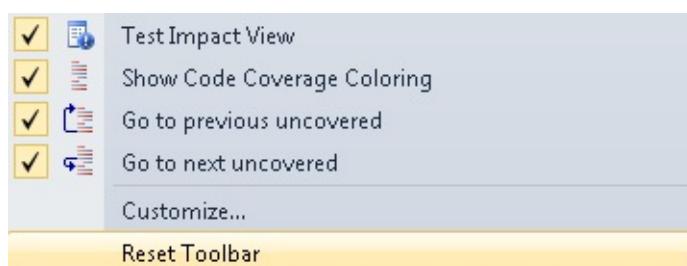
## AX.10 Reset Toolbars

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0029

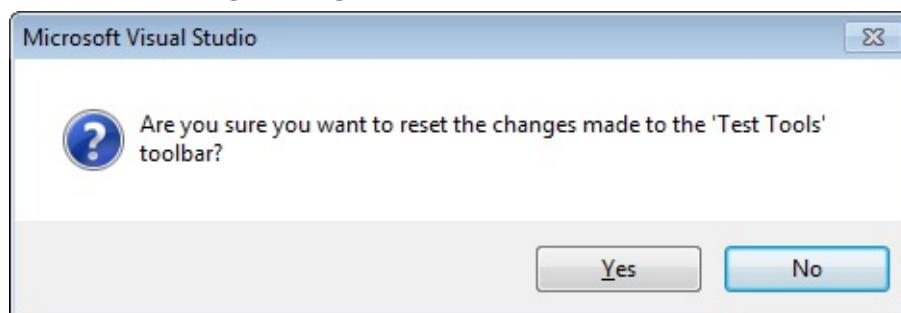
You can reset any toolbar to its default settings. Just click the drop-down arrow to the right of any toolbar, and then click Add or Remove Buttons:



Click Reset Toolbar:



You now see the following dialog box:



Click Yes to remove any custom buttons and to reset the toolbar to its default settings.

### **NOTE**

In Visual Studio 2008, you have to go through an extra menu to get to the Reset Toolbar option:

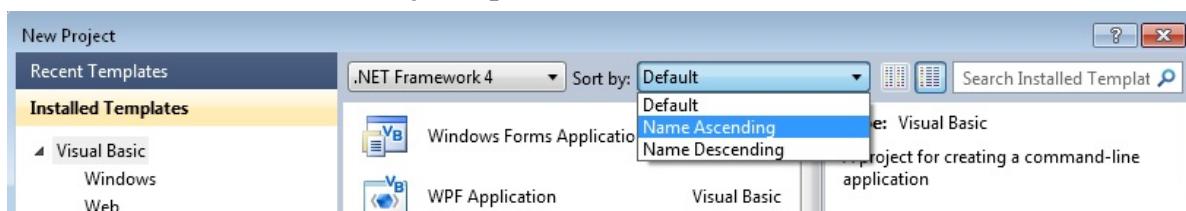


## Additional Tips from Chapter 2

### AX.11 Sorting Templates in the New Project Dialog Box

<b>DEFAULT</b>	Ctrl+Shift+N
<b>VISUAL BASIC 6</b>	Ctrl+Shift+N; Ctrl+N
<b>VISUAL C# 2005</b>	Ctrl+Shift+N
<b>VISUAL C++ 2</b>	Ctrl+Shift+N
<b>VISUAL C++ 6</b>	Ctrl+Shift+N
<b>VISUAL STUDIO 6</b>	Ctrl+N
<b>WINDOWS</b>	Alt,F, N, P (new project); Alt,F, D, N (add new project)
<b>MENU</b>	File   New Project; File   Add New Project
<b>COMMAND</b>	File.NewProject; File.AddNewProject
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipProj0003

Ever just want to have an alphabetical list of templates in the New Project dialog box? Just use the new Sort By drop-down list:

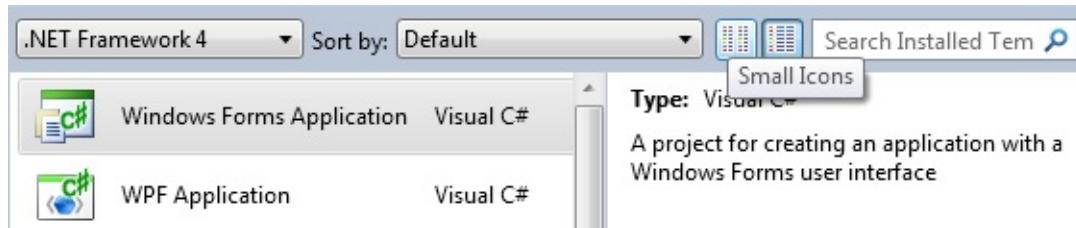


### AX.12 Toggle Icon Size in the New Project Dialog Box

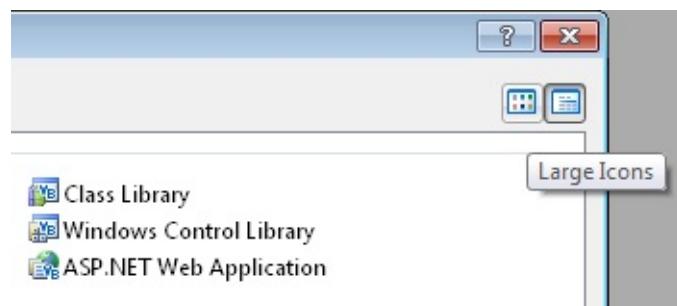
<b>DEFAULT</b>	Ctrl+Shift+N
<b>VISUAL BASIC 6</b>	Ctrl+Shift+N; Ctrl+N
<b>VISUAL C# 2005</b>	Ctrl+Shift+N

<b>VISUAL C++ 2</b>	Ctrl+Shift+N
<b>VISUAL C++ 6</b>	Ctrl+Shift+N
<b>VISUAL STUDIO 6</b>	Ctrl+N
<b>WINDOWS</b>	Alt, F, N, P (new project); Alt, F, D, N (add new project)
<b>MENU</b>	File   New   Project
<b>COMMAND</b>	File.NewProject
<b>VERSIONS</b>	2005,2008,2010
<b>CODE</b>	vstipProj0007

When you create a new project (or add a new item) in Visual Studio, you can change the icon size from small to medium (called *large* in Visual Studio 2008). To do this, you need to find the buttons that change the icon to your desired size and then click them. Unfortunately, these buttons appear in different places, depending on your Visual Studio version. In Visual Studio 2010, for example, the buttons are located toward the middle-right of the New Project or Item dialog box, next to the Sort By field:



In Visual Studio 2005 and Visual Studio 2008, the buttons are at the far right, as shown in the following illustration:

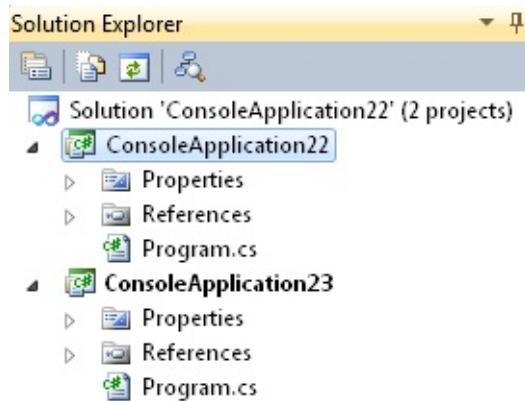


## AX.13 Choosing the StartUp Project

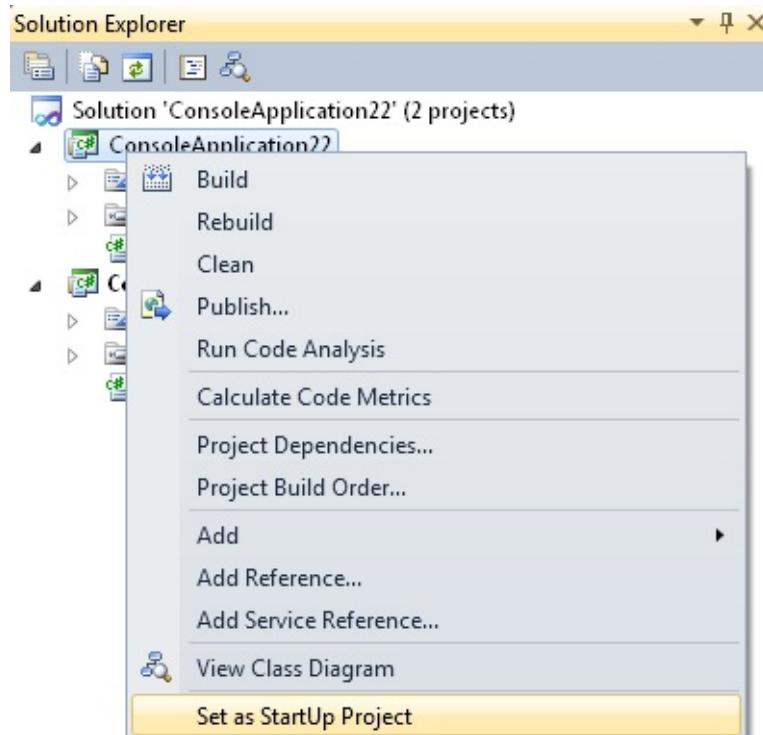
<b>WINDOWS</b>	Alt,P, A (with project selected in Solution Explorer); Shift+F10, A (with project selected in Solution Explorer)
----------------	--

<b>MENU</b>	Project   Set as StartUp Project; [Right-Click a project in Solution Explorer]   Set as StartUp Project
<b>COMMAND</b>	Project.SetasStartUpProject
<b>VERSIONS</b>	2005,2008,2010
<b>CODE</b>	vstipEnv0014

When you work with multiple projects, one is usually the StartUp Project. That's the one that starts up first when you start (with or without debugging). It's easy to spot the current StartUp Project because its name appears in bold type in Solution Explorer:



To quickly change the Startup Project, just right-click a project in Solution Explorer and choose Set As StartUp Project from the context menu:



## AX.14 Linked Items in Projects

<b>DEFAULT</b>	Shift+Alt+A; Ctrl+Shift+D
<b>VISUAL BASIC 6</b>	Ctrl+D; Shift+Alt+A
<b>VISUAL C# 2005</b>	Shift+Alt+A
<b>VISUAL C++ 2</b>	Shift+Alt+A
<b>VISUAL C++ 6</b>	Shift+Alt+A
<b>VISUAL STUDIO 6</b>	Shift+Alt+A; Ctrl+Shift+D
<b>WINDOWS</b>	Alt, P, G
<b>MENU</b>	Project   Add Existing Item
<b>COMMAND</b>	Project.AddExistingItem
<b>VERSIONS</b>	2005,2008,2010
<b>CODE</b>	vstipProj0022

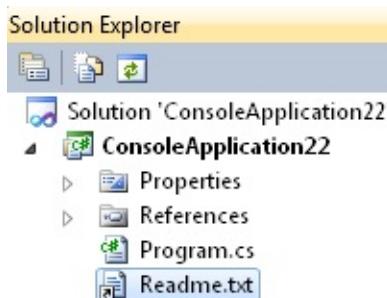
You sometimes have a shared resource that you want to include in your project. Traditionally, you would add the existing item (Shift+Alt+A), and Visual Studio

would make a copy for you.

However, did you know you can just link to the item instead? You would typically do this if you had a shared resource on, say, a network drive that you want to include but don't want to have a copy in your project. To do this, go to the Add Existing Item dialog box (Shift+Alt+A) and choose Add As Link from the Add drop-down list.



Now a link to the file is added to your project rather than a copy:



You can tell a file is a link because it has an arrow indicator in the icon, as shown in the preceding graphic—much like the arrow you see on shortcut icons in Windows. The usual caveats apply here, just as they do to any shortcut. For example, for Visual Studio to use the file, you need to make sure that the path to the linked file is accessible.

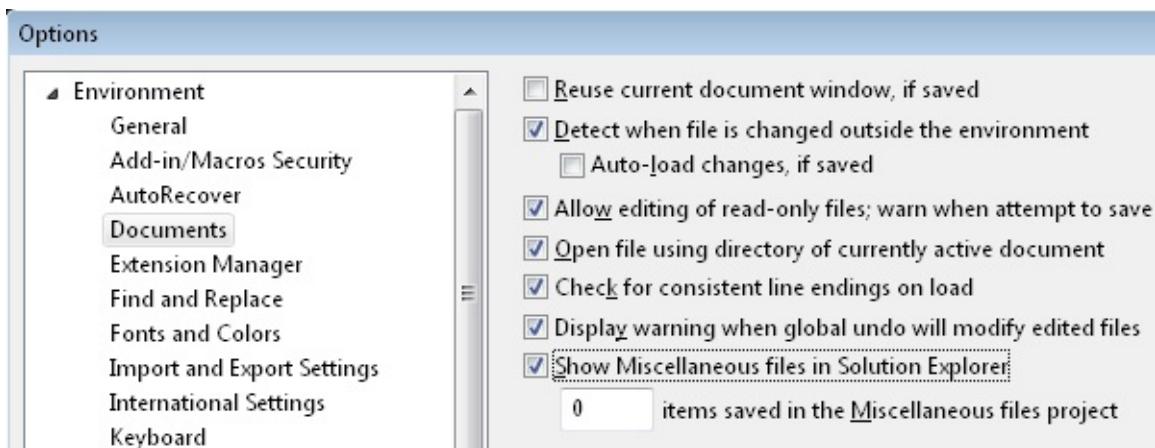
## AX.15 Using the Miscellaneous Files Project

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Documents
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2008,2010
<b>CODE</b>	vstipProj0012

Did you know that Solution Explorer can display a Miscellaneous Files project for files that you do not want to permanently associate with a project or solution?

Maybe you still want to track and be able to open certain files quickly—but not associate them with the project or solution. For example, you can create and edit files by using the Visual Studio editors without creating a project. You can also work on files that you want to use temporarily—such as files in which you keep development notes while you work on your solution.

To get the Miscellaneous Files folder to show up, go to Tools | Options | Environment | Documents and select the Show Miscellaneous Files In Solution Explorer option:

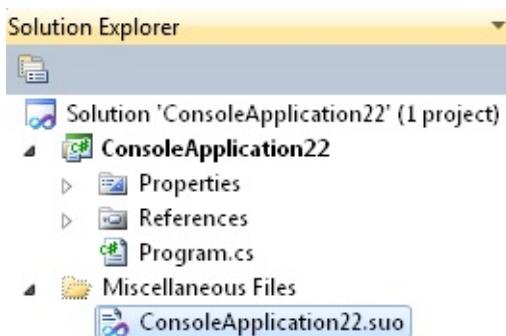


Pay particular attention to the Items Saved In The Miscellaneous Files Project field, which is initially set to zero. Leaving this value set to zero shows only extra files you currently have open; however, setting it to another number indicates how many files you want to remember at any given time. If you aren't sure about what this value should be, to get started, I suggest setting this value to at least five.

#### NOTE

The Miscellaneous Files Project does not show up until you open up at least one file that goes in it.

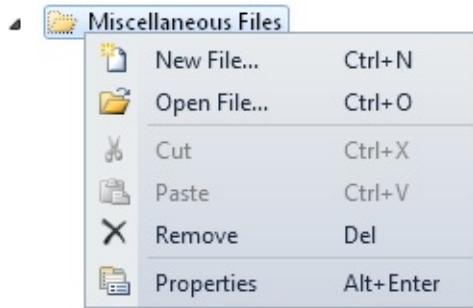
After you open a file that doesn't belong to the solution, the folder shows up, as shown in the following illustration:



With the folder now showing, you can right-click and open, create, or remove files:

**NOTE**

Removing files from this folder doesn't delete them permanently.



## AX.16 Change the Order of Your Application Settings

VERSIONS	2008,2010
LANGUAGES	C#, VB
CODE	vstipProj0024

When working in your project properties, you might sometimes find yourself using a variety of application settings (I'm assuming you know how to create and use application settings; if not, you can find more information at <http://msdn.microsoft.com/en-us/library/a65txeh.aspx>):

Resources	Name	Type	Scope	Value
Services	BackgroundCol...	System.Dra...	User	0, 51, 204
Settings*	ForegroundColor	System.Dra...	User	204, 102, 153
Reference Paths	Path	string	Application	..\..\stuff
Signing	WindowColor	System.Dra...	User	255, 80, 80
Security	SettingsName	string	User	Bubba's Stuff
*	Setting	string	User	

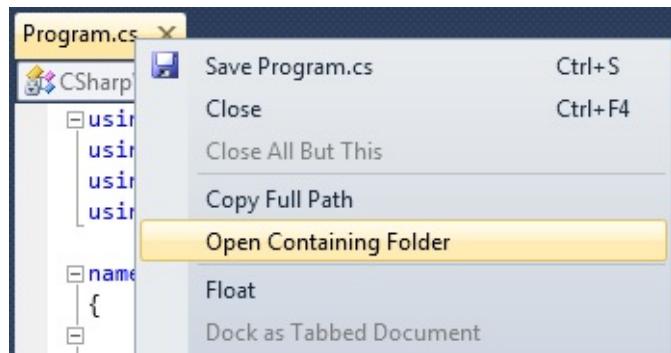
But what if you want to organize these settings? For example, suppose you want the WindowColor setting to appear with the other color settings. You can change

the sequence to your preference.

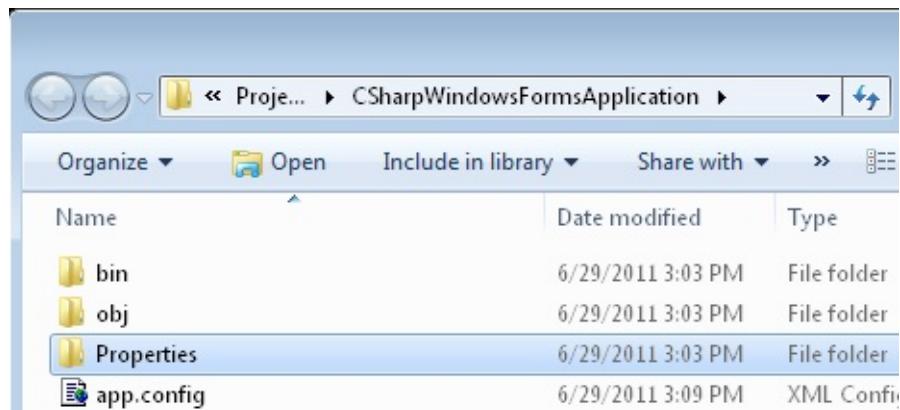
**WARNING**

Changing a setting requires you to manually edit your settings file—which could get you into trouble if you make a mistake, so do this at your own risk. Also, make a copy of your settings file before editing it so that you can recover if a problem occurs.

First, save any pending changes, and close the Properties window. Now open up the project location:

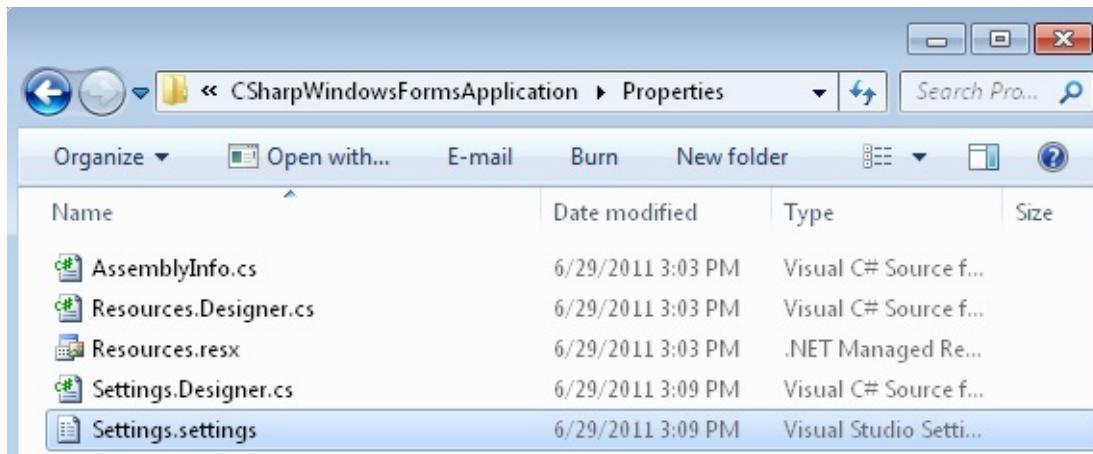


Locate your Properties folder:



Name	Date modified	Type
bin	6/29/2011 3:03 PM	File folder
obj	6/29/2011 3:03 PM	File folder
Properties	6/29/2011 3:03 PM	File folder
app.config	6/29/2011 3:09 PM	XML Config

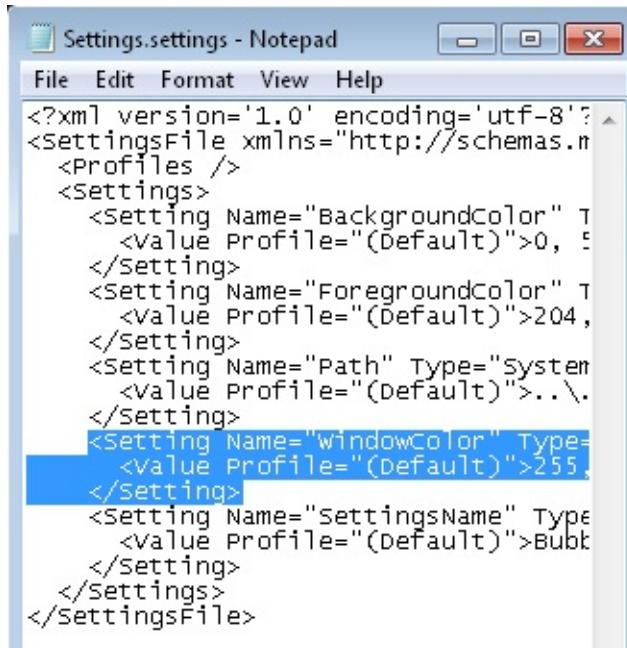
Locate your Settings file, named Settings.settings in this example:



Open the Settings.settings file with a plain text editor such as Notepad. Notice that the file is an XML file:

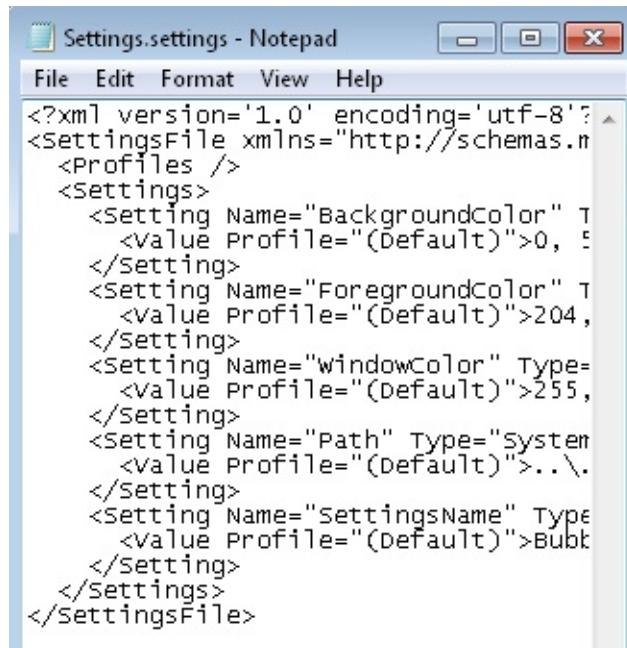
```
<?xml version='1.0' encoding='utf-8'?>
<SettingsFile xmlns="http://schemas.microsoft.com/2003/10/Serialization/settings">
  <Profiles />
  <Settings>
    <Setting Name="BackgroundColor" Type="System.String">
      <value Profile="(Default)">0, 51, 102, 255</value>
    </Setting>
    <Setting Name="ForegroundColor" Type="System.String">
      <value Profile="(Default)">204, 204, 204, 255</value>
    </Setting>
    <Setting Name="Path" Type="System.String">
      <value Profile="(Default)">..\AppData\Local\Temp\</value>
    </Setting>
    <Setting Name="WindowColor" Type="System.String">
      <value Profile="(Default)">255, 255, 255, 255</value>
    </Setting>
    <Setting Name="SettingsName" Type="System.String">
      <value Profile="(Default)">Bubble</value>
    </Setting>
  </Settings>
</SettingsFile>
```

As you see in the preceding illustration, each setting resides in a `<Setting>` element. You just need to rearrange the elements in the order you want them. In this case, select the `<Setting>` element whose name attribute is `WindowColor`, as shown in the following illustration:



```
<?xml version='1.0' encoding='utf-8'?>
<SettingsFile xmlns="http://schemas.microsoft.com/2003/10/Serialization/Settings">
  <Profiles />
  <Settings>
    <Setting Name="BackgroundColor" Type="Color">
      <Value Profile="(Default)">0, 51, 102, 255</Value>
    </Setting>
    <Setting Name="ForegroundColor" Type="Color">
      <Value Profile="(Default)">204, 204, 204, 255</Value>
    </Setting>
    <Setting Name="Path" Type="System.String">
      <Value Profile="(Default)">..\</Value>
    </Setting>
    <Setting Name="WindowColor" Type="Color">
      <Value Profile="(Default)">255, 255, 255, 255</Value>
    </Setting>
    <Setting Name="SettingsName" Type="System.String">
      <Value Profile="(Default)">Bubble</Value>
    </Setting>
  </Settings>
</SettingsFile>
```

Next just cut and paste it where you want the setting to show up. For this example, place it just after the ForegroundColor <Setting> element:



```
<?xml version='1.0' encoding='utf-8'?>
<SettingsFile xmlns="http://schemas.microsoft.com/2003/10/Serialization/Settings">
  <Profiles />
  <Settings>
    <Setting Name="BackgroundColor" Type="Color">
      <Value Profile="(Default)">0, 51, 102, 255</Value>
    </Setting>
    <Setting Name="ForegroundColor" Type="Color">
      <Value Profile="(Default)">204, 204, 204, 255</Value>
    </Setting>
    <Setting Name="WindowColor" Type="Color">
      <Value Profile="(Default)">255, 255, 255, 255</Value>
    </Setting>
    <Setting Name="Path" Type="System.String">
      <Value Profile="(Default)">..\</Value>
    </Setting>
    <Setting Name="SettingsName" Type="System.String">
      <Value Profile="(Default)">Bubble</Value>
    </Setting>
    <Setting Name="NewSetting" Type="System.String">
      <Value Profile="(Default)">New Value</Value>
    </Setting>
  </Settings>
</SettingsFile>
```

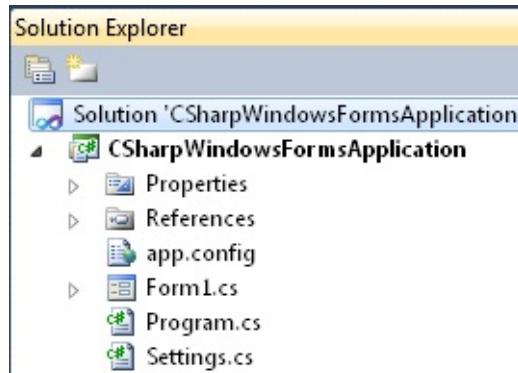
Save your changes and close the file. Now, when you go back into your Project properties, you'll see the settings arranged in their new order:

Resources	Name	Type	Scope	Value
Services	BackgroundCol...	System.Dra...	User	0, 51, 204
Settings	ForeColor	System.Dra...	User	204, 102, 153
Reference Paths	WindowColor	System.Dra...	User	255, 80, 80
Signing	Path	string	Application	..\..\stuff
Security	SettingsName	string	User	Bubba's Stuff
*				

## AX.17 Hide or Show the Solution File in Solution Explorer

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Projects and Solutions   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005,2008,2010
<b>CODE</b>	vstipProj0008

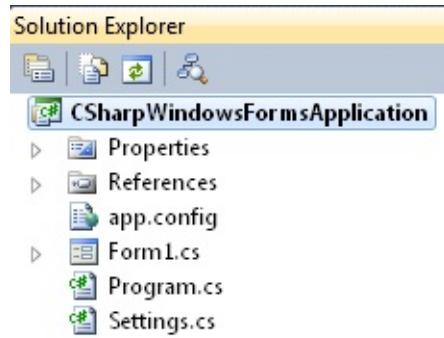
If you don't like seeing the solution file in Solution Explorer, you can easily hide it (or show it if you have it hidden). First, take a look at the default view of Solution Explorer, with the solution file showing:



To hide the solution file, select Tools | Options | Projects And Solutions | General, and clear the Always Show Solution option:

<input checked="" type="checkbox"/> Show advanced build configurations
<input type="checkbox"/> Always show solution
<input checked="" type="checkbox"/> Save new projects when created

The result is shown below:



#### NOTE

This feature works only when you have just one project in the solution; if you have multiple projects in your solution, Visual Studio ignores this setting and shows you the solution node.

## AX.18 New Project Dialog Preferred Language

<b>WINDOWS</b>	Alt,T,I
<b>MENU</b>	Tools   Import and Export Settings
<b>COMMAND</b>	Tools.ImportandExportSettings
<b>VERSIONS</b>	2005,2008,2010
<b>CODE</b>	vstipEnv0041

As you're examining items that Visual Studio can export, you might come across the New Project Dialog Preferred Language option, under General Settings:

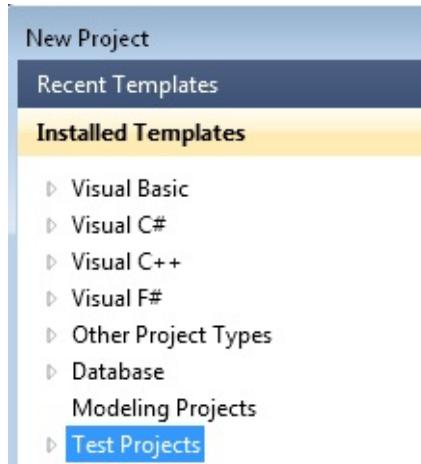
A screenshot of a 'Which settings do you want to export?' dialog. It has a list of checkboxes on the left and a description on the right. The checkboxes include: Find Options, Find Symbol Options, Menu and Command Bar Customizations, New Project Dialog Preferred Language (which is selected), and Object Browser Options. The description for 'New Project Dialog Preferred Language' states: 'Specifies the organization of the folder nodes in the New Project dialog.'

You might wonder what that means. The best way to explain is to show you a little bit more about the inner workings of Visual Studio.

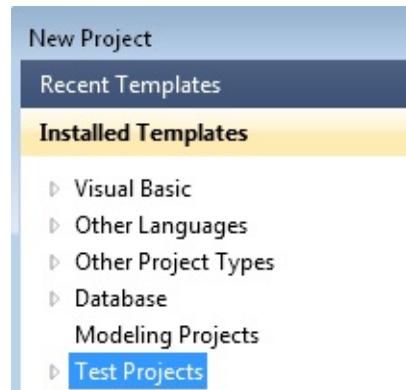
When you installed Visual Studio, you might recall having chosen your preferred development settings:

-  General Development Settings
-  Project Management Settings
-  Visual Basic Development Settings
-  Visual C# Development Settings
-  Visual C++ Development Settings
-  Visual F# Development Settings
-  Web Development
-  Web Development (Code Only)

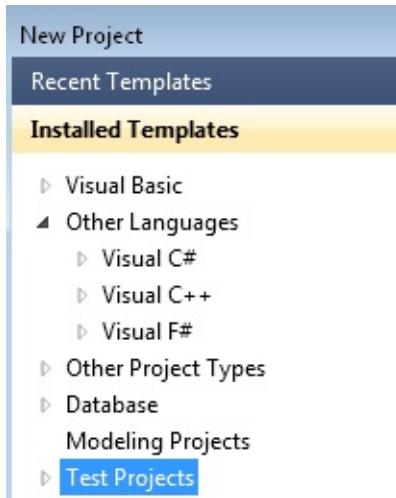
Some of these options are language-specific, and some are not. If you chose General Development Settings, for example, you have no preferred language in the New Project dialog box, so you see a list of languages:



Without a preferred language, the New Project dialog box groups all the language-specific templates into separate nodes, so you can easily select the one you want. However, if you chose a development setting option associated with a particular language, such as VB, you see something different in the New Project dialog box:



As you can see, the other top-level language nodes have disappeared, because you already indicated that Visual Basic is your preferred language. Instead, you'll see an Other Languages node that represents all the other languages:



So, to come full circle, when you export the New Project Dialog Preferred Language item, it saves this structure for you, which you can then later import.

## AX.19 Optimizing Your Project Code

<b>DEFAULT</b>	Alt+Enter (with project selected in Solution Explorer)
<b>VISUAL BASIC 6</b>	Alt+Enter (with project selected in Solution Explorer)
<b>VISUAL C# 2005</b>	Alt+Enter (with project selected in Solution Explorer)
<b>VISUAL C++ 2</b>	Alt+Enter (with project selected in Solution Explorer)
<b>VISUAL C++ 6</b>	Alt+F7; Alt+Enter (with project selected in Solution Explorer)
<b>WINDOWS</b>	Alt,P, P
<b>MENU</b>	Project   [Project Name] Properties
<b>COMMAND</b>	Project.Properties
<b>VERSIONS</b>	2008,2010
<b>LANGUAGES</b>	C#, C++, VB
<b>CODE</b>	vstipProj0014

In vtipDebug0032 ([AX.132 Understanding Just My Code](#), page A215), we touched on optimization. When optimization is turned off (the default setting for Debug builds), it factors into the code being considered “yours” for the purposes of determining what is “Just My Code.” Generally, you won’t turn this on for Debug builds. If you do, debug symbols are not generated and you can’t step

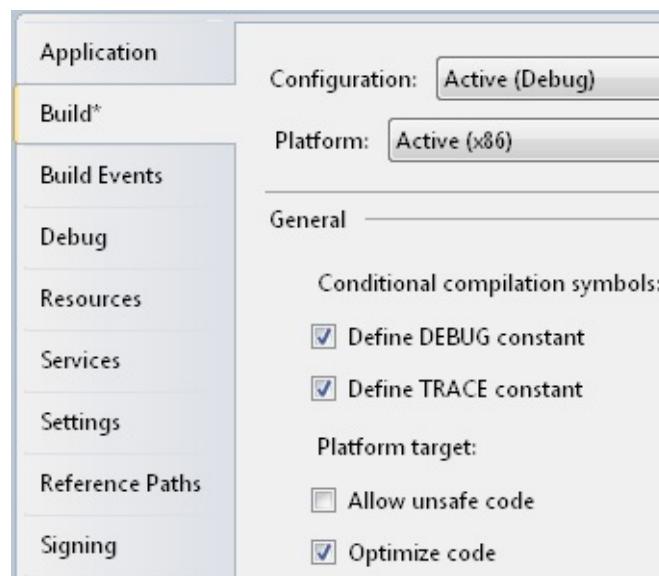
through your code.

When you create a Release build, Visual Studio turns optimization on by default. So what is optimization? According to the documentation, the optimization option “enables or disables optimizations performed by the compiler to make your output file smaller, faster, and more efficient.”

It’s useful to know where to find this option.

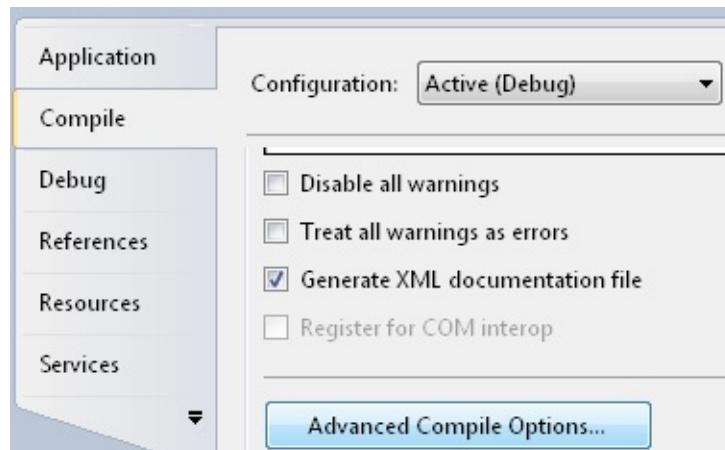
## C#

In C#, you’ll find the Optimize Code option in the Project properties on the Build tab:

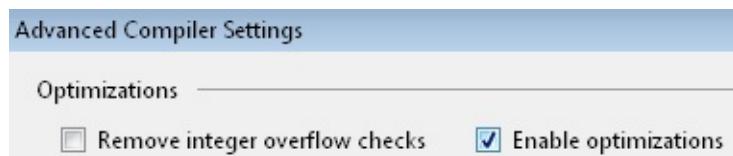


## VB

In VB, it is also in the Project properties, but on the Compile tab, and you need to click Advanced Compile Options to find it:



Next, locate the Enable Optimizations option:



## C++

While VB and C# have only a single option to control optimization, C++ supports a number of more specific optimizations. For example, you can choose to optimize for application speed or for program size. Enabling the optimizations setting from the Project properties enables full optimization (/Ox).

You can get a sense of the full range of /O features in the following list at <http://msdn.microsoft.com/en-us/library/k1ack8f1.aspx>:

- /O1 optimizes code for minimum size.
- /O2 optimizes code for maximum speed.
- /Ob controls inline function expansion.
- /Od disables optimization, speeding compilation and simplifying debugging.
- /Og enables global optimizations.
- /Oi generates intrinsic functions for appropriate function calls.
- /Os tells the compiler to favor optimizations for size over optimizations for speed.
- /Ot (a default setting) tells the compiler to favor optimizations for speed over optimizations for size.
- /Ox selects full optimization.
- /Oy suppresses the creation of frame pointers on the call stack for quicker function calls.

## Finally

Optimization does a lot of cool things that are great for a shipping application—but not for one you are currently working on and debugging. It's easy to get deep into what the various optimization options actually do; Eric Lippert, of Microsoft, wrote an excellent article on this subject, titled “What Does the Optimize Switch Do,” which you can find at

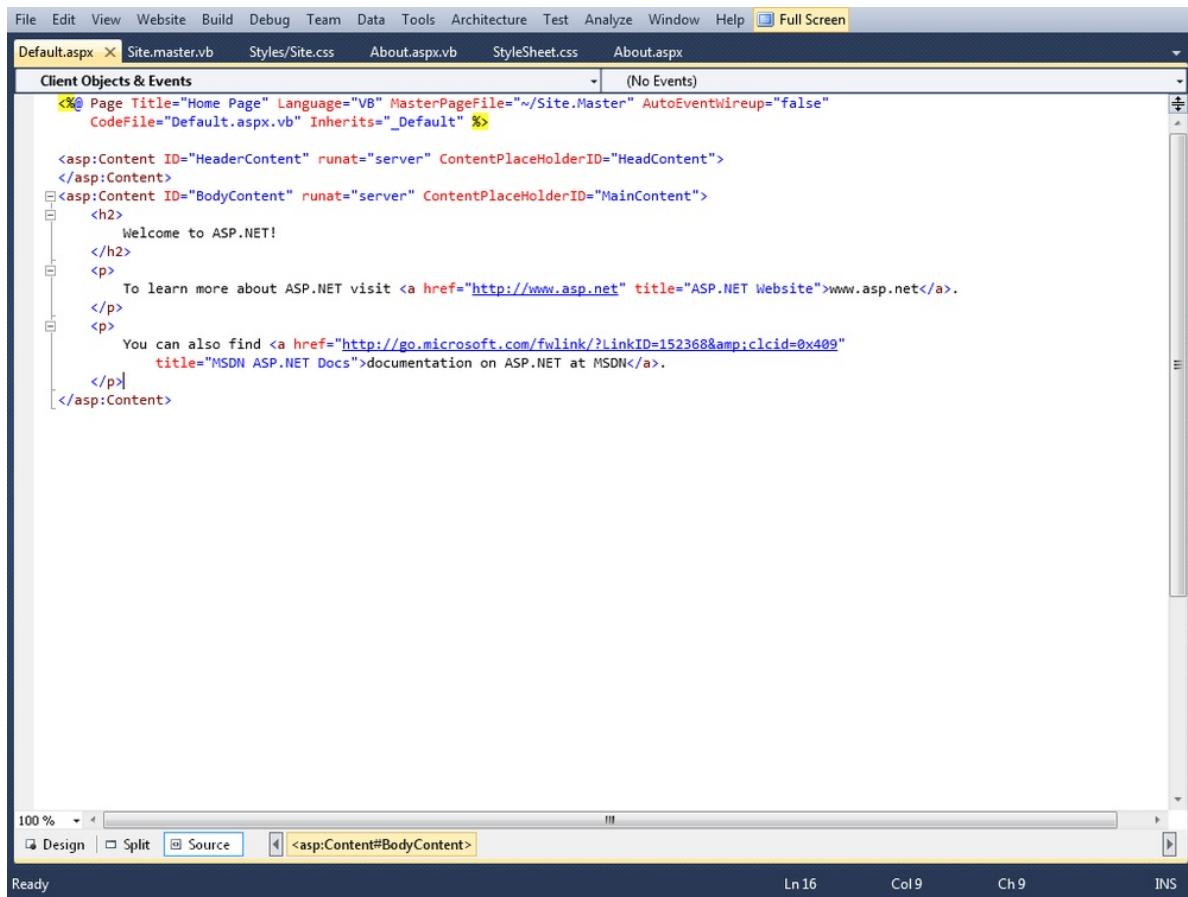
<http://blogs.msdn.com/b/ericlippert/archive/2009/06/11/what-does-the-optimize-switch-do.aspx>.

## Additional Tips from Chapter 3

### AX.20 Full Screen Mode

<b>DEFAULT</b>	Shift+Alt+Enter
<b>VISUAL BASIC 6</b>	Shift+Alt+Enter
<b>VISUAL C# 2005</b>	Shift+Alt+Enter
<b>VISUAL C++ 2</b>	Shift+Alt+Enter
<b>VISUAL C++ 6</b>	Shift+Alt+Enter
<b>VISUAL STUDIO 6</b>	Shift+Alt+Enter
<b>WINDOWS</b>	Alt,V, U
<b>MENU</b>	View   Full Screen
<b>COMMAND</b>	View.FullScreen
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0024

You can quickly switch from any current window state to Full Screen Mode by pressing Shift+Alt+Enter:



By default, this action hides the toolbars and takes up as much of the screen as possible. The advantage, of course, is that you get more real estate to work with when you need more room.

To come out of Full Screen Mode, just press Shift+Alt+Enter again and you are returned to normal view.

## AX.21 Split Your Windows Horizontally

<b>WINDOWS</b>	Alt,W, P (toggle split and remove)
<b>MENU</b>	Window   Split; Window   Remove Split
<b>COMMAND</b>	Window.Split (toggle split and remove)
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0004

In Visual Studio, you can split your windows horizontally. This feature has been available in Microsoft products for quite a while. Just go to Window | Split on the

menu bar, or you can use the following mouse technique.

Go the upper-right corner of a document window, and look for the splitter control, as shown in the following illustration:



Click and drag the control down to begin the split process:



Now you have a horizontal split, so you can do things like see different sections of your document at the same time:

A screenshot of the Visual Studio IDE showing a horizontal split between two code editors. Both editors display the same ASP.NET page source code. The top editor shows the initial page content, and the bottom editor shows additional content that was added after the split. The interface includes standard Windows-style scroll bars and a status bar at the bottom.

To remove a split, select Window | Remove Split—or just double-click the line separating the two sections.

## AX.22 Sorting Items in the Toolbox

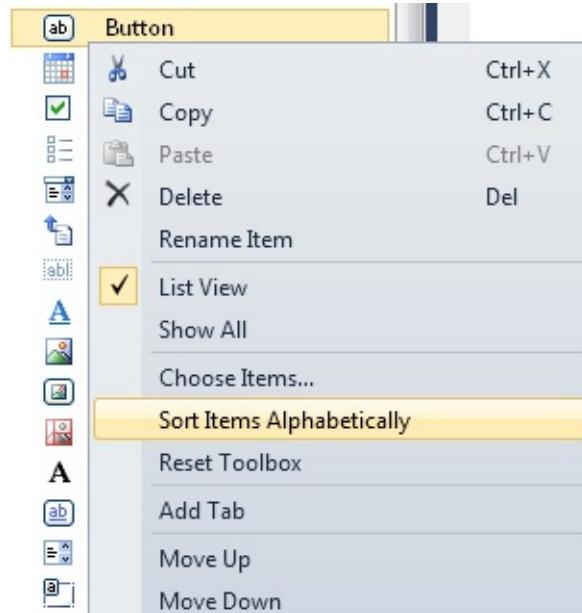
<b>WINDOWS</b>	Shift,F10, O (with the Toolbox selected)

<b>MENU</b>	[Right Click the Toolbox]   Sort Items Alphabetically
<b>COMMAND</b>	Tools.SortItemsAlphabetically
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0052

### WARNING

As far as I know, there is no easy way to undo the sorting action other than resetting the Toolbox, so make sure you really want the Toolbox items listed alphabetically before you apply this tip.

If you don't like the default sort order for items in the Toolbox, you can right-click the Toolbox and then choose to sort the items alphabetically:



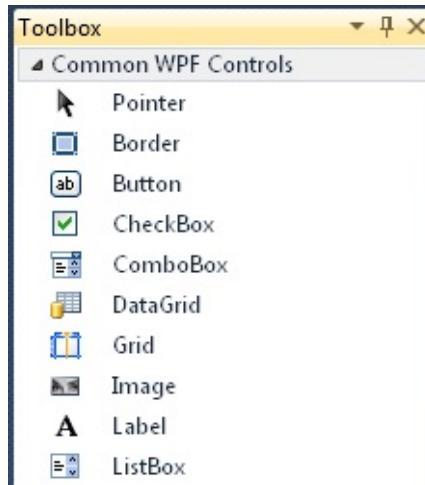
Selecting this option sorts the items by name, assuming they weren't sorted that way already:



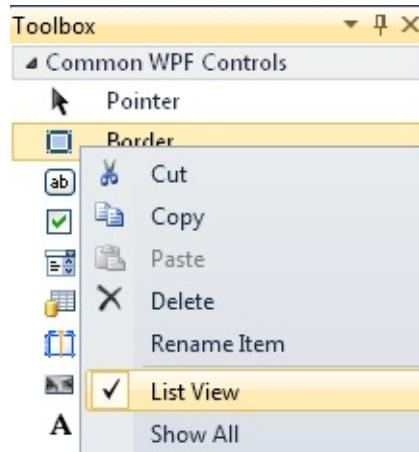
## AX.23 Icon vs. List View in the Toolbox

<b>WINDOWS</b>	Shift,F10, L (with the Toolbox selected will toggle List View)
<b>MENU</b>	[Right Click the Toolbox]   List View (toggle)
<b>COMMAND</b>	Tools.ListView
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0053

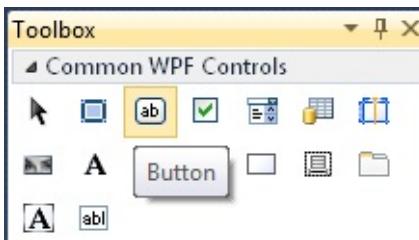
The default view of items in the Toolbox is List View, as shown in the following illustration:



You can change this to the Icon View if you prefer, by right-clicking the Toolbox and clicking List View, which turns off the List View option:

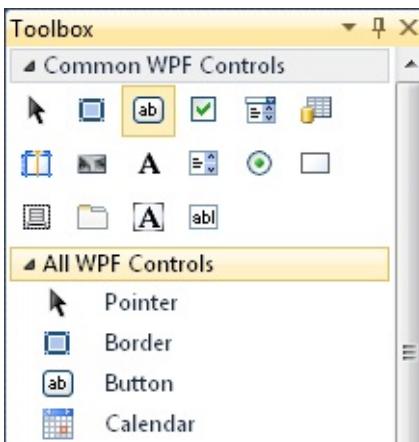


This produces the result shown in the following illustration:



To get the List View back again, right-click in Icon View, and click List View again.

This option applies only on the current Toolbox tab, so you can have your tabs organized differently based on your preference:



## AX.24 Hide the Status Bar

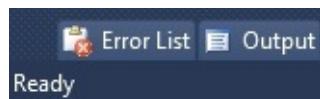
WINDOWS	Alt,T, O
MENU	Tools   Options   Environment   General

<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0024

Removing the Status Bar gives you a little extra space at the bottom of your screen. Without it you cannot get status messages, so remove it only if you are sure you don't need it. To remove the Status Bar, go to Tools | Options | Environment | General and clear the Show Status Bar option:



Before (status bar is where the word “Ready” is located):



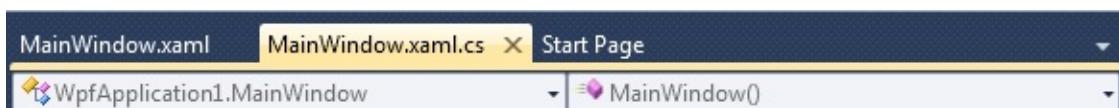
After (no status bar):



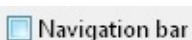
## AX.25 Remove the Navigation Bar

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   [All Languages or Specific Language]
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0028

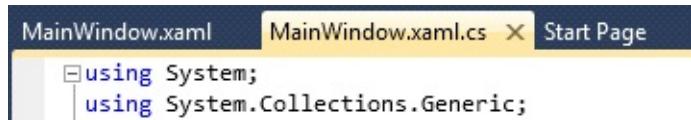
If you find you don't use the navigation bar, shown in the following illustration, and want a little extra space, how can you make it go away?



Just go to Tools | Options | Text Editor | [All Languages or Specific Language], and clear the Navigation Bar check box:



The following illustration shows the result:

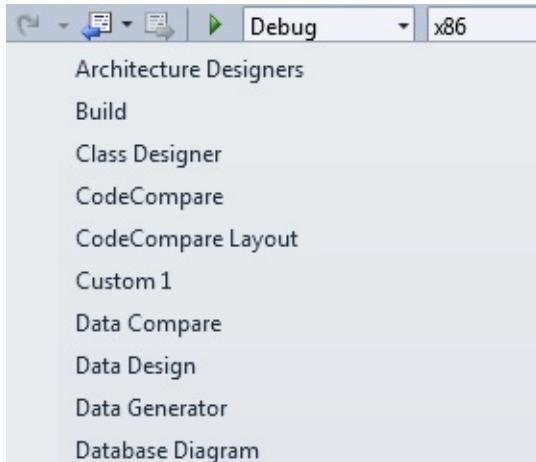


```
MainWindow.xaml      MainWindow.xaml.cs X Start Page
using System;
using System.Collections.Generic;
```

## AX.26 Show Any Toolbar

<b>WINDOWS</b>	Alt,V, T, [Up or Down] Arrow
<b>MENU</b>	View   Toolbars   [Toolbar]
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0025

You can show any toolbar at any time. Just right-click an existing toolbar, and left-click the one you want to see:



Not all toolbars will have their buttons available, because it depends on your context. For example, the Database Diagram Toolbar (shown in the following illustration) buttons are not available until you are actually working on a diagram:

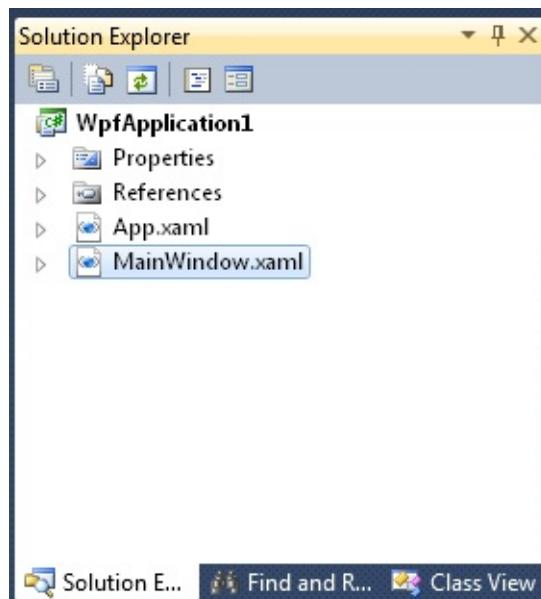


## AX.27 Changing Auto-Hide Behavior for Tool Windows



<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Environment   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0035

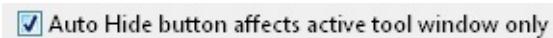
By default, you could easily have several tool windows in a tab group, as shown here:



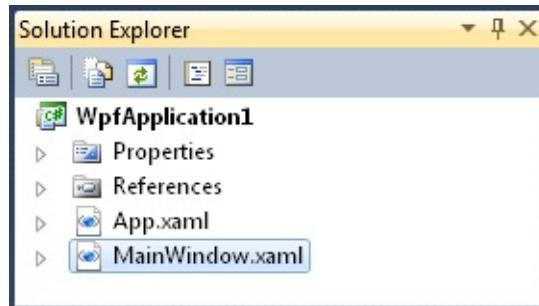
So to clear your workspace, you click the auto-hide button:



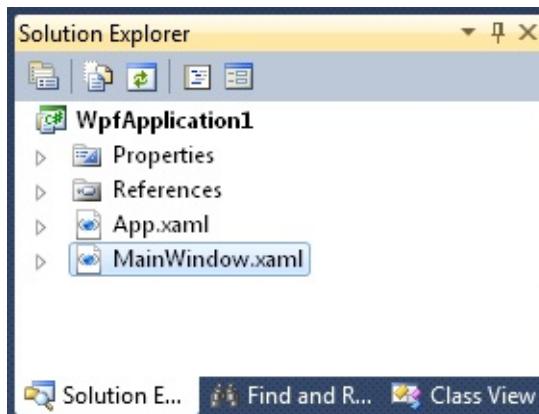
Now all the tool windows are hidden together. Next you can go to Tools | Options | Environment | General and select the Auto Hide Button Affects Active Tool Window Only option:



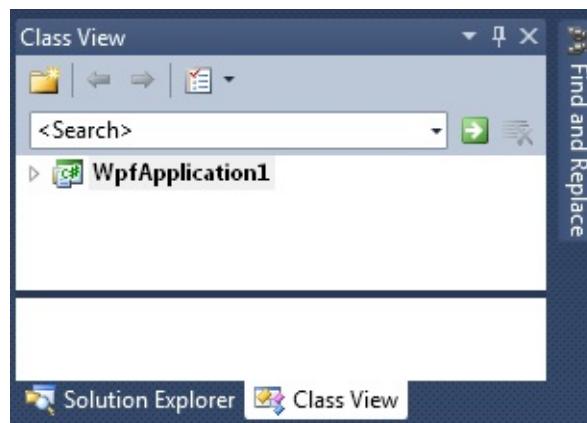
Now you no longer get tabs at the bottom when you interact with a tool window:



The tool windows have to be docked individually to get the tabs again:



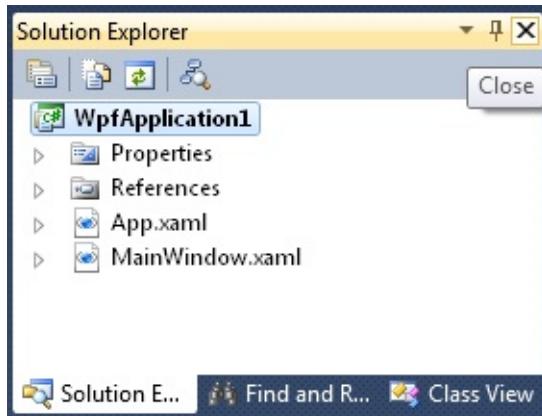
Now, when you auto-hide a window, only that window is hidden; the rest of the tab group remains visible:



## AX.28 Closing a Tool Window Tab Group

<b>WINDOWS</b>	Shift,Esc
<b>MENU</b>	Tools   Options   Environment   General
<b>COMMAND</b>	Window.CloseToolWindow
<b>VERSIONS</b>	2005, 2008, 2010

By default, when you click the Close button for a tool window in a tab group, it normally closes only the current tool window:



However, you can change this behavior by going to Tools | Options | Environment | General and clearing the Close Button Affects Active Tool Window Only check box, as shown in the following illustration:

Close button affects active tool window only

Now, closing any one tool window in the tab group causes all tool windows in that group to be closed.

## AX.29 Copy and Paste with the Command Prompt

WINDOWS	Enter (copy)
MENU	System Menu   Edit   [Mark, Copy, Paste, Select All]
CODE	vstipTool0057

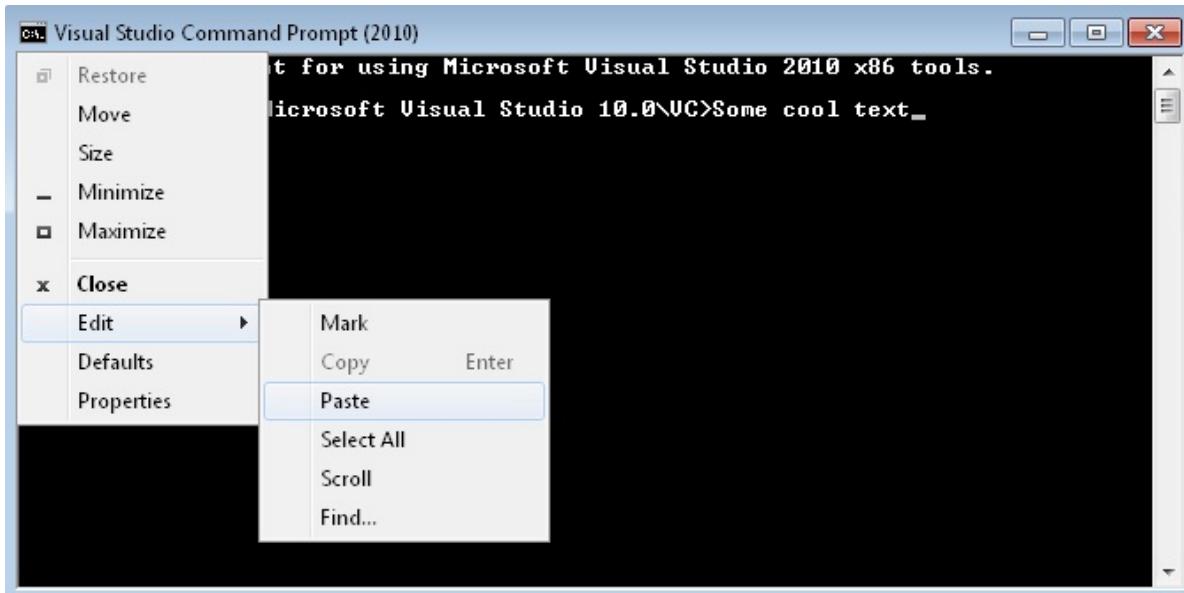
In a previous tip, we looked at how to work with the command prompt history. In this tip, we examine how to copy and paste text to and from the Command Prompt window.

### Pasting

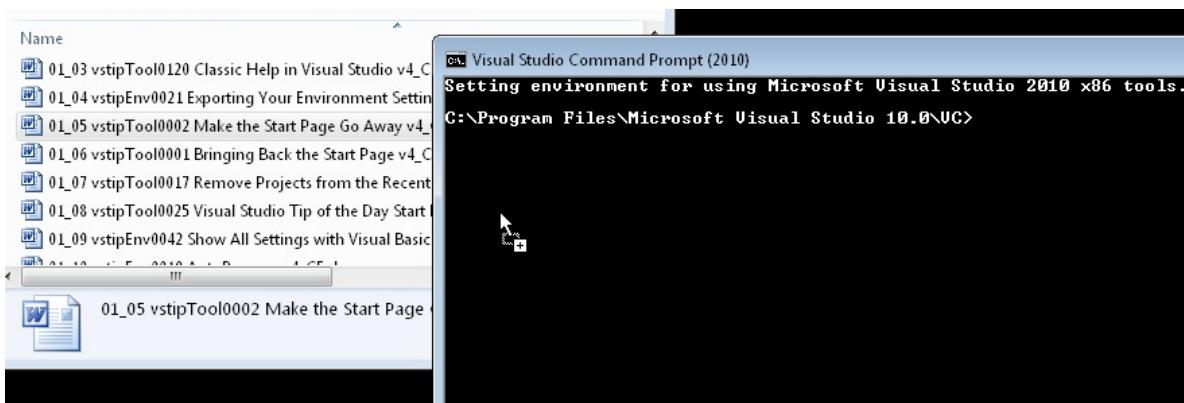
Visual Studio offers a rich set of tools to help you work with text in the command prompt windows. For example, you can copy text and paste it by going to System Menu | Edit | Paste, as shown in the following illustration.

## NOTE

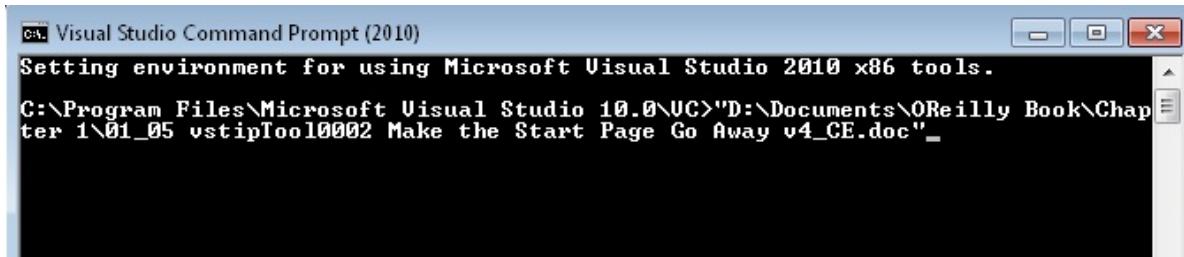
The System menu is the icon in the upper-left corner of the window.



If you have a folder or file path you want to use, you can always find the folder or file in Windows Explorer and drag the item to the command prompt, as shown in the following illustration:

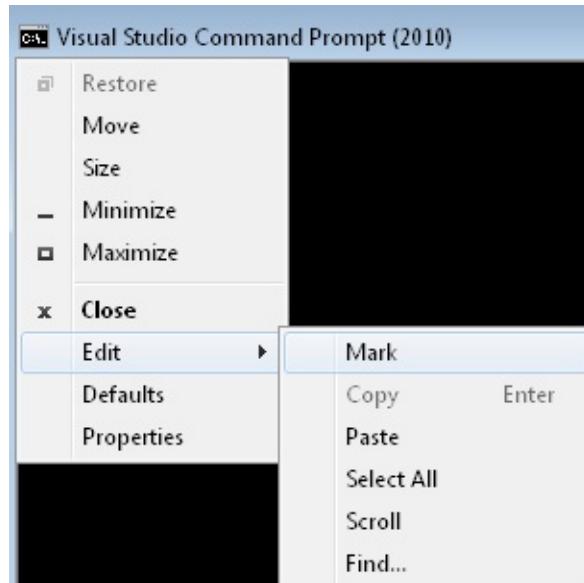


The preceding action pastes the full path to the folder or file in the window:

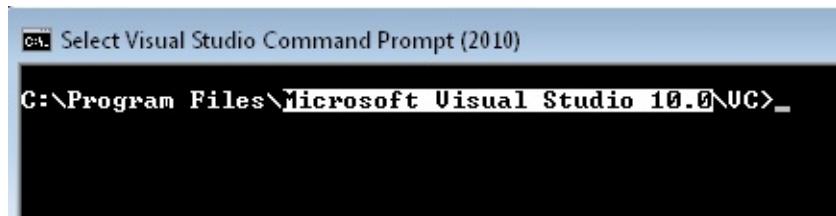


## Copying

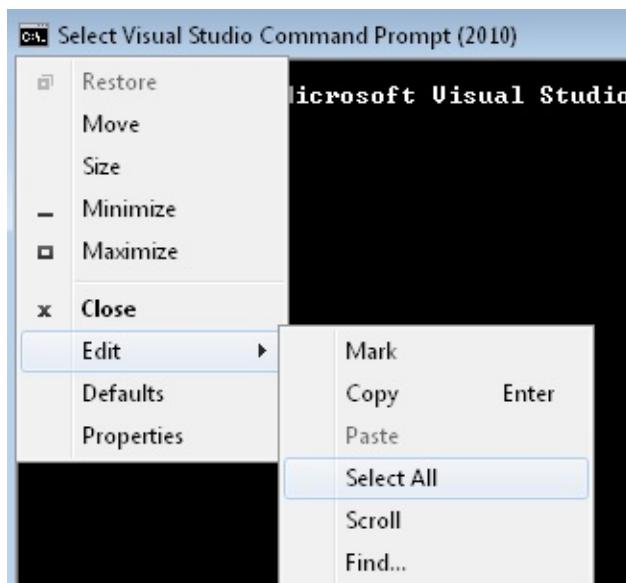
You can copy text from the command prompt by marking the text. Go to System Menu | Edit | Mark, as shown in the following illustration:



Highlight the text you want by clicking and dragging over that text:



If you want, you can always go to System Menu | Edit | Select All:

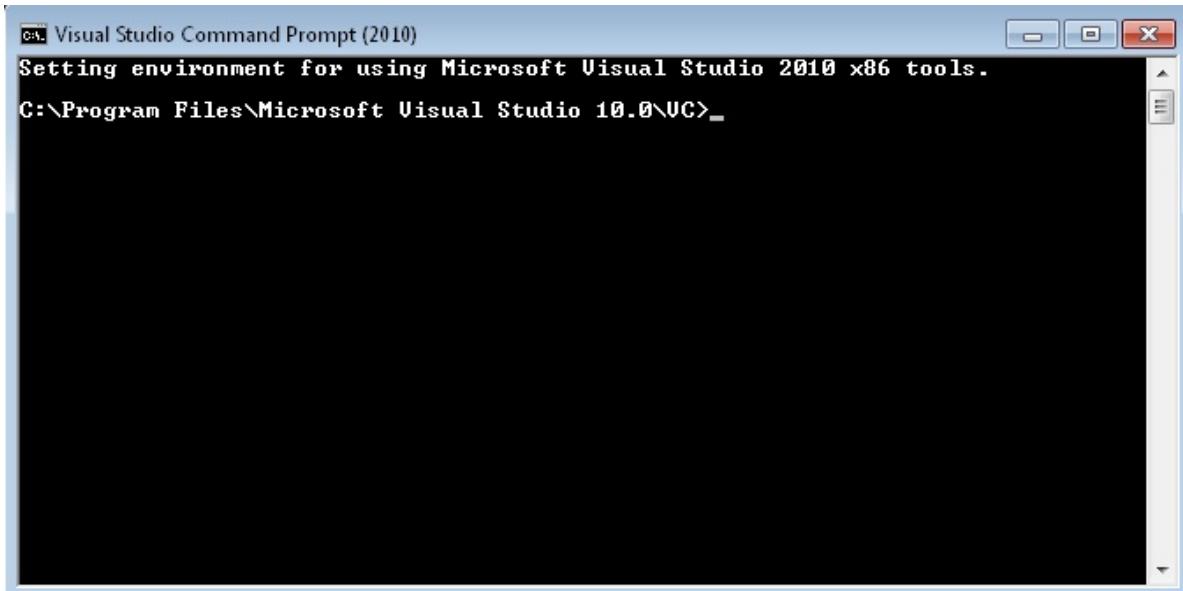


Press Enter to copy the text to the clipboard, and then paste the text where you want.

## AX.30 Customize the Command Prompt

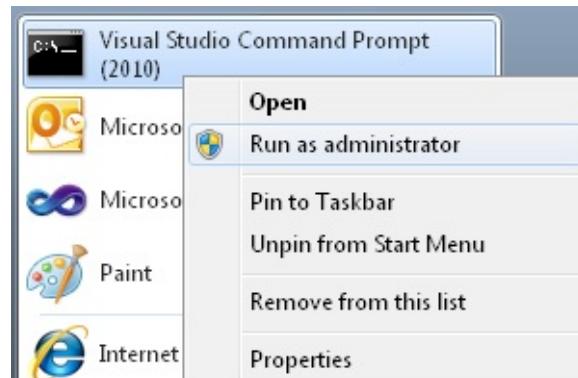
WINDOWS	Alt,Space, P
MENU	System Menu   Properties
CODE	vstipTool0058

If you are going to use the command prompt, you might want to customize the look and feel:

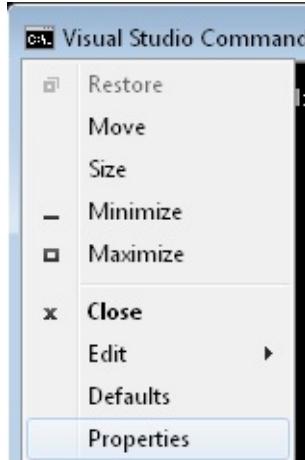


### NOTE

If you want to customize the Command window, you need to run it as Administrator. To do this, just right-click the command prompt icon and choose Run As Administrator:



Let's review some basic settings you might want to take advantage of early on. First go to System Menu | Properties:



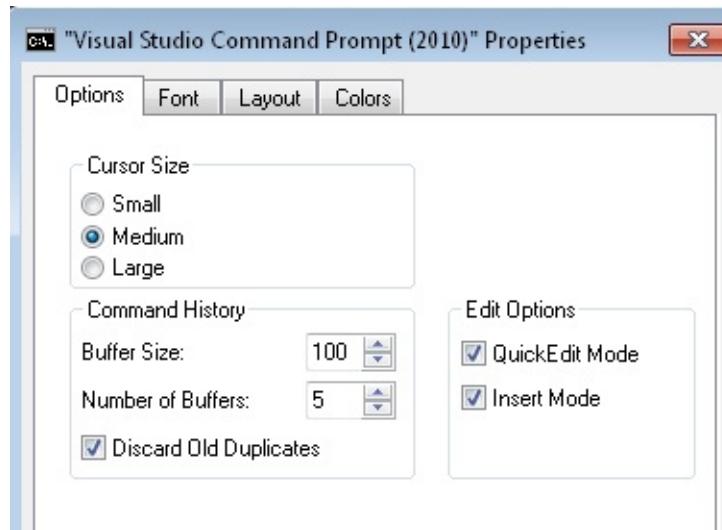
#### NOTE

Just as a reminder, the System menu is the icon at the upper-left corner of the window:



## Options

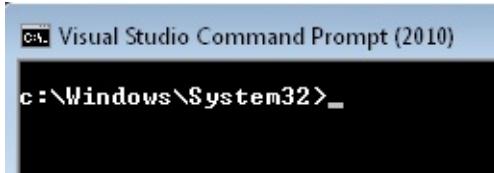
On the Options tab, notice that we have several items of interest:



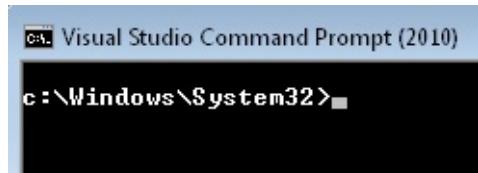
### Cursor Size

Represents the size of the flashing cursor.

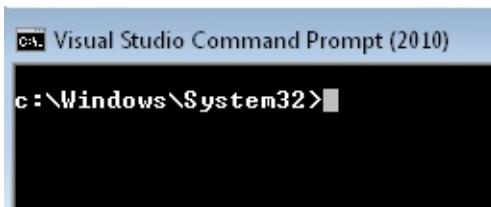
### Small



### Medium



### Large

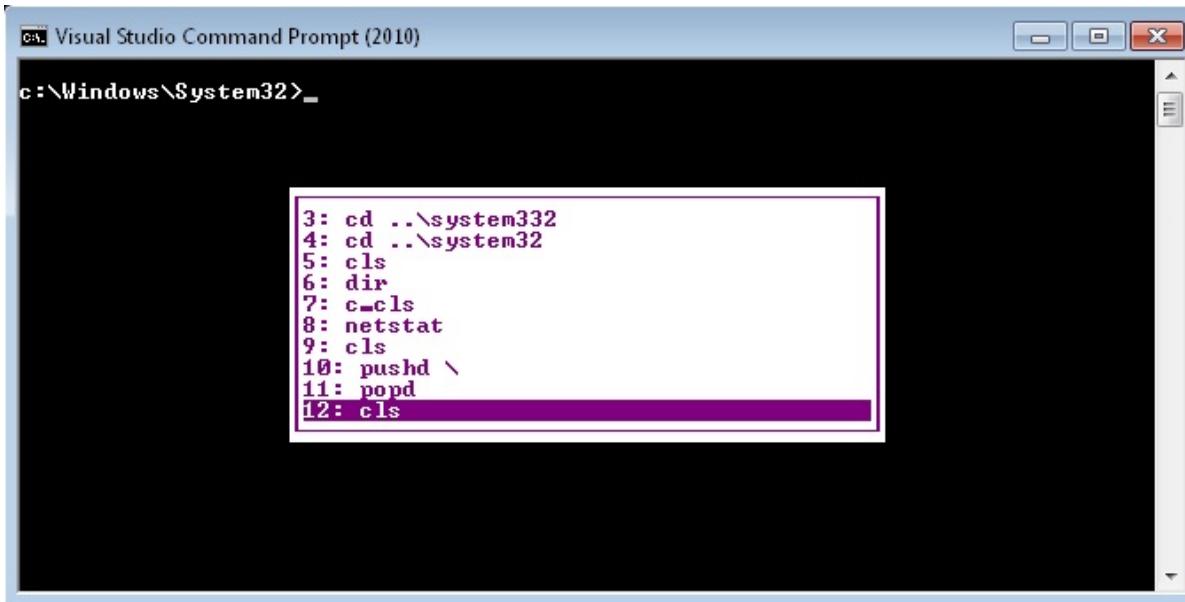


## Command History

Deals with how many commands are remembered by setting the buffer size (number of commands to remember) times the number of buffers. The default is 200 commands, but this value can be changed to 999 for each value for a max total of 998,001 commands that can be remembered. I usually set Buffer Size to 100 and Number Of Buffers to 5, for a total of 500 commands remembered.

I usually select the Discard Old Duplicates option. Some people leave this off for running the same command several times, so you might want to leave this off in some cases. Selecting this option removes duplicates so that they don't show up several times in the command history.

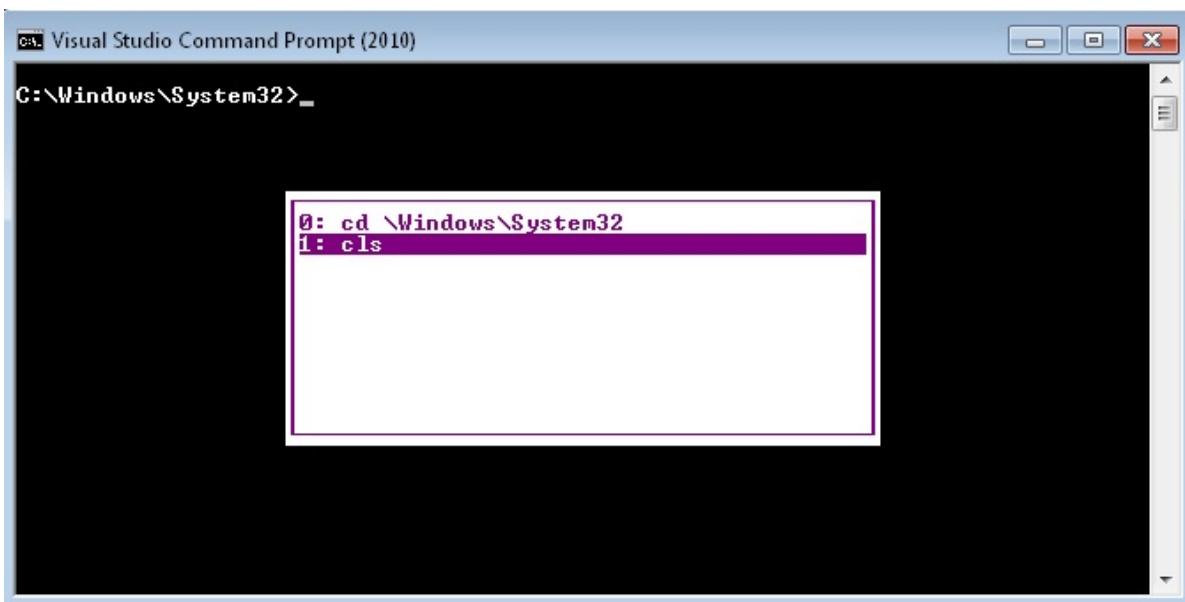
Before:



A screenshot of a Visual Studio Command Prompt window titled "Visual Studio Command Prompt (2010)". The window shows a command history buffer with the following entries:

```
c:\Windows\System32>_
3: cd ..\system32
4: cd ..\system32
5: cls
6: dir
7: c_mcls
8: netstat
9: cls
10: pushd \
11: popd
12: cls
```

After:



A screenshot of a Visual Studio Command Prompt window titled "Visual Studio Command Prompt (2010)". The window shows a command history buffer with the following entries:

```
C:\Windows\System32>_
0: cd \Windows\System32
1: cls
```

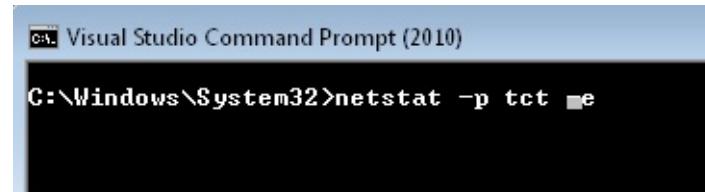
## Edit Options

Define how you can do certain actions. Quick Edit Mode allows you to use your mouse to work with text in the window, so I suggest you turn it on. Insert Mode is on by default and simply means that if you move the cursor into a command (using the arrow keys), you can type and it inserts (instead of overwrites) the text after it.

Before:

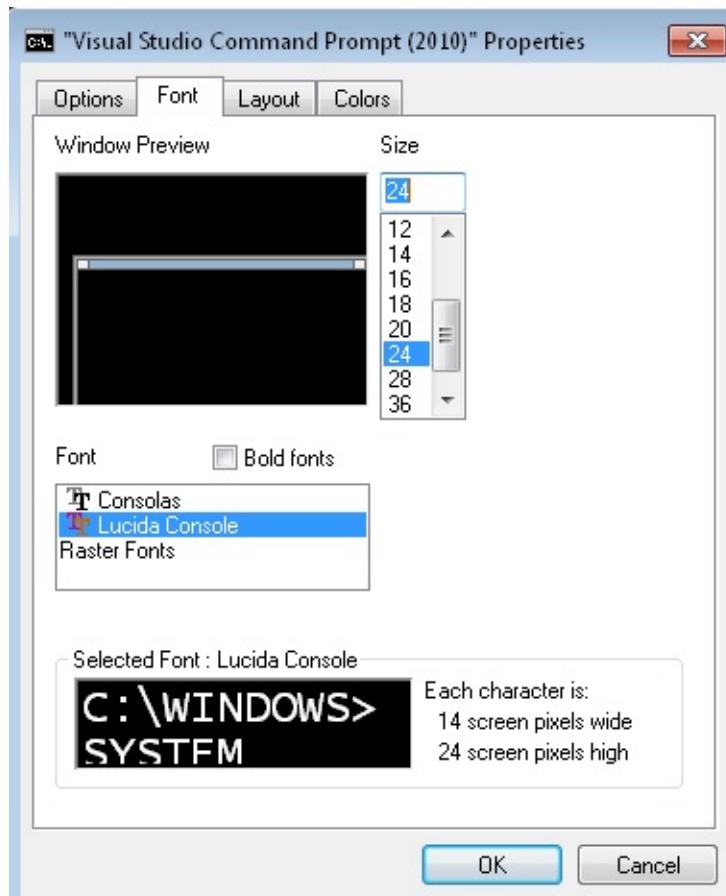


After:



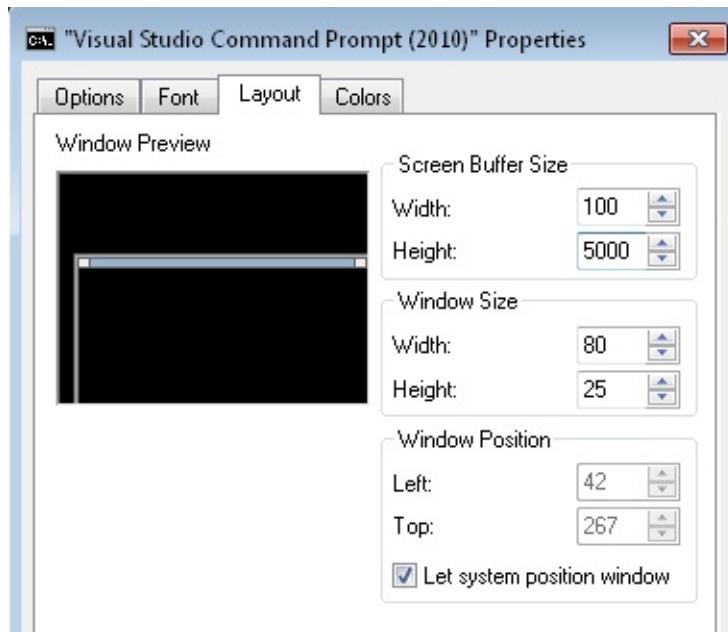
## Font

The Font tab is pretty self-explanatory. It allows you to pick various font properties for the command window. Play with these to suit your needs:



## Layout

The Layout tab is very important. Let's look at some of these options:



## Screen Buffer Size

Allows you to set how wide and tall you want the window information to be. Width should be less than the window size width you anticipate. Because I tend to maximize my command windows, I set this to 100. Height is how many lines you want to be able to scroll back to. I set this to 5000 just because I like large buffer sizes. Most folks tend to set this to around 1000 or so.

## Window Size

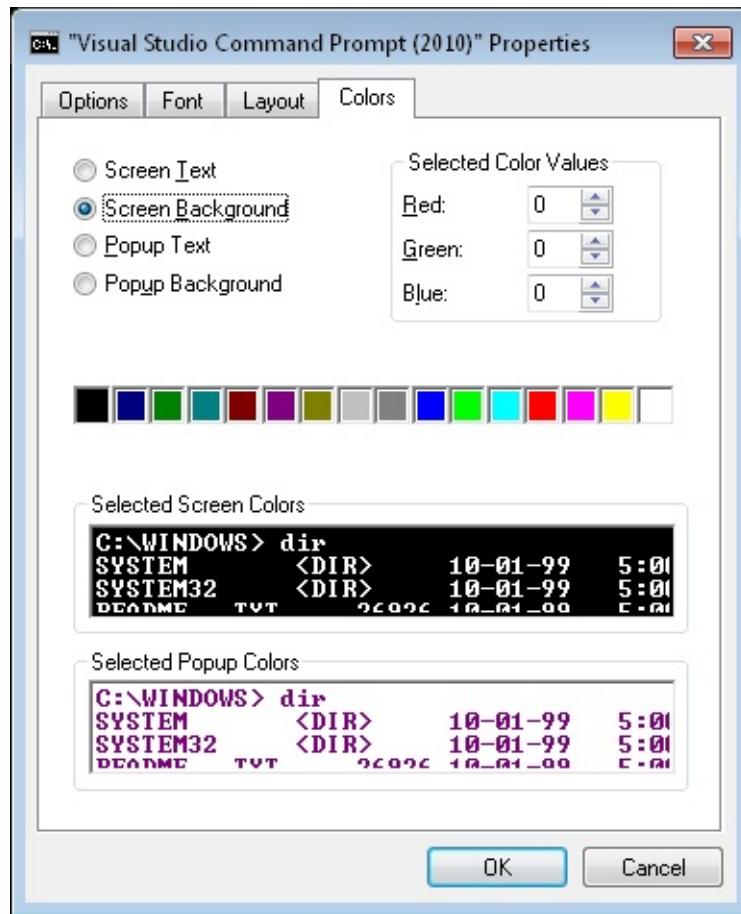
Sets the initial size of the command window. You can play with these settings to find a suitable size for you. Because I maximize my command windows, I just leave this setting alone.

## Window Position

The position you want the window to start in. I would leave this setting be, but if you want to change it, set Left and Top to 0 and then increase from there to suit your taste.

## Colors

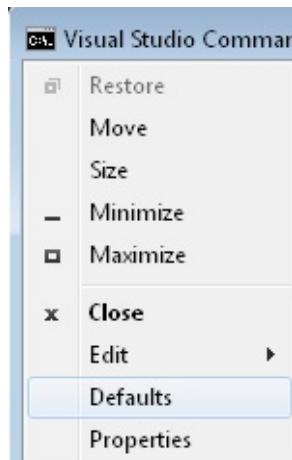
The Colors tab allows you to change two major aspects of the color scheme for the command window: one for the regular window and one for the pop-up window you get when you press F7:



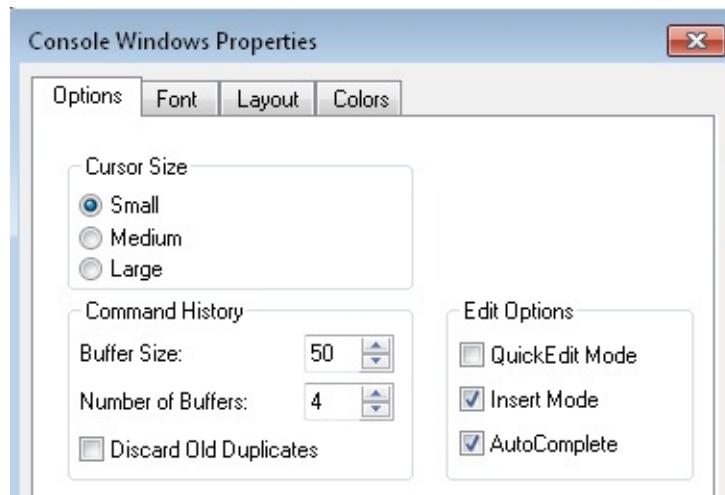
These settings are completely personal preferences, so try adjusting them and looking at the previews at the bottom to determine what schemes suit you. I tend to favor the old school “green screen” colors on my Command windows (green text with a black background).

## Resetting Back to Defaults

If you totally mess up your settings, you can always get all the default settings back by going to System Menu | Defaults:



After clicking Default, you see the properties with *all tabs set to their default settings*. Just click OK, and you have all your defaults back:



#### NOTE

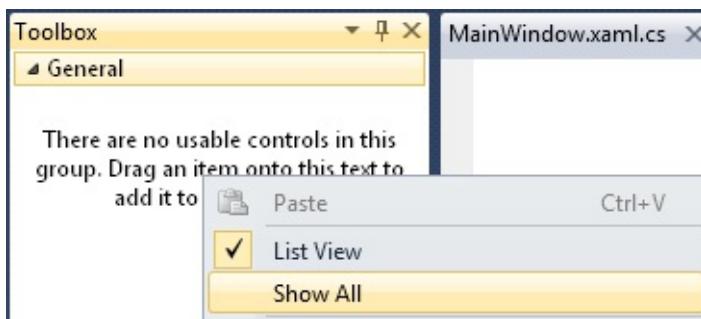
When you restore the default settings, notice the AutoComplete option that wasn't enabled before. This option allows for tab completion of file names. (See vstipTool0056, [03.17 Command Prompt Tab Completion](#), page 105.) You should always leave this setting on.

## AX.31 Show All Toolbox Controls

<b>WINDOWS</b>	Shift,F10, S (with Toolbox selected)
<b>MENU</b>	Right-click   Show All
<b>COMMAND</b>	Tools.ShowAll
<b>VERSIONS</b>	2005, 2008, 2010

Sometimes when you install controls for the Toolbox, you might not see the controls because they are in the wrong context. For example, a WPF control wouldn't show up in the Toolbox while you are typing code. When the controls don't show up, you might think the add-in failed to install the Toolbox controls. This tip shows how you can confirm everything installed in the Toolbox.

To see all the controls that are installed, right-click the Toolbox and select Show All, as shown in the following illustration:



Now all the controls are visible, regardless of context:

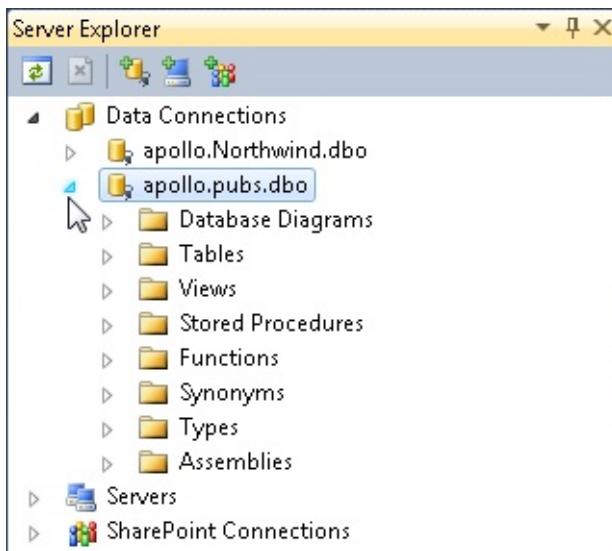


## AX.32 Server Explorer: Data Connections

<b>DEFAULT</b>	Ctrl+Alt+S (view server explorer)
<b>VISUAL BASIC 6</b>	Ctrl+Alt+S (view server explorer)
<b>VISUAL C# 2005</b>	Ctrl+Alt+S; Ctrl+W, L; Ctrl+W, Ctrl+L (view server explorer)
<b>VISUAL C++ 2</b>	Ctrl+Alt+S (view server explorer)
<b>VISUAL C++ 6</b>	Ctrl+Alt+S (view server explorer)
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+S (view server explorer)

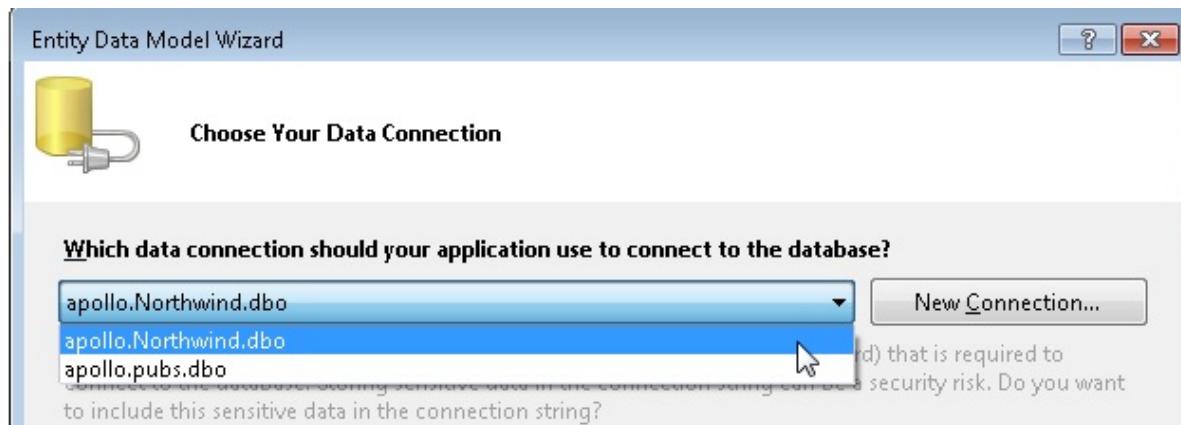
<b>WINDOWS</b>	Alt,V, V (view server explorer); Alt,T, D (connect to database)
<b>MENU</b>	View   Server Explorer; Tools   Connect to Database
<b>COMMAND</b>	View.ServerExplorer; Tools.ConnecttoDatabase
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0121

Server Explorer (Ctrl+Alt+S) is the server management tool window that comes with Visual Studio. One of the things you can use this window for is to open data connections:



## Data Connections in Other Areas

The data connections you have in Server Explorer can be used in other areas, such as ADO.NET Entity Data Models:



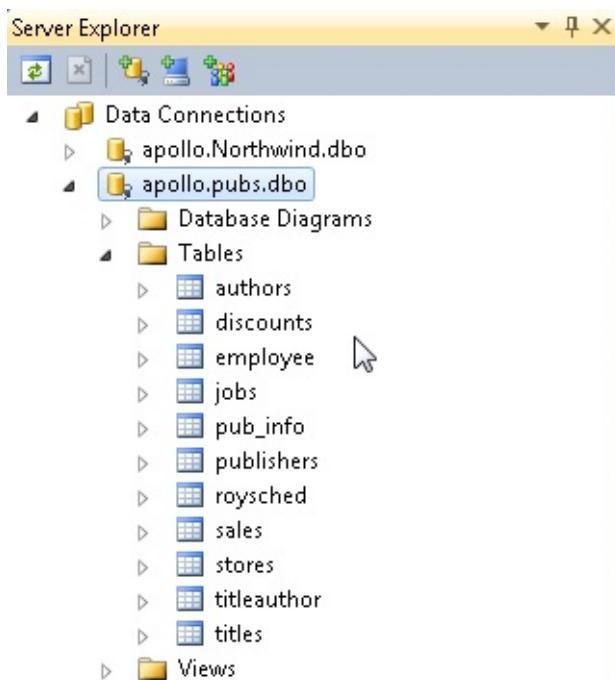
When you go to create Entity Data Models that are generated from a database, you

can choose from the data connections you already have or you can create a new connection to be added to the list.

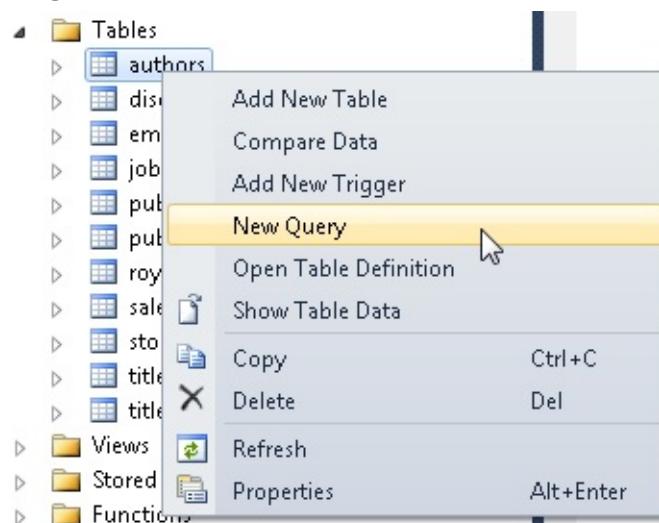
## Data Connections in Server Explorer

There is an incredible amount of power and control that is available here. Let's look at a couple of examples.

### Tables

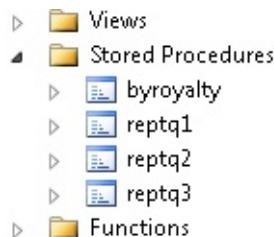


At the most basic level, you can list out the tables in the database. Additionally, you can right-click any table and see most of the features you can leverage, as shown in the following illustration:

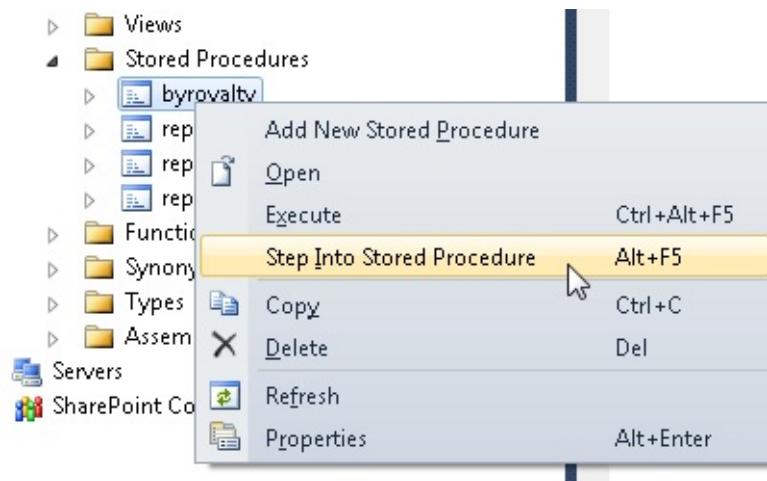


Notice that you can add a new table, create a new query, or just show the data in the table, among other tasks.

## Stored Procedures



One of the great database features in Visual Studio is that not only can you create and edit stored procedures, but you can step into them as well:



## Finally

Don't take the Database Connection features in Server Explorer for granted. These features make a great deal of power available to you, and you should invest some time to see whether these are helpful for your work.

## AX.33 Server Explorer: Server Event Logs

<b>DEFAULT</b>	Ctrl+Alt+S
<b>VISUAL BASIC 6</b>	Ctrl+Alt+S
<b>VISUAL C# 2005</b>	Ctrl+Alt+S; Ctrl+W, L; Ctrl+W, Ctrl+L
<b>VISUAL C++ 2</b>	Ctrl+Alt+S
<b>VISUAL C++ 6</b>	Ctrl+Alt+S

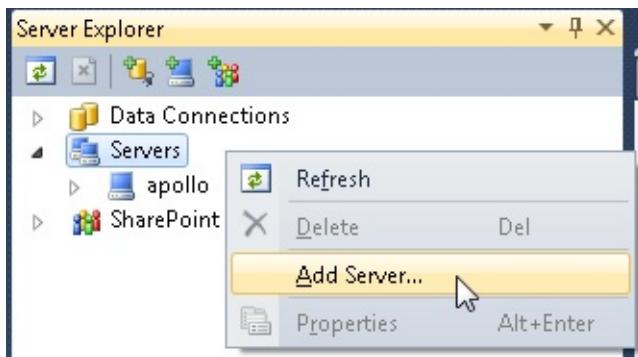
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+S
<b>WINDOWS</b>	Alt,V, V
<b>MENU</b>	View   Server Explorer
<b>COMMAND</b>	View.ServerExplorer
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0122

Most people believe that Server Explorer is the tool window we use for data connections (see vstipTool0121, [AX.32 Server Explorer: Data Connections](#), page A58), and they assume that's pretty much where the experience ends.

This unsung hero really does a lot more if you let it. For example, the Servers section comes with a power that you might not even know existed—working with servers.

## Adding Servers

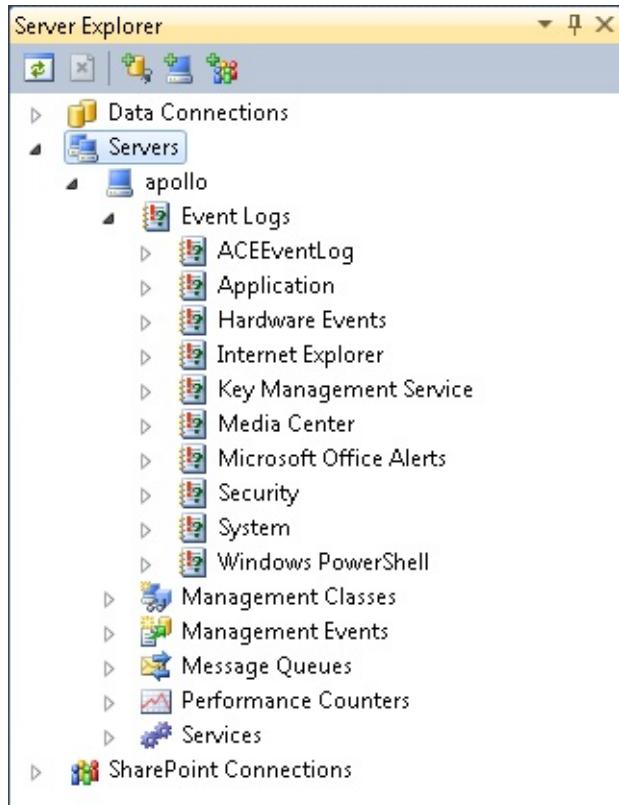
To get started, Server Explorer comes with the local machine already in the list of Servers, plus you can add additional servers as needed:



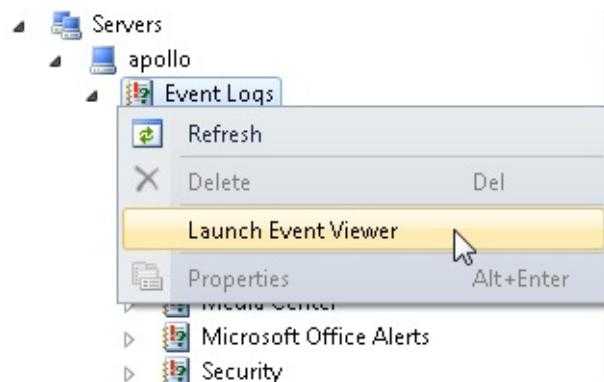
Don't be fooled by the term "Server"; it really means any computer that you want to connect to for information. In these examples, my "server" is a Microsoft Windows 7 Professional-based machine.

## Event Logs

After you have the server that you want, you can start working with features. The Event Logs section is a perfect example. The following illustration shows what is on my machine for Event Logs in Server Explorer:



You can use this area to start the Event Viewer if you need it:



Or you can take a quick view of events in any of the various categories by drilling down into the tree:

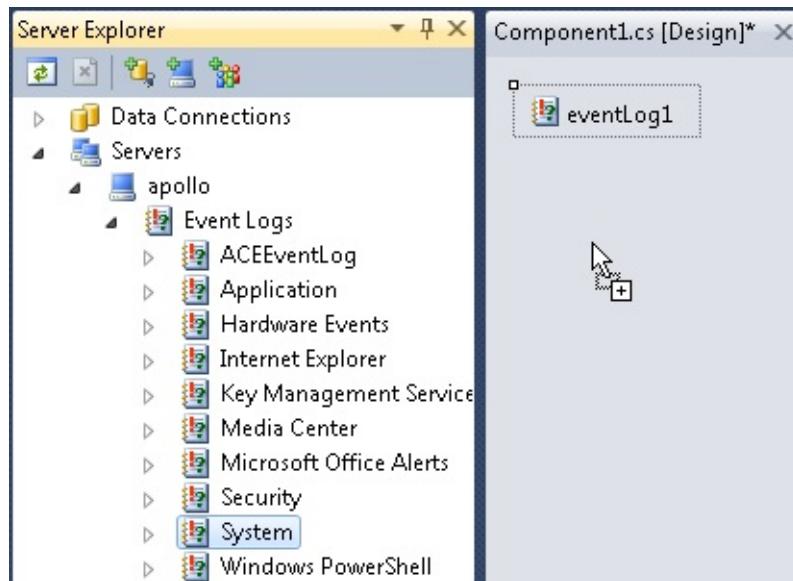


## Click and Drag Components

Event log components can even be dragged onto a Windows form or component class design surface so that you can manipulate them:

**NOTE**

For more information, see the topic “How to: Add Items from Server Explorer,” at <http://msdn.microsoft.com/en-us/library/84s2c1k0.aspx>.



When the component is in place, you can write code to perform various actions. In the following example, an entry is written to the event log:

```
public void WriteToLog()
{
    eventLog1.WriteEntry("Log entry!");
}
```

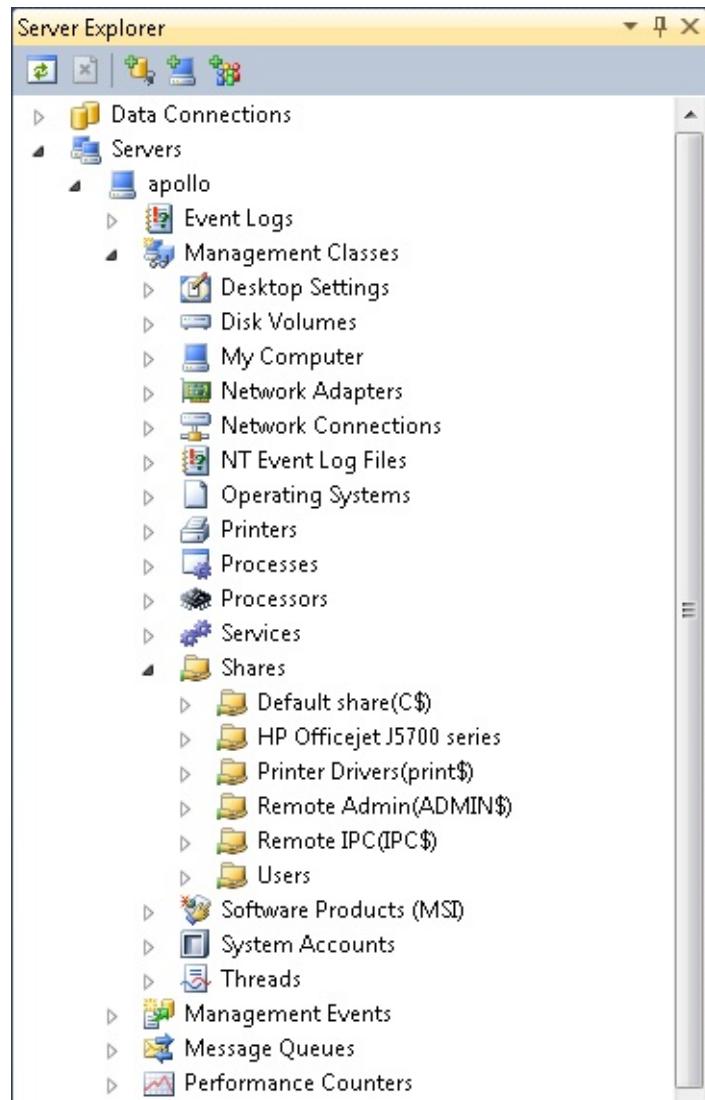
This is a great way to have event log information available to the user. Taking time to learn Server Explorer can definitely bring you benefits in your daily work.

## AX.34 Server Explorer: Server Management Classes

<b>DEFAULT</b>	Ctrl+Alt+S
<b>VISUAL BASIC 6</b>	Ctrl+Alt+S

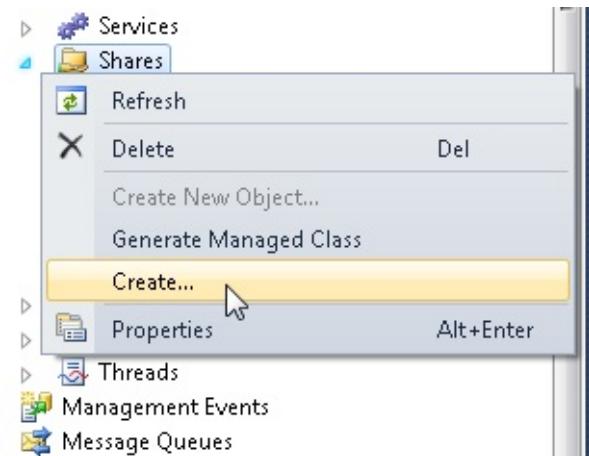
<b>VISUAL C# 2005</b>	Ctrl+Alt+S; Ctrl+W, L; Ctrl+W, Ctrl+L
<b>VISUAL C++ 2</b>	Ctrl+Alt+S
<b>VISUAL C++ 6</b>	Ctrl+Alt+S
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+S
<b>WINDOWS</b>	Alt,V, V
<b>MENU</b>	View   Server Explorer
<b>COMMAND</b>	View.ServerExplorer
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0123

The management classes in Server Explorer give you a view into the system. For example, you can use management classes to see network shares, as shown in the following illustration:

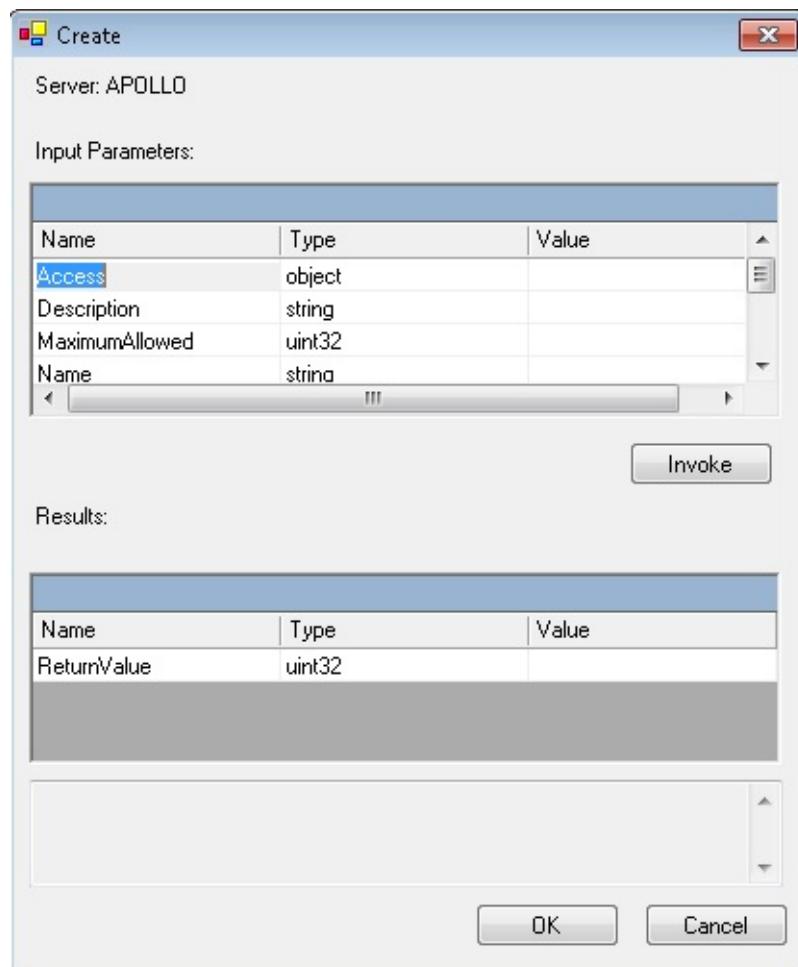


## Create

You can also right-click a node and choose Create from the menu to make a new instance:



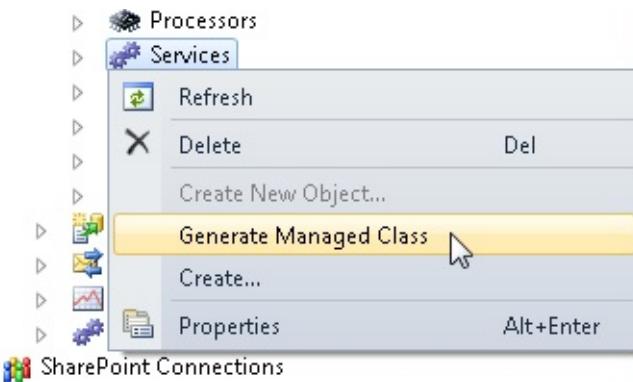
Unfortunately, you get a really ugly and unfriendly dialog box to use when you do this, as shown in the following illustration:



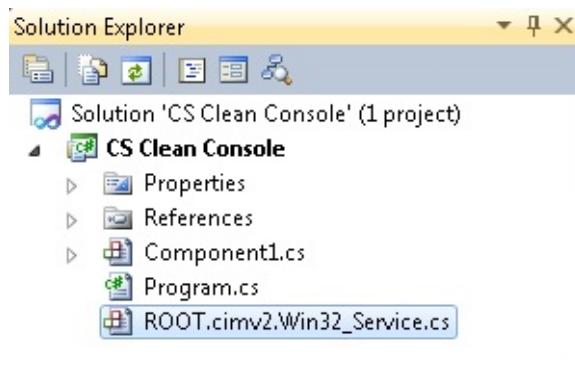
## Generate Classes

One better way to work with the management classes is to right-click one and then

generate a class from the menu:



This gives you a class you can use as you see fit:



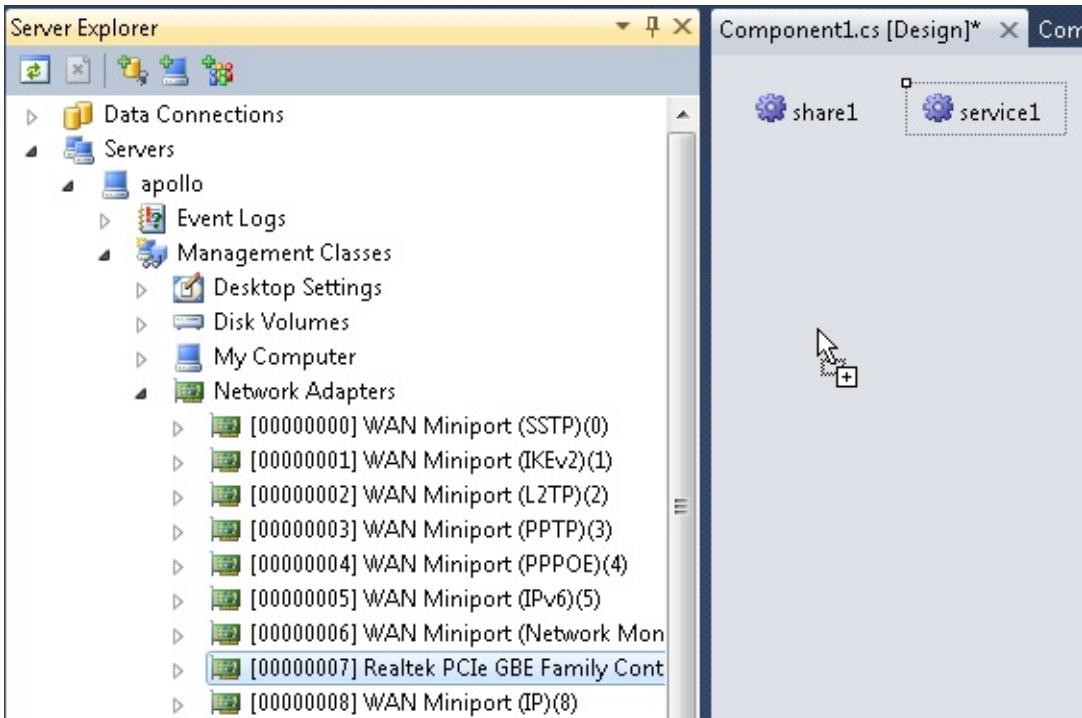
```
// Private property to hold the WMI namespace in which the class resides.  
private static string CreatedWmiNamespace = "ROOT\\cimv2";  
  
// Private property to hold the name of WMI class which created this class.  
private static string CreatedClassName = "Win32_Service";
```

## Click and Drag Components

The management classes can also be dragged onto a Windows form or component class design surface so that you can manipulate them.

### NOTE

For more information, see the topic “How to: Add Items from Server Explorer,” at <http://msdn.microsoft.com/en-us/library/84s2c1k0.aspx>.



## AX.35 Window Layouts: File View

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0053

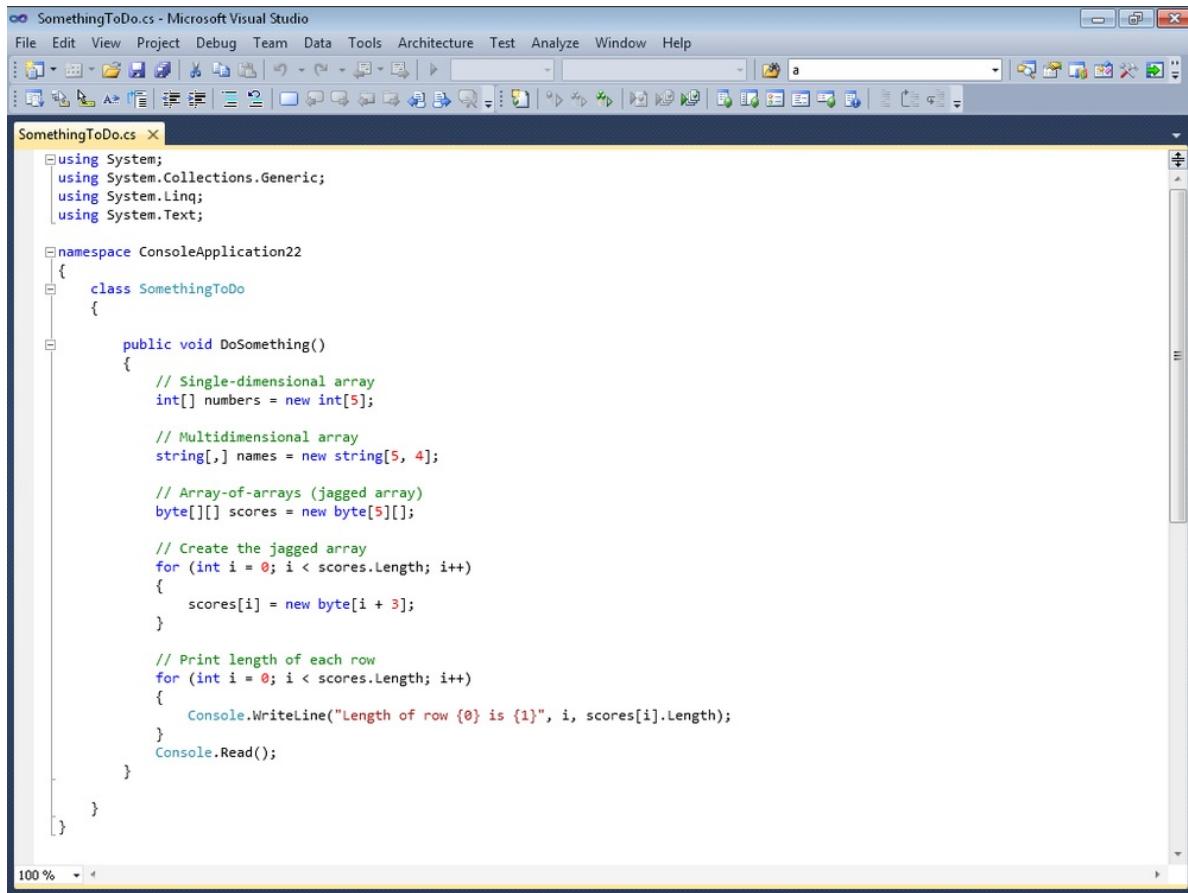
I talked about the different window layouts in vstipEnv0051, [03.08 Window Layouts: The Four Modes](#), on page 90, and vstipEnv0052, [03.09 Window Layouts: Design, Debug, and Full Screen](#), on page 91. Now I want to cover a lesser-known layout known as File View.

You get to File View by putting in the file name of any file associated with Visual Studio at the command prompt. In the following example, I've changed my directory to one of my solutions, and I'm putting in the name "SomethingToDo.cs":

```
Visual Studio Command Prompt (2010)
d:\Documents\Visual Studio 2010\Projects\ConsoleApplication22>SomethingToDo.cs
d:\Documents\Visual Studio 2010\Projects\ConsoleApplication22>
```

When I tried to do this without running the command prompt as Administrator, it wouldn't load the file. So you might have to run the command prompt as

Administrator. If Visual Studio is already open, it shows the file and if not, it loads and shows the file:



The screenshot shows the Microsoft Visual Studio interface with the title bar "SomethingToDo.cs - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, and Help. Below the menu is a toolbar with various icons. The main window displays the code for "SomethingToDo.cs". The code uses namespaces System, System.Collections.Generic, System.Linq, and System.Text. It defines a class SomethingToDo with a method DoSomething(). This method demonstrates different array types: a single-dimensional array of integers, a multidimensional string array, and a jagged byte array. It prints the length of each row of the jagged array to the console.

```
SomethingToDo.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication22
{
    class SomethingToDo
    {
        public void DoSomething()
        {
            // Single-dimensional array
            int[] numbers = new int[5];

            // Multidimensional array
            string[,] names = new string[5, 4];

            // Array-of-arrays (jagged array)
            byte[][] scores = new byte[5][];

            // Create the jagged array
            for (int i = 0; i < scores.Length; i++)
            {
                scores[i] = new byte[i + 3];
            }

            // Print length of each row
            for (int i = 0; i < scores.Length; i++)
            {
                Console.WriteLine("Length of row {0} is {1}", i, scores[i].Length);
            }
            Console.Read();
        }
    }
}
```

Notice that it is a very sparse layout. The good news is that you can customize it any way you want, to have quick access to the tools you need most. I'll add Server Explorer for now:

The screenshot shows the Microsoft Visual Studio interface. The title bar reads "SomethingToDo.cs - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, Help. The toolbar has various icons for file operations like Open, Save, Print, etc. The left sidebar contains the "Server Explorer" and "Solution Explorer" panes. The "Server Explorer" pane shows "Data Connections" with "PUBS.MDF" selected. The "Solution Explorer" pane shows a project named "SomethingToDo.cs" with files "SomethingToDo.cs" and "Program.cs". The main code editor window displays the following C# code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication22
{
    class SomethingToDo
    {
        public void DoSomething()
        {
            // Single-dimensional array
            int[] numbers = new int[5];

            // Multidimensional array
            string[,] names = new string[5, 4];

            // Array-of-arrays (jagged array)
            byte[][] scores = new byte[5][];

            // Create the jagged array
            for (int i = 0; i < scores.Length; i++)
            {
                scores[i] = new byte[i + 3];
            }

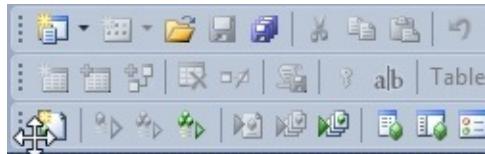
            // Print length of each row
            for (int i = 0; i < scores.Length; i++)
            {
                Console.WriteLine("Length of row {0} is {1}", i, scores[i].Length);
            }
            Console.Read();
        }
    }
}
```

If you make changes, they are saved to your .vssettings file when you close Visual Studio. Remember that each mode is a distinct area, so the changes you make here are not seen in any of the other window modes.

## AX.36 Rearrange Your Toolbars

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0027

You can easily rearrange your toolbars by simply putting your mouse over the grip control (the four vertical dots to the left of every toolbar), as shown in the following illustration:



When you get the four-directional mouse pointer, just click and drag the toolbar to the new position:



Be careful; it's real easy to get a shortened toolbar on your hands when moving toolbars beside other toolbars, as shown in the following illustration:



#### NOTE

You know that you aren't seeing all the buttons on a toolbar if there are two arrows side-by-side on the far right of that toolbar, as shown in the following illustration:



Of course, this shortened toolbar might be exactly what you want because the rest of the buttons are accessed by clicking on the drop-down arrow:



If this is not what you want, just click and drag the grip control of the toolbar to the right of the shortened toolbar and uncover as many buttons as you like:



In Visual Studio 2008 and 2005, you can also make the toolbar a floating one by dragging it off the toolbar area, as shown in the following illustration:

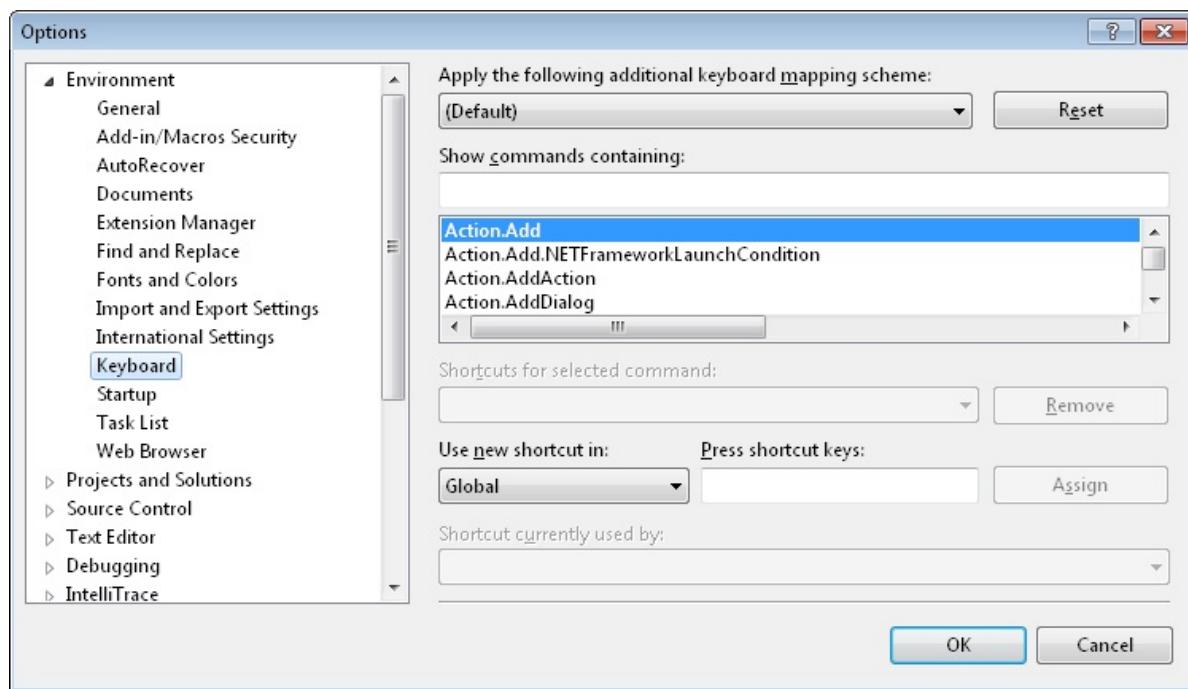


To put it back, just double-click anywhere on the title bar of the toolbar.

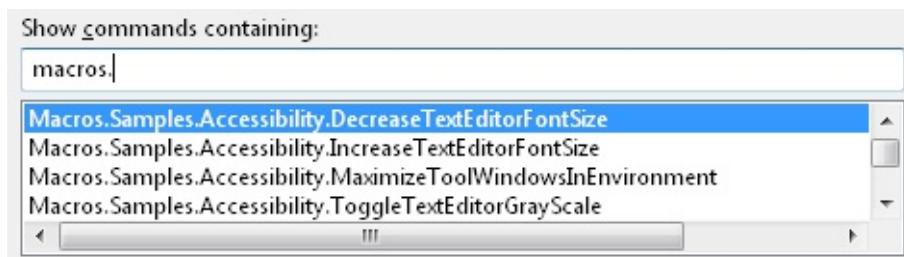
## AX.37 Create a Shortcut Key for a Macro

<b>WINDOWS</b>	Alt, T, O
<b>MENU</b>	Tools   Options   Environment   Keyboard
<b>COMMAND</b>	Tools.CustomizeKeyboard
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0066

We covered creating shortcuts in vstipTool0063 ([03.26 Keyboard Shortcuts: Creating New Shortcuts](#), page 127). Assuming you have a macro you would like to attach a shortcut key to, you can easily make your macros accessible. Let's look at an example to show you how. First, go to Tools | Options | Environment | Keyboard:



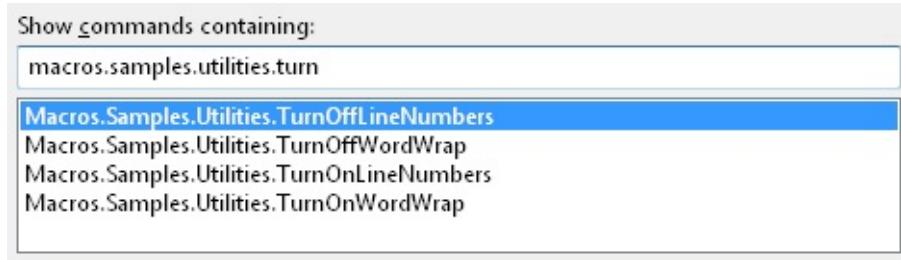
To access your macros, type **macros.** in the Show Commands Containing area:



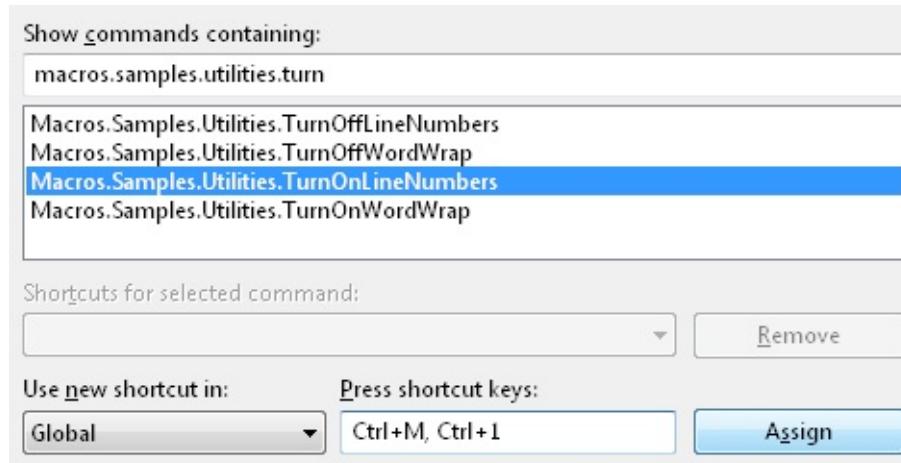
In this case, let's bind the Macros.Samples.Utilities.TurnOffLineNumbers and Macros.Samples.Utilities.TurnOnLineNumbers macros.

**WARNING**

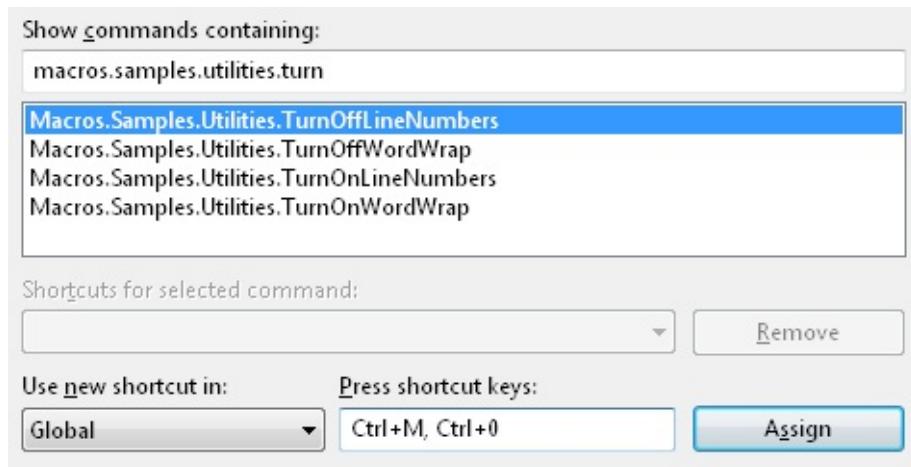
Make sure the keyboard shortcuts you use here aren't already assigned to something else. They most likely aren't, but you should double-check by reviewing vtipTool0063, [03.26 Keyboard Shortcuts: Creating New Shortcuts](#) page 127.



Let's bind Macros.Samples.Utilities.TurnOnLineNumbers to Ctrl+M, CTRL+1, as shown in the following illustration:



Let's bind Macros.Samples.Utilities.TurnOffLineNumbers to Ctrl+M, Ctrl+0 (zero), as shown in the following illustration:



Click OK, and let's test out our shortcuts. Go to any source code, and press Ctrl+M, Ctrl+0 to turn line numbers off and Ctrl+M, Ctrl+1 to turn them on.

## AX.38 How to Run External Executables from the Command Window

<b>COMMAND</b>	Tools.Shell
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0089

You can run external programs from the Command Window (Ctrl+Alt+A) by using the **Tools.Shell** command. This can be useful if you run certain executables often (like Xcopy) and want to turn the action into a command alias. See vstipTool0068 ([03.20 Understanding Commands: Aliases](#), page 113).

The general syntax for the command is as follows:

```
Tools.Shell [/command] [/output] [/dir:folder] path [args]
```

### Arguments

Following are some key arguments you can use for this command:

- **/commandwindow, /command, /c, or /cmd**

Optional. Specifies that the output for the executable is displayed in the Command Window.

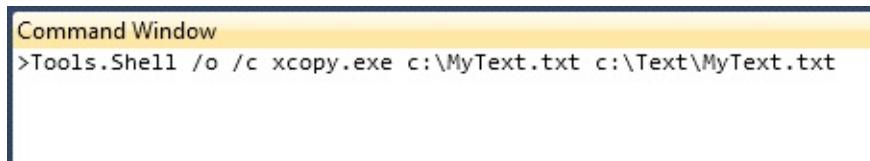
- **/dir:folder or /d: folder**

Optional. Specifies the working directory to be set when the program is run.

- **/outputwindow, /output, /out, or /o**

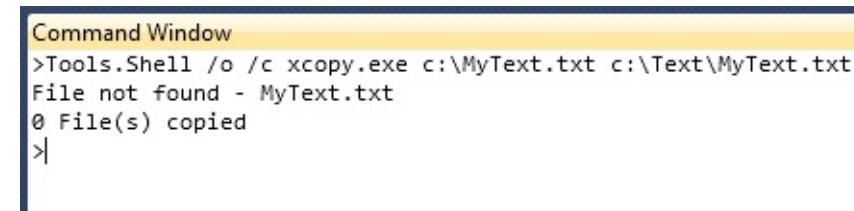
Optional. Specifies that the output for the executable is displayed in the Output window.

For example, to run the Xcopy command, it would appear as shown in the following illustration:



```
Command Window
>Tools.Shell /o /c xcopy.exe c:\MyText.txt c:\Text\MyText.txt
```

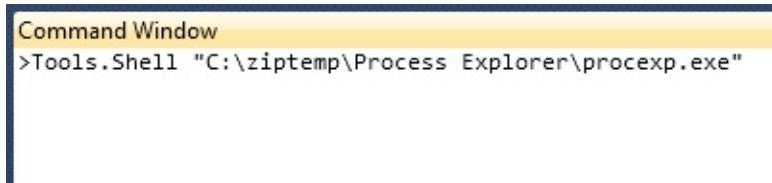
Just for fun, because I don't actually have the files in the sample command, the output shown in the following illustration is the result of running the Xcopy command:



```
Command Window
>Tools.Shell /o /c xcopy.exe c:\MyText.txt c:\Text\MyText.txt
File not found - MyText.txt
0 File(s) copied
>
```

As you can see, the output from Xcopy was redirected to the Command Window and tells me that it can't find a file.

If the item you want to run isn't in the path environment variable, you must include the full path (surrounded by quotes if any spaces are in the path), as shown in the following illustration:



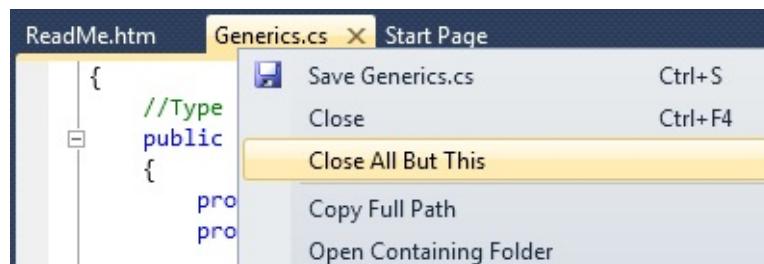
```
Command Window
>Tools.Shell "C:\ziptemp\Process Explorer\procexp.exe"
```

## Additional Tips from Chapter 4

### AX.39 Close All But This on the File Tab Channel

<b>DEFAULT</b>	Alt+- (minus sign), A [VS2010 Only]
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	Alt+- (minus sign), A [VS2010 Only]
<b>VISUAL C++ 2</b>	Alt+- (minus sign), A [VS2010 Only]
<b>VISUAL C++ 6</b>	Alt+- (minus sign), A [VS2010 Only]
<b>VISUAL STUDIO 6</b>	Alt+- (minus sign), A [VS2010 Only]
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	File.CloseAllButThis; Window.ShowDockMenu
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0011

If you ever get the urge to close every file except the one you are currently working on, you can just right-click the current file tab and choose Close All But This:

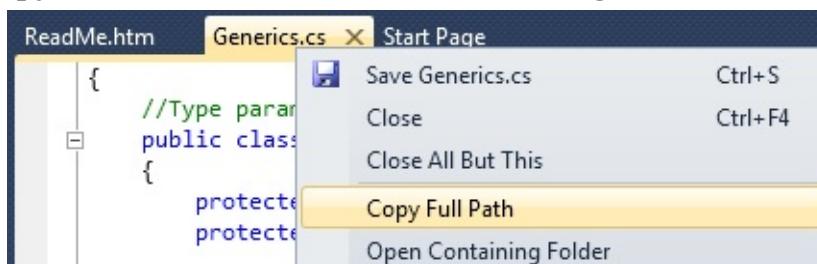


### AX.40 Copy a File's Full Path from the File Tab

<b>DEFAULT</b>	Alt+- (minus sign), F [VS2010 Only]
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	Alt+- (minus sign), F [VS2010 Only]
<b>VISUAL C++ 2</b>	Alt+- (minus sign), F [VS2010 Only]

<b>VISUAL C++ 6</b>	Alt+- (minus sign), F [VS2010 Only]
<b>VISUAL STUDIO 6</b>	Alt+- (minus sign), F [VS2010 Only]
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	File.CopyFullPath; Window.ShowDockMenu
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0013

You can quickly copy the full path of any file. Just right-click the tab for the file, and choose Copy Full Path, as shown in the following illustration:



You now have the full path in your clipboard so that you can paste it (Ctrl+V) anywhere you need it.

## AX.41 Understanding the File Tab Channel Drop-Down Button

<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0012

Here is a convenient way to know whether you are seeing all your files on the File Tab Channel.

Let's say you have a couple of files open: Notice that the button to the far right looks like an arrow pointing down:



But if you open up a few more files so that they all can't be seen on the file channel, the button changes to let you know that all the files are not visible on the File Tab Channel (indicated by the line above the arrow in the following illustration):

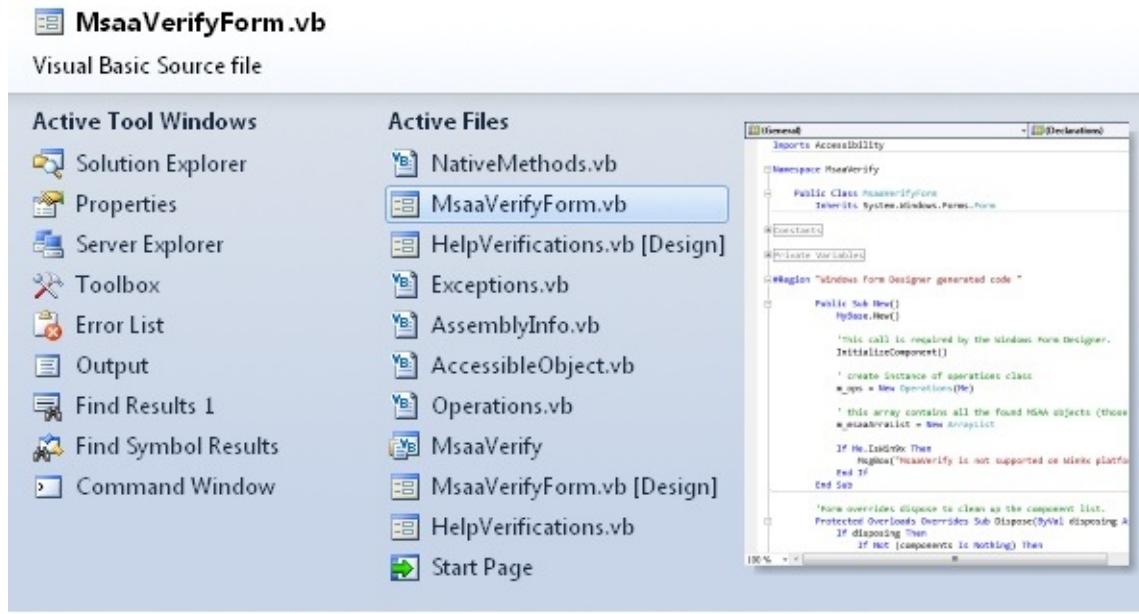


If you click the button, it gives you a list of all the files so that you can pick the one you want from the list.

## AX.42 How to Disable the IDE Navigator

<b>DEFAULT</b>	Ctrl+F6 (next); Ctrl+Shift+F6 (previous)
<b>VISUAL BASIC 6</b>	Ctrl+F6 (next); Ctrl+Shift+F6 (previous)
<b>VISUAL C# 2005</b>	Ctrl+F6 (next); Ctrl+Shift+F6 (previous)
<b>VISUAL C++ 2</b>	Ctrl+F6; Ctrl+Tab (next); Ctrl+Shift+F6; Ctrl+Shift+Tab (previous)
<b>VISUAL C++ 6</b>	Ctrl+F6 (next); Ctrl+Shift+F6 (previous)
<b>VISUAL STUDIO 6</b>	Ctrl+F6; Ctrl+Tab (next); Ctrl+Shift+F6; Ctrl+Shift+Tab (previous)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Window.NextDocumentWindow; Window.PreviousDocumentWindow
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0039

By default, if you press Ctrl+Tab, you get the IDE Navigator, as shown in the following illustration:

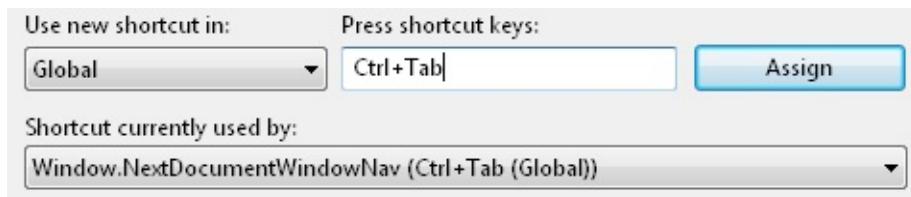


Some people don't like this feature and instead would like to just iterate through

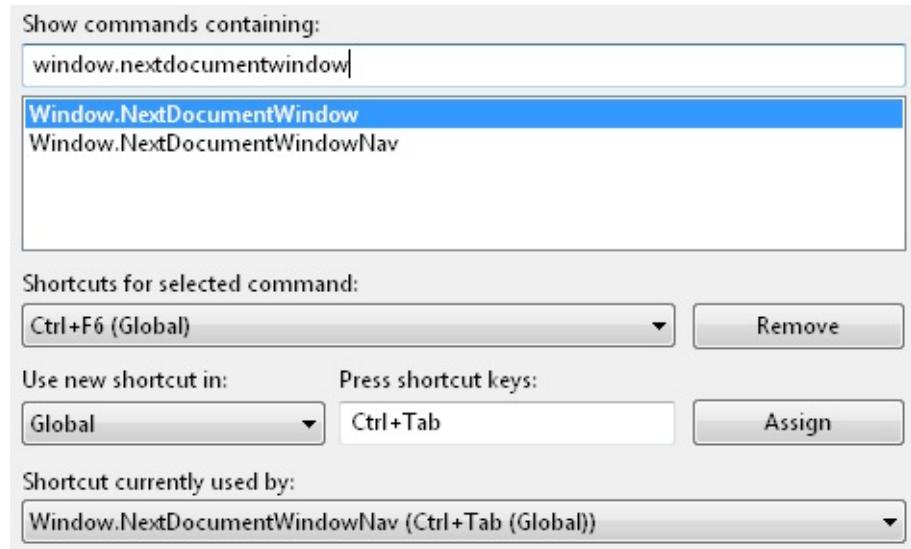
open document tabs.

This behavior is bound to Ctrl+ F6 and to Ctrl+Shift+F6 in the General settings, but some people don't like this key combination.

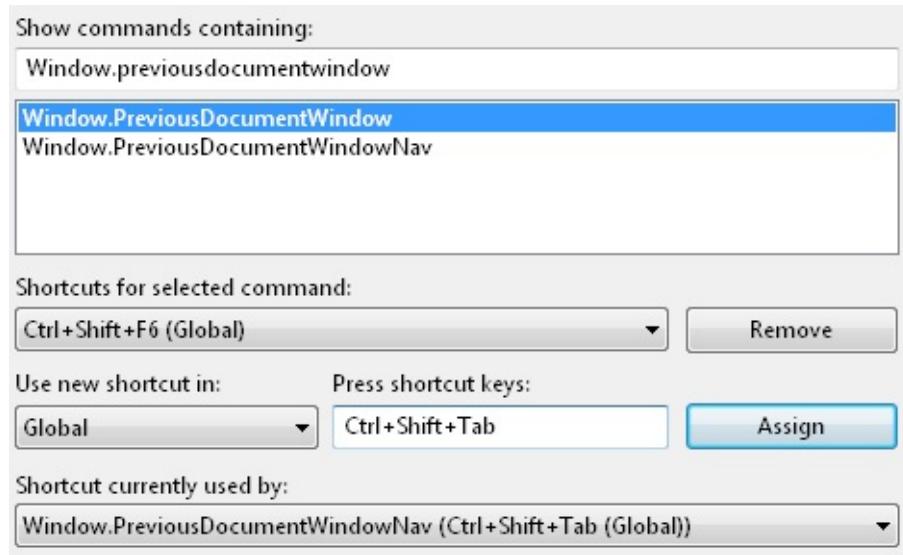
You can easily rebind the commands to Ctrl+Tab and Ctrl+Shift+Tab. If you go to Tools | Options | Environment | Keyboard, you can see that Ctrl+Tab is bound to Window.NextDocumentWindowNav:



If you assign Ctrl+Tab to Window.NextDocumentWindow instead, you can see the result in the following illustration:



The IDE Navigator does not show up anymore, and instead Ctrl+Tab iterates through the open document tabs. If you like this, you might want to bind Ctrl+Shift+Tab to Window.PreviousDocumentWindow as well:



If you don't like the new setup, you can always reverse the process and rebind Ctrl+Tab and Ctrl+Shift+Tab to Window.NextDocumentWindowNav and Window.PreviousDocumentWindowNav, respectively.

For more information about binding keyboard shortcuts, refer to vstipTool0063 ([03.26 Keyboard Shortcuts: Creating New Shortcuts](#) page 127).

## AX.43 Thumbnail Previews in the IDE Navigator

VERSIONS	2010
CODE	vstipTool0113

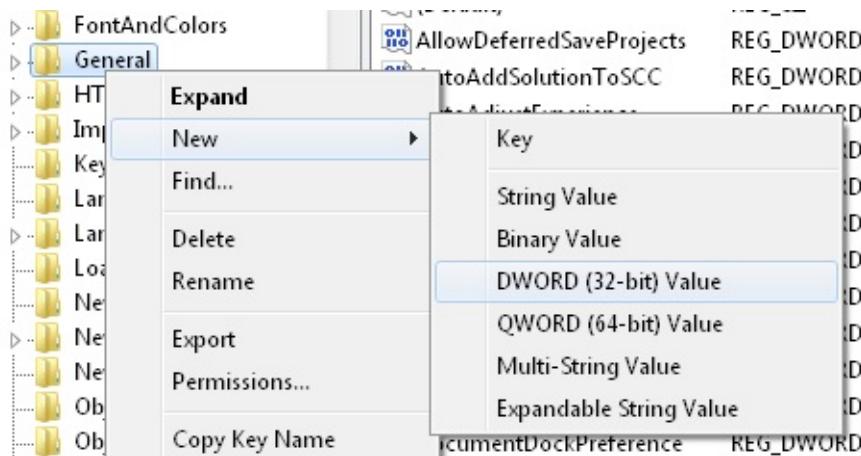
In vstipTool0023 ([04.08 Using the IDE Navigator](#), page 160), I show you how to use the IDE Navigator. This next tip comes from Paul Harrington on the Visual Studio Team. In Visual Studio 2008, when you used the IDE Navigator, you could also see thumbnail previews of the documents in the list. However, this feature was removed from Visual Studio 2010. The good news is that the feature is still there.

### WARNING

This tip requires you to add a value to the registry, so you do this at your own risk. And this solution has been known to not always work 100 percent of the time.

Open up the registry (regedit.exe), and go to HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\10.0\General.

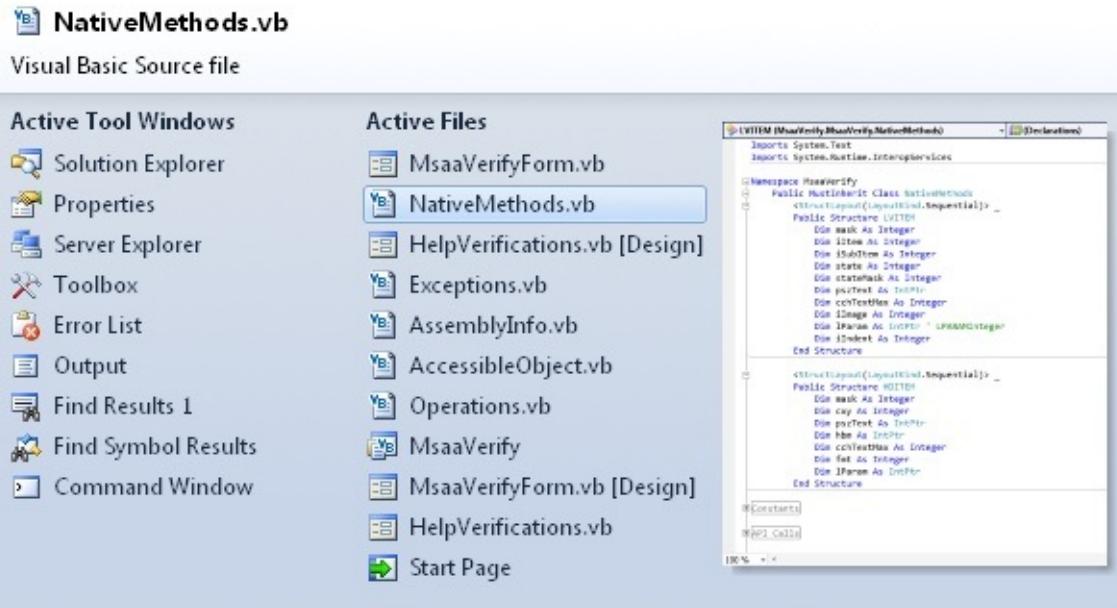
Now right-click the General key, and add a new DWORD value:



Call it ShowThumbnailsOnNavigation, and set the value to 1:

ShowThumbnailsOnNavigation REG\_DWORD 0x00000001 (1)

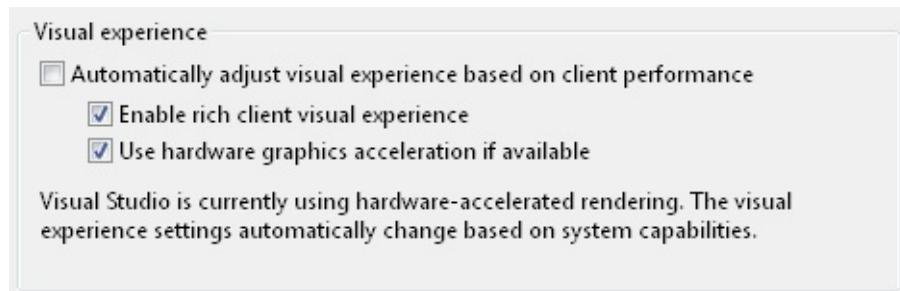
Now you should have the thumbnail preview available:



D:\Documents\Visual Studio 2010\Projects\MsaaVerifySource\NativeMethods.vb

## Troubleshooting

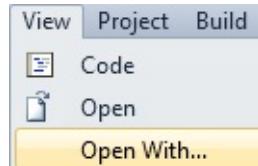
This is a little graphics intensive, so if you don't see the thumbnail, it could be because you haven't enabled the rich client visual experience under Tools | Options | Environment | General:



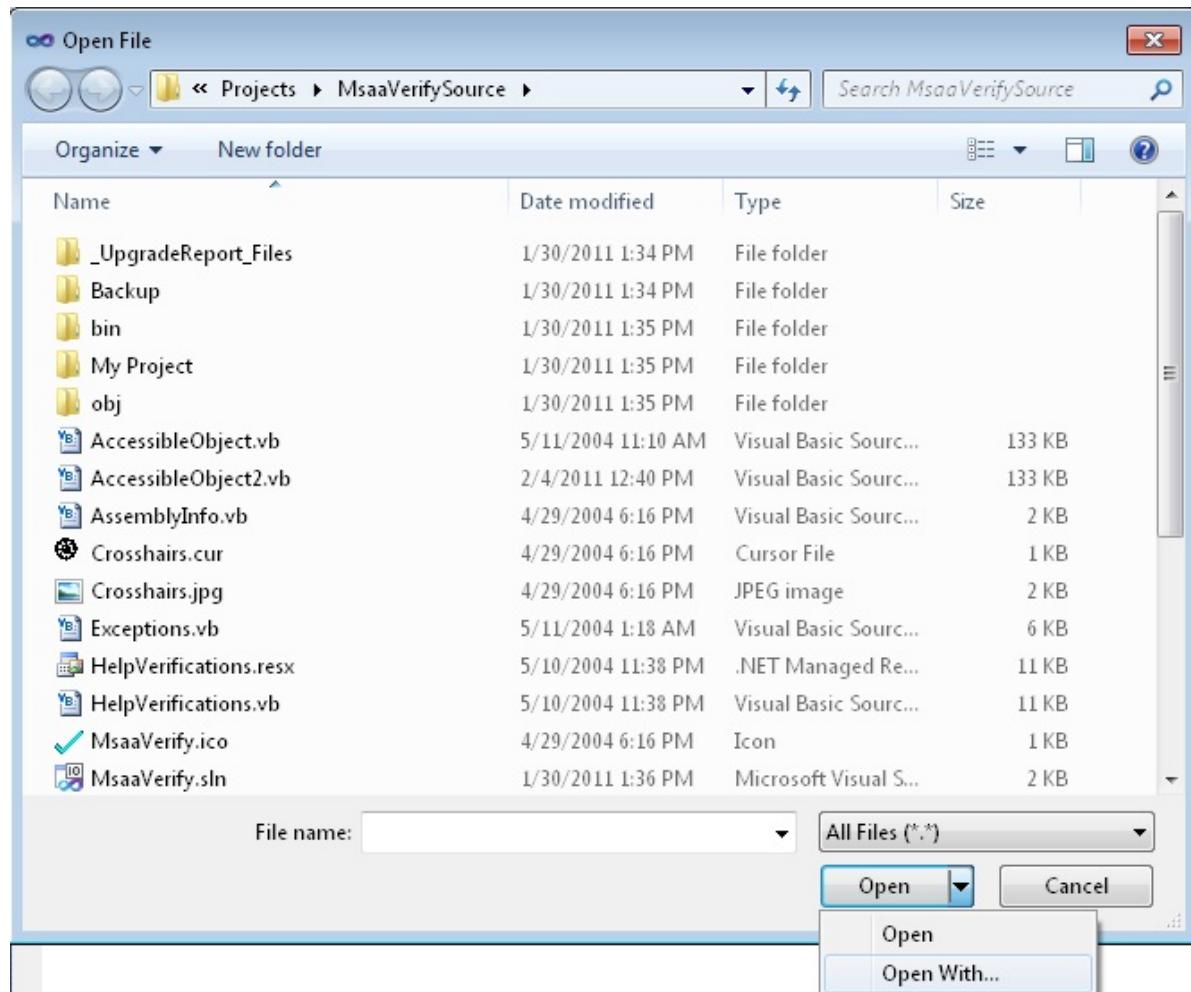
## AX.44 Changing Editors Using Open With

<b>WINDOWS</b>	Alt,V, N
<b>MENU</b>	View   Open With
<b>COMMAND</b>	View.OpenWith
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEnv0037

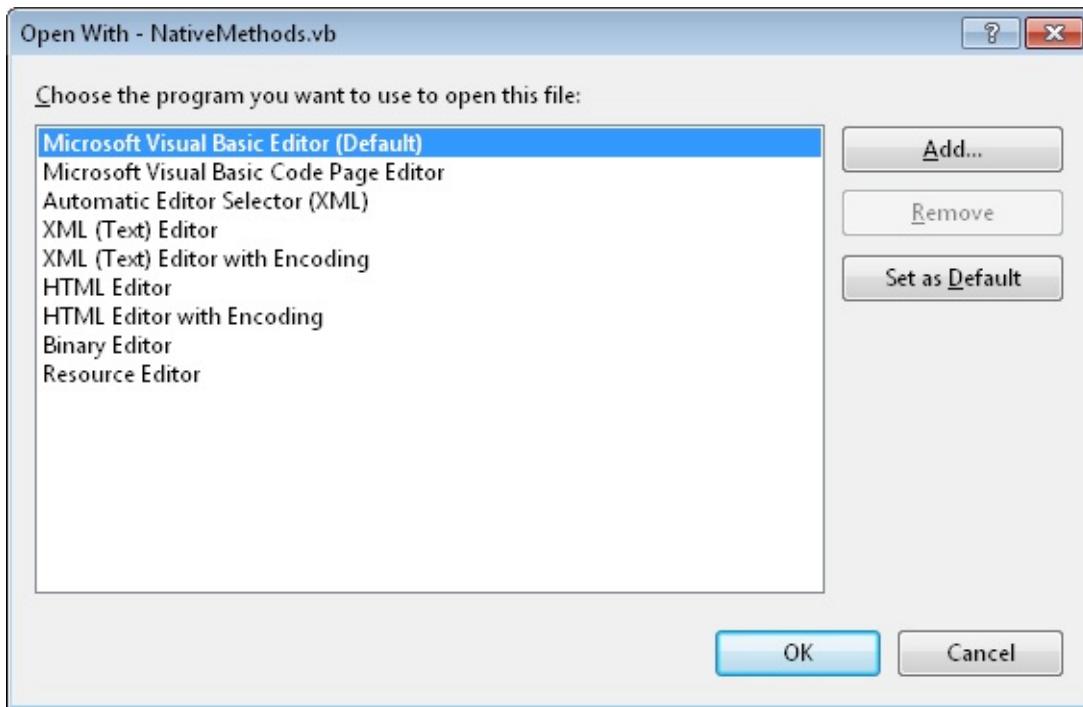
You can use Open With to change the editor used to view a file. You can use this feature in several ways. For example, if you have an existing file open, you can go to View | Open With:



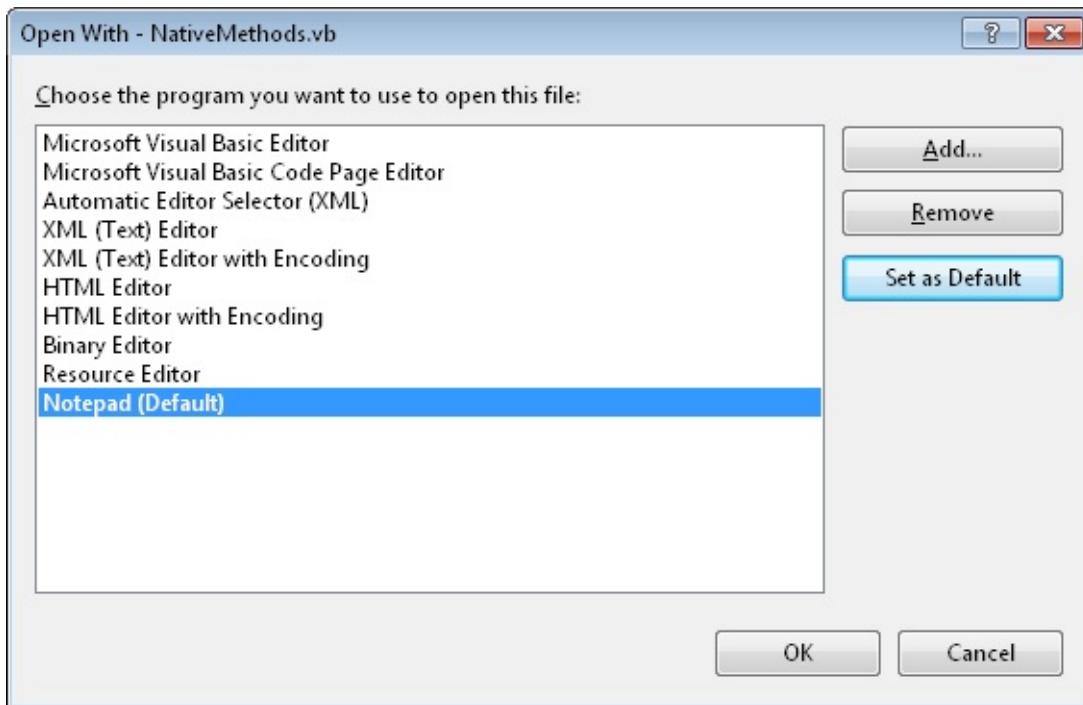
For files not opened yet, you can use this option from inside the Open File dialog box:



Regardless of the method used, you get the Open With dialog box:



Notice, in this example, that one editor is the default editor for the file type you want to open. This can easily be changed by selecting a new editor and clicking Set As Default:

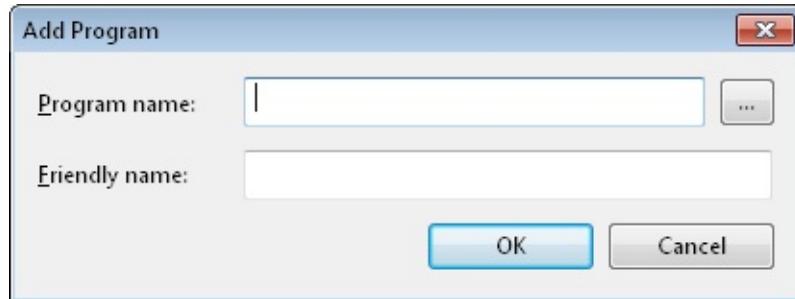


### WARNING

Use this feature at your own risk, because it's important to be sure of the editor you are using before you

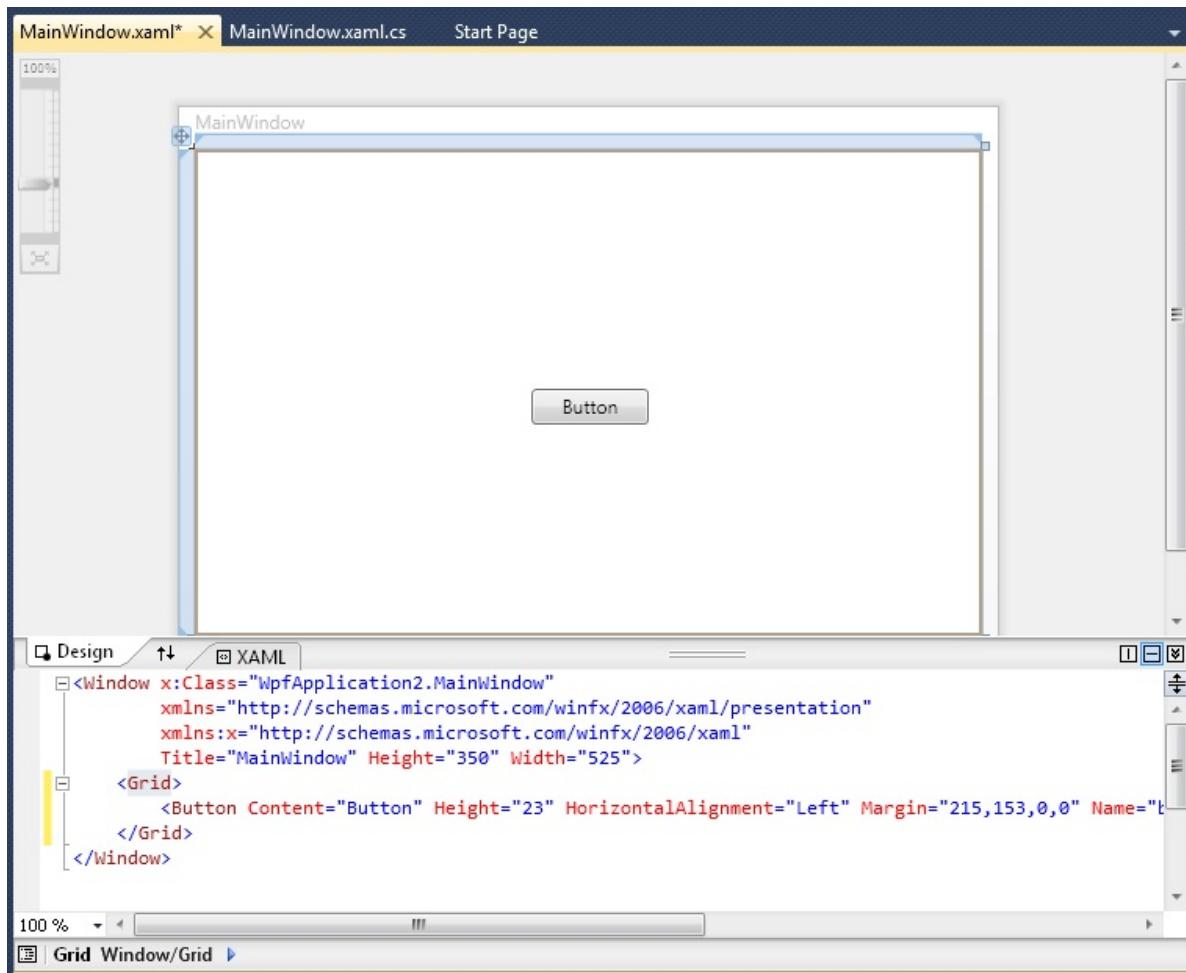
make it the default.

Notice the Add button, shown in the following illustration, which enables you to indicate new programs that you want to use for opening files:



## Example

One common use of Open With is to enable source code editing for WPF files. Normally, you get a designer/code view:



To enable source code editing more easily, first use Open With and select Source

## Code (Text) Editor:

XML (Text) Editor with Encoding  
Source Code (Text) Editor  
Source Code (Text) Editor With Encoding  
HTML Editor

Now you should see a “code only” view of the XAML:



The screenshot shows a Windows application window titled "Source Code (Text) Editor". The tab bar at the top includes "MainWindow.xaml" (which is highlighted in yellow), "MainWindow.xaml", "MainWindow.xaml.cs", and "Start Page". The main content area displays the XAML code for a window:

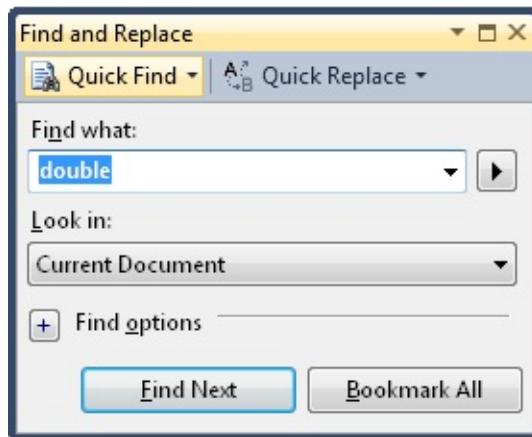
```
<Window x:Class="WpfApplication2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Button Content="Button" Height="23" HorizontalAlignment="Left"
    </Grid>
</Window>
```

## Additional Tips from Chapter 5

### AX.45 Using a Simple Quick Find

<b>DEFAULT</b>	Ctrl+F
<b>VISUAL BASIC 6</b>	Ctrl+F
<b>VISUAL C# 2005</b>	Ctrl+F
<b>VISUAL C++ 2</b>	Alt+F3
<b>VISUAL C++ 6</b>	Ctrl+F
<b>VISUAL STUDIO 6</b>	Ctrl+F
<b>WINDOWS</b>	Alt,E, F, F
<b>MENU</b>	Edit   Find and Replace   Quick Find
<b>COMMAND</b>	Edit.Find
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0006

You can do a simple find anytime you want by pressing Ctrl+F to bring up the Quick Find window:



The Find What area is automatically prepopulated with the word the cursor was currently on, or you can type in a new one.

To start, just press Enter or click Find Next, and the find operation finds the next instance of the search term you are looking for. By default, it looks from the

current cursor location downward.

The search continues until you reach the end of the document, and then it returns to the beginning and shows the following dialog box:

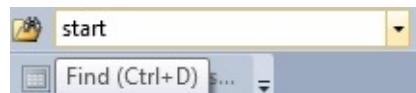


As you can see, you can turn this off by clearing the Always Show This Message check box so that it doesn't annoy you.

## AX.46 Using the Find Combo Box

<b>DEFAULT</b>	Ctrl+D
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	Ctrl+ /
<b>VISUAL C++ 2</b>	Ctrl+D; Ctrl+F; Ctrl+A
<b>VISUAL C++ 6</b>	Ctrl+D
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+F
<b>WINDOWS</b>	Ctrl+Shift+F
<b>COMMAND</b>	Edit.GoToFindCombo
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0009

While it tends to get ignored sometimes, the Find Combo box is actually quite useful. In the following examples, I'll show how you can quickly use this area while you are writing your code:



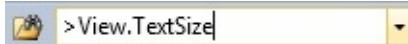
It can be used as a quick way to go to a line without any dialog boxes popping up. Just press Ctrl+D to get to the combo box, and then type in a line number, and finally, press Ctrl+G to go to the line number you typed:



Also, it can be used to execute commands by pressing Ctrl+Forward slash (/) followed by your command:

**NOTE**

Ctrl+/ is bound to the **Tools.GoToCommandLine** command for all languages except C#.

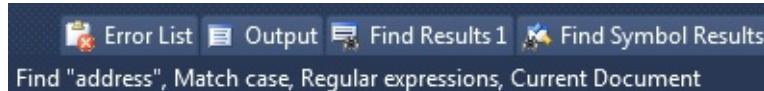


Its primary use, however, is to perform a simple find operation. Press Ctrl+D, type in whatever you are looking for, and press Enter:

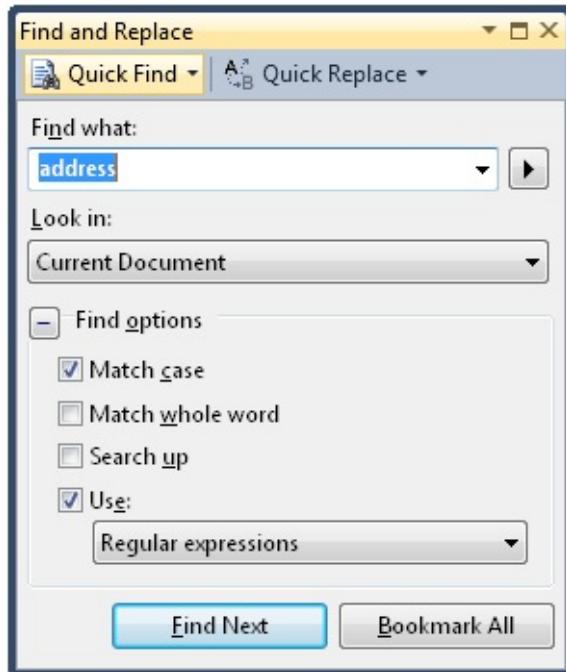


It finds the next instance of the search term. If you have a lot of text to go through, you can hold down the Enter key to quickly go through the document. One other thing to note is that, unlike a Quick Find, this find does *not* give you a dialog box telling you when you have looped around to where you started.

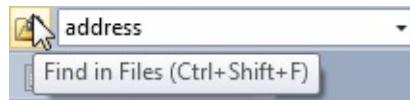
When you use this feature, *pay attention to the status bar* at the bottom. It shows you all the options that have been set for the current search:



The options used here are set in the Quick Find tool window (Ctrl+F):



Last, but certainly not least, is the Find In Files button, which brings up the Find In Files tool window (see vtipFind0013, [05.09 Find In Files: Find Options](#), page 186):

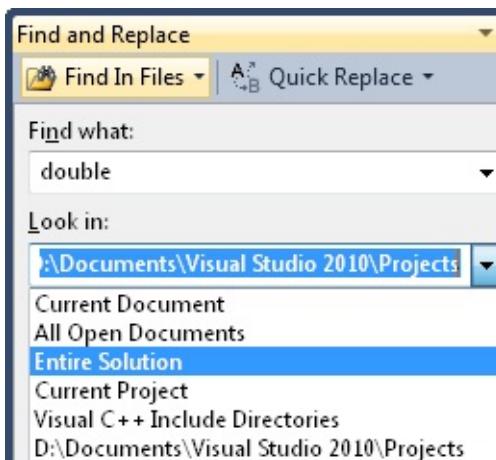


## AX.47 Customize the Files to Search with Find In Files

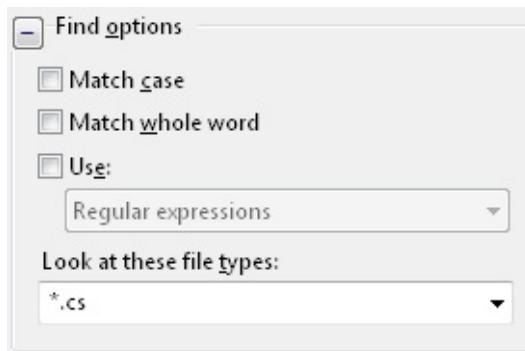
<b>DEFAULT</b>	Ctrl+Shift+F
<b>VISUAL BASIC 6</b>	Ctrl+Shift+F
<b>VISUAL C# 2005</b>	Ctrl+Shift+F
<b>VISUAL C++ 2</b>	Ctrl+Shift+F
<b>VISUAL C++ 6</b>	Ctrl+Shift+F
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt, E, F, I
<b>MENU</b>	Edit   Find and Replace   Find In Files
<b>COMMAND</b>	Edit.FindinFiles

<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0005

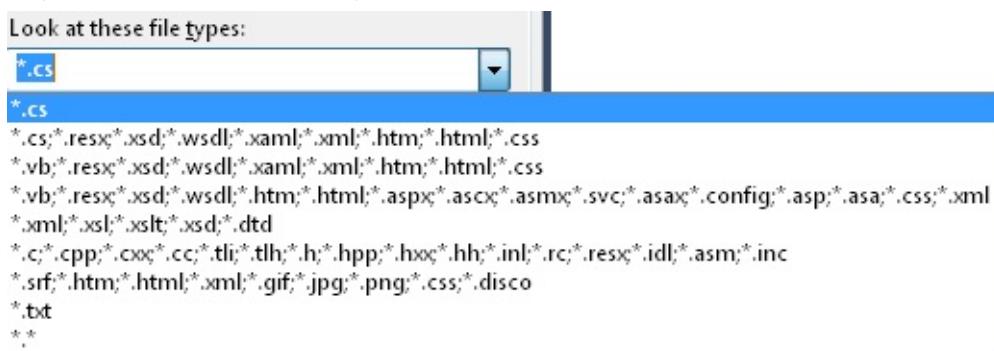
In the Find In Files dialog box (Ctrl+Shift+F), choose any option *except* Current Document and All Open Documents:



The Look At These File Types combo box is enabled under Find Options:



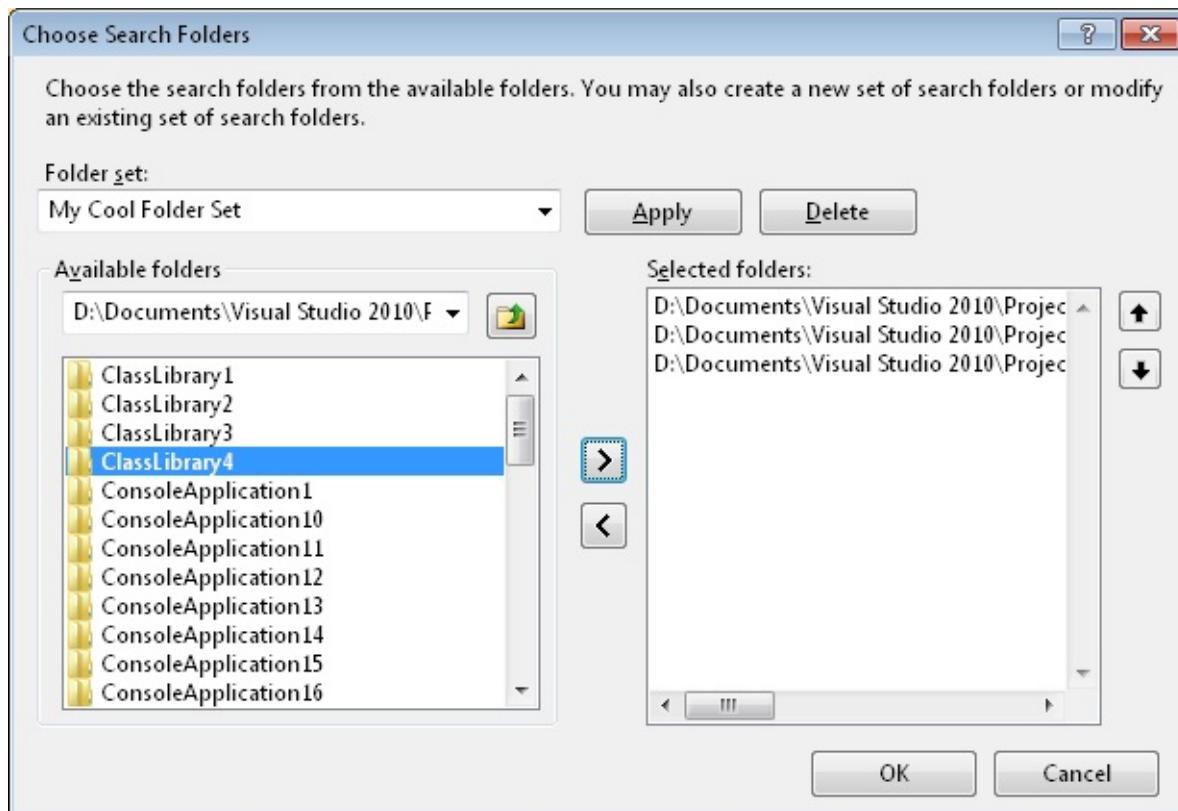
Now you can choose from the predetermined list of file types, or you can include your own (semicolon delimited) list:



You can also create a customized set of folders to search in by going back to the Look In area and clicking the ellipsis:



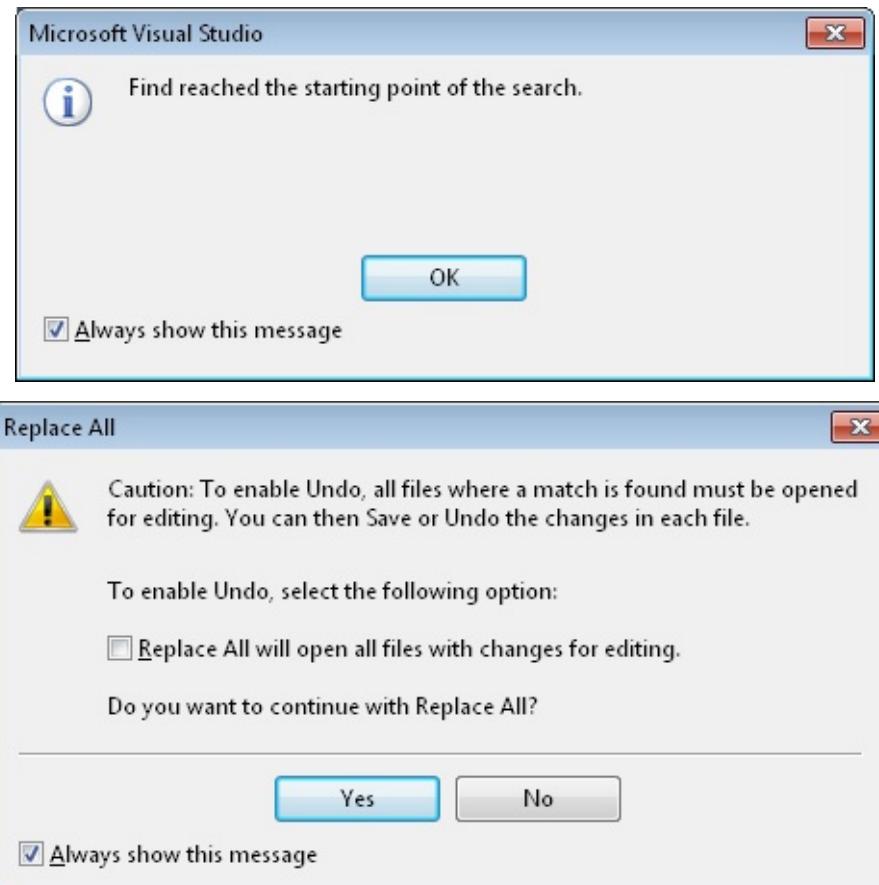
Now you have the Choose Search Folders dialog box, where you can completely customize the folders to look in, and you can even create a custom name for the folder set by typing a new name in the Folder Set area:



## AX.48 How to Show and Hide Find Messages

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Find and Replace
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0017

When working with the various find-tool windows, you sometimes get messages based on what you are doing. These range from informational messages to warning messages, as shown in the following illustrations:



If you clear the Always Show This Message check box, the messages go away. But what if you want the messages back or don't want to wait for a message to pop up until you can turn the messages off? No problem. Just go to Tools | Options | Environment | Find And Replace:

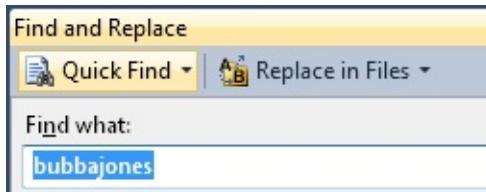


The Display Informational Messages and Display Warning Messages check boxes toggle, showing these messages when you use a find operation. Informational messages really aren't very important, but you should consider carefully whether turning off warning messages is a good idea.

## AX.49 How to Not Automatically Search for the Currently Selected Word

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Find and Replace
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipFind0018

Does it annoy you when your find operations automatically populate with the word you happen to have the cursor in?



You can simply go to Tools | Options | Environment | Find And Replace and clear the Automatically Populate Find What With Text From The Editor check box:

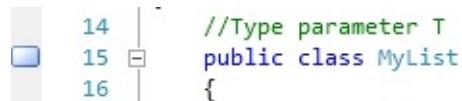


## AX.50 Setting Bookmarks

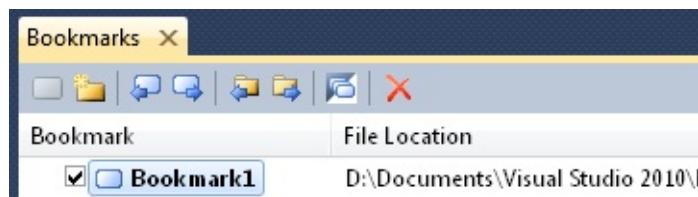
<b>DEFAULT</b>	Ctrl+K, Ctrl+K
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+K; Ctrl+K, T
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+K; Ctrl+B, Ctrl+T; Ctrl+B, T
<b>VISUAL C++ 2</b>	Ctrl+F2
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+K; Ctrl+F2
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+K
<b>WINDOWS</b>	Alt,E, K, T
<b>MENU</b>	Edit   Bookmarks   Toggle Bookmark
<b>COMMAND</b>	Edit.ToggleBookmark
<b>VERSIONS</b>	2005, 2008, 2010

Bookmarks are a pretty cool feature that a lot of people don't seem to know about. Essentially, bookmarks provide a way to mark locations in your code. Unlike comment tokens ("TODOs"), bookmarks are not stored with the source code and thus are seen only by you.

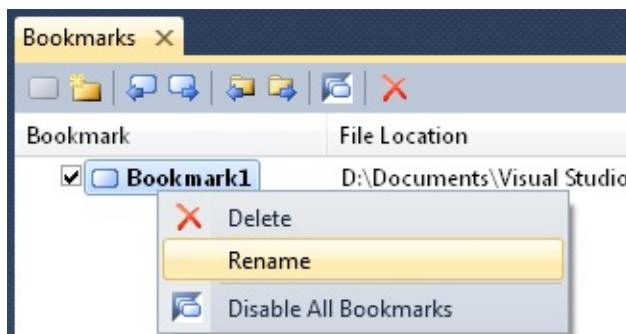
You have numerous ways to set a Bookmark. The simplest way is to use Ctrl+K, Ctrl+K to create a single bookmark:



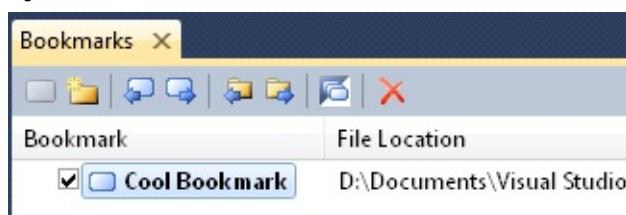
When you set a bookmark, it creates a glyph in the margin (see image above) and creates an entry in the Bookmarks window:



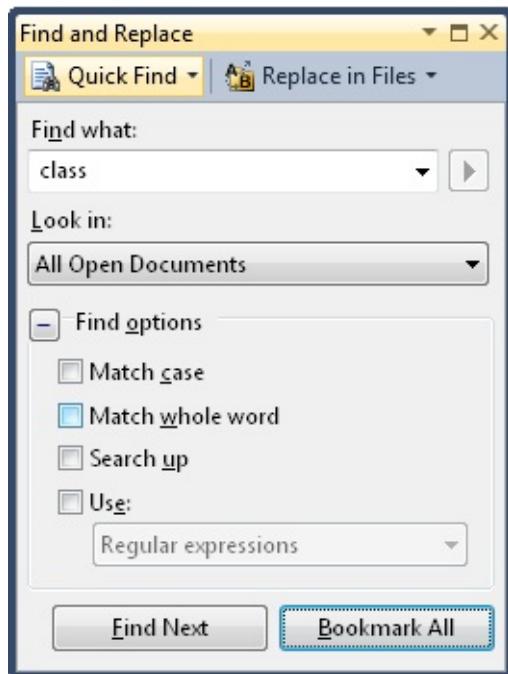
The good news is that you don't have to keep the default name that is given for the bookmark. Just right-click the entry in the window, and choose Rename:



Then put in whatever you want for the name:



You can continue to use either the keyboard command or the menu option to create bookmarks. Another great way to create bookmarks is to use the Bookmark All (bottom right) button in the Quick Find dialog box (Ctrl+F):

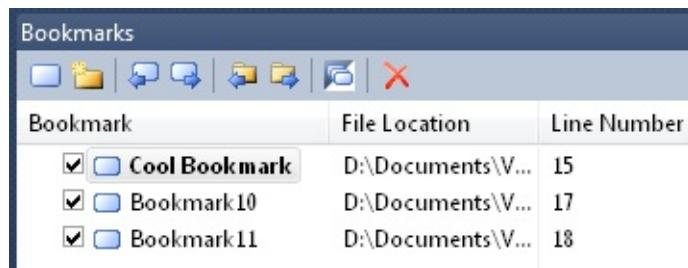


The Bookmark All button becomes available only if you choose Current Document or All Open Documents from the Look In drop-down box.

## AX.51 Organizing Bookmarks

VERSIONS	2005, 2008, 2010
CODE	vstipTool0048

The Bookmarks window gives you some basic information about the bookmarks, and you have the ability to rename them as well:



However, when you have a lot of bookmarks, it is probably a good idea to organize your bookmarks. You can drag the bookmarks around to reorganize them in the list:

<input checked="" type="checkbox"/>  Bookmark44	D:\Documents\V... 23
<input checked="" type="checkbox"/>  Bookmark45 	D:\Documents\V... 24
<input checked="" type="checkbox"/>  Bookmark46	D:\Documents\V... 25
<input checked="" type="checkbox"/>  Bookmark47	D:\Documents\V... 26

Another thing you can do is create folders in the Bookmarks window (Ctrl+K, Ctrl+F):

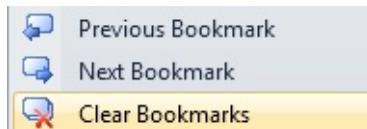


Just give the folder(s) a name, and you have a great place to organize bookmarks:



Now you can just drag your bookmarks into the folder(s) you have created:

Naturally, if you have no need for a bookmark, you can simply press Del to delete the currently selected one, or you can go to Edit | Bookmarks | Clear Bookmarks to remove all your bookmarks:



## AX.52 Navigating Bookmarks

<b>DEFAULT</b>	Ctrl+K, Ctrl+P (previous bookmark); Ctrl+K, Ctrl+N (next bookmark); Ctrl+Shift+K, Ctrl+Shift+P (previous bookmark in folder); Ctrl+Shift+K, Ctrl+Shift+N (next bookmark in folder)
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+P (previous bookmark); Ctrl+K, P (previous bookmark); Ctrl+K, Ctrl+N (next bookmark); Ctrl+K, N (next bookmark); Ctrl+Shift+K, Ctrl+Shift+P (previous bookmark in folder); Ctrl+Shift+K, Ctrl+Shift+N (next bookmark in folder)
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+P (previous bookmark); Ctrl+B, Ctrl+P (previous bookmark); Ctrl+B, P (previous bookmark); Ctrl+K, Ctrl+N (next bookmark); Ctrl+B, Ctrl+N (next bookmark); Ctrl+B, N (next bookmark); [no shortcut] (previous bookmark in folder); [no shortcut] (next bookmark in folder)
<b>VISUAL C++ 2</b>	Shift+F2 (previous bookmark); F2 (next bookmark); Ctrl+Shift+K, Ctrl+Shift+P (previous bookmark in folder); Ctrl+Shift+K, Ctrl+Shift+N (next bookmark in folder)
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+P (previous bookmark); Shift+F2 (previous bookmark); Ctrl+K, Ctrl+N (next bookmark); F2 (next bookmark); Ctrl+Shift+K, Ctrl+Shift+P (previous bookmark in folder); Ctrl+Shift+K, Ctrl+Shift+N (next bookmark in folder)

<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+P (previous bookmark); Ctrl+K, Ctrl+N (next bookmark); Ctrl+Shift+K, Ctrl+Shift+P (previous bookmark in folder); Ctrl+Shift+K, Ctrl+Shift+N (next bookmark in folder)
<b>WINDOWS</b>	Alt+E, K, [P (previous),B (next)]
<b>MENU</b>	Edit   Bookmarks   [Previous, Next] Bookmark [In Folder, Document]
<b>COMMAND</b>	Edit.[Previous, Next] Bookmark; Edit.[Previous, Next] Bookmark [In Folder, Document]
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0049

After you have created and organized your bookmarks, you want to navigate them. You can go to Edit | Bookmarks to see several available options:



## [Next,Previous] Bookmark In Document

This option allows you to restrict browsing to only those bookmarks in the current open document, independently of how they are organized in the Bookmarks window.

So, even though you have lots of bookmarks in different folders within the Bookmarks window, this option moves only between the bookmarks in the current document, ignoring any organization.

## [Next,Previous] Bookmark In Folder

This option allows you to restrict browsing to only those bookmarks in the current folder in the Bookmarks window, independent of how they are arranged in the source code.

Pretty much the opposite of the previous feature, this feature ignores how the bookmarks are organized in the source code and moves only within the bookmarks in the current folder within the Bookmarks window:

Bookmark	File Location	Line Number
Worker Role		
Bookmark54	D:\Documents\V...	23
Bookmark55	D:\Documents\V...	24
Bookmark56	D:\Documents\V...	25
Bookmark57	D:\Documents\V...	26
Bookmark58	D:\Documents\V...	16
Bookmark59	D:\Documents\V...	17
Bookmark60	D:\Documents\V...	18
Data Logic		

When you reach the last Bookmark in the folder, it loops back around to the first bookmark in the current folder:

Bookmark	File Location	Line Number
Worker Role		
Bookmark54	D:\Documents\V...	23
Bookmark55	D:\Documents\V...	24
Bookmark56	D:\Documents\V...	25
Bookmark57	D:\Documents\V...	26
Bookmark58	D:\Documents\V...	16
Bookmark59	D:\Documents\V...	17
Bookmark60	D:\Documents\V...	18
Data Logic		

## [Next,Previous] Bookmark

This option allows you to navigate between bookmarks in the Bookmarks window. This feature is very similar to the [Previous, Next] Bookmark In Folder feature and moves sequentially through bookmarks.

The difference comes when you reach the last bookmark in a folder. Instead of looping back around to the first bookmark in the folder, this option continues to the next folder and moves sequentially through those bookmarks as well (and so on):

Bookmark60	D:\Documents\V...	18
Data Logic		
Bookmark50	D:\Documents\V...	21
Bookmark51	D:\Documents\V...	22

## Additional Tips from Chapter 6

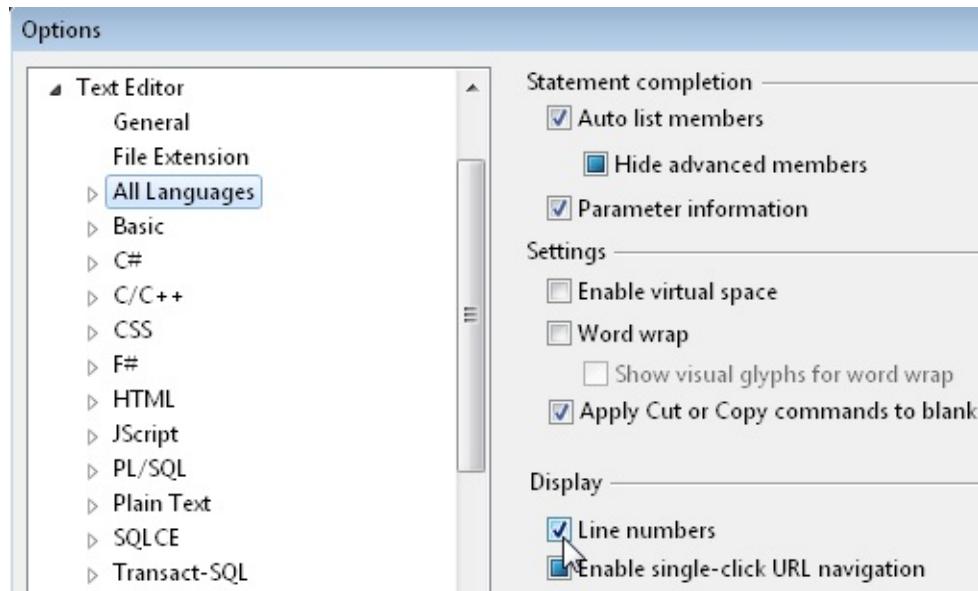
### AX.53 Turn On Line Numbers

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Text Editor   All Languages   General   Display
<b>COMMAND</b>	Macros.Samples.Utilities.TurnOnLineNumbers; Macros.Samples.Utilities.TurnOffLineNumbers
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0025

As you can see in the following illustration, it's great to have line numbers in your code:

```
12  □namespace G
13  {
14      //Type
15  □public
16  {
17      pro
18      pro
19
20      //
21  □pro
22  {
```

Line numbers are not on by default. To turn on line numbers, just go to Tools | Options | Text Editor | General | Display and select the Line Numbers check box:

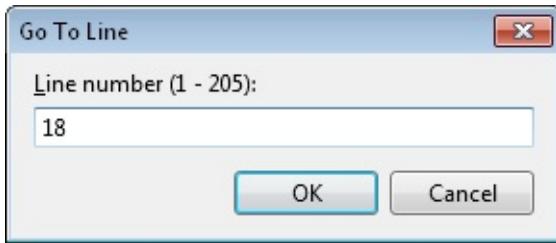


## AX.54 Go to a Line Number

<b>DEFAULT</b>	Ctrl+G
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	Ctrl+G
<b>VISUAL C++ 2</b>	Ctrl+G
<b>VISUAL C++ 6</b>	Ctrl+G
<b>VISUAL STUDIO 6</b>	Ctrl+G
<b>WINDOWS</b>	Alt,E,G
<b>MENU</b>	Edit   Go To
<b>COMMAND</b>	Edit.GoTo
<b>VERSIONS</b>	2005, 2008, 2010, 2010 SP1
<b>CODE</b>	vstipEdit0026

You have three main ways you can go to any line number in your code.

First, you can go to any line number by simply pressing Ctrl+G to see the following dialog box. Just type in your desired line number, and click OK. The cursor moves to the line number you typed:



Second, you can double-click in the status bar area that shows your current location (lower-right part of your screen) to get the “Go To Line” dialog box:

Ln 18      Col 1      Ch 1

Third, you can use the Find Combo box to quickly go to a line number. This technique does not work with a default installation of Visual Studio 2010 but was fixed with Service Pack 1. It requires two steps:

1. Press Ctrl+D to put your cursor into the Find Combo box:

2. Type in any line number, and press Ctrl+G to go to that line number:

#### NOTE

In Visual Studio 2010, you might have to press the escape key (Esc) to get out of the Find Combo box and back to your code.

## AX.55 Comment and Uncomment Code

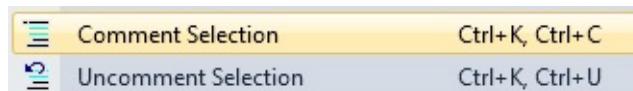
<b>DEFAULT</b>	Ctrl+K, Ctrl+C (comment); Ctrl+K, Ctrl+U (uncomment)
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+C (comment); Ctrl+K, Ctrl+U (uncomment)
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+C (comment); Ctrl+E, Ctrl+C (comment); Ctrl+E, C (comment); Ctrl+E, Ctrl+U (uncomment); Ctrl+E, U (uncomment); Ctrl+K, Ctrl+U (uncomment)
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+C (comment); Ctrl+K, Ctrl+U (uncomment)
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+C (comment); Ctrl+K, Ctrl+U (uncomment)

<b>WINDOWS</b>	Alt,E, V, M; Alt,E, V, E;
<b>MENU</b>	Edit   Advanced   Comment Selection; Edit   Advanced   Uncomment Selection
<b>COMMAND</b>	Edit.CommentSelection; Edit.UncommentSelection
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0047

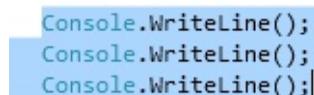
Sometimes it's the simple things we forget about. So I present to you the classic Comment and Uncomment selection. Naturally, you have the Comment and Uncomment buttons, shown in the following illustration:



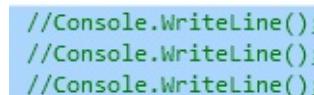
And, of course, you have the menu items:



But it's the keyboard shortcuts that are really important. These, predictably, comment or uncomment lines of code for you. So let's say you have some code you want commented out. Just select it, as shown in the following illustration:



Then press **Ctrl+K, Ctrl+C**:



OK, great, but what if you don't want to use the mouse? No problem. Just hold **Alt+Shift+[Up or Down Arrow]** to do a vertical selection. You don't have to select the entire line to comment or uncomment it.

#### NOTE

In Visual Studio 2005 and Visual Studio 2008, you have to go right or left one character before you can go up or down for vertical selection.

```
//Console.WriteLine();
//Console.WriteLine();
//Console.WriteLine();
```

Then press Ctrl+K, Ctrl+U (in this example):

```
Console.WriteLine();
Console.WriteLine();
Console.WriteLine();
```

And there you go. You now have Comment and Uncomment actions anytime you want them.

## AX.56 Select the Current Word

<b>DEFAULT</b>	Ctrl+W
<b>VISUAL BASIC 6</b>	Ctrl+Shift+W
<b>VISUAL C# 2005</b>	Ctrl+Shift+W
<b>VISUAL C++ 2</b>	Ctrl+W
<b>VISUAL C++ 6</b>	Ctrl+W
<b>VISUAL STUDIO 6</b>	Ctrl+W
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.SelectCurrentWord
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0039

You can easily select the current word in Visual Studio by putting your cursor in the word:

```
public
```

And then just press Ctrl+W:

```
public
```

## AX.57 Delete Through the Beginning or End of a Word

<b>DEFAULT</b>	Ctrl+Del (delete to end); Ctrl+Backspace (delete to start)
<b>VISUAL BASIC 6</b>	Ctrl+Del (delete to end); Ctrl+Backspace (delete to start)

<b>VISUAL C# 2005</b>	Ctrl+Del (delete to end); Ctrl+Backspace (delete to start)
<b>VISUAL C++ 2</b>	Ctrl+Del (delete to end); Ctrl+Backspace (delete to start)
<b>VISUAL C++ 6</b>	Ctrl+Del (delete to end); Ctrl+Backspace (delete to start)
<b>VISUAL STUDIO 6</b>	Ctrl+Del (delete to end); Ctrl+Backspace (delete to start)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.WordDeleteToEnd; Edit.WordDeleteToStart
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0040

You can delete from the current cursor position through the beginning or end of a word. Let me illustrate with a simple example. Let's say you want to change the word "public":

Given the current cursor position, you could delete to the end of the word by pressing Ctrl+Del:

Then you could type the new word:

This is a somewhat contrived example, but you get the idea. Additionally, Ctrl+Backspace deletes from the current cursor location through the beginning of a word.

## AX.58 Click and Drag Text to a New Location

<b>COMMAND</b>	OtherContextMenus.DragandDrop.MoveHere
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0041

Often you will find yourself with the need to move text around in the Editor. Text can easily be moved around by simply selecting it and dragging it to a new location. Start with something you want to move, like a method:

```

45 ┌─┐     public MyList()
46   └─┘     {
47       head = null;
48   }
49
50 //T as method parameter type.
51 ┌─┐     public void AddHead(T t)
52   └─┘     {
53         Node n = new Node(t);
54         n.Next = head;
55         head = n;
56     }

```

In this case, you might want to collapse the code to make it easier to select and move, as shown in the following illustration:

```

45 ┌─┐     public MyList()...
49
50 //T as method parameter type.
51 ┌─┐     public void AddHead(T t)
52   └─┘     {
53         Node n = new Node(t);
54         n.Next = head;
55         head = n;
56     }
57

```

Now just select the text, and then click and drag (left mouse button) to move the cursor to the destination:

```

45 ┌─┐     public MyList()...
49
50 //T as method parameter type.
51 ┌─┐     public void AddHead(T t)
52   └─┘     {
53         Node n = new Node(t);
54         n.Next = head;
55         head = n;
56     }
57
58
59
60

```

Release the mouse button to finish the move to the new location:

```

45
46
47      //T as method parameter type.
48  □ public void AddHead(T t)
49  {
50      Node n = new Node(t);
51      n.Next = head;
52      head = n;
53  }
54
55
56 □ public MyList()
57  {
58      head = null;
59  }
60

```

This technique can also be used to move text from one file to another. Just select the text, and then click and drag your mouse pointer over the tab for the new file:



Even though you get the “can’t drop” indicator, it switches to the new file. Just keep holding the mouse button down, and move the cursor to the new location:

```

8 □ class Class1
9 {
10    |
11    }
12 }
13

```

Then release and you are all set.

## AX.59 Make Selection Uppercase or Lowercase

<b>DEFAULT</b>	Ctrl+Shift+U (uppercase); Ctrl+U (lowercase)
<b>VISUAL BASIC 6</b>	Ctrl+Shift+U (uppercase); Ctrl+U (lowercase)
<b>VISUAL C# 2005</b>	Ctrl+Shift+U (uppercase); Ctrl+U (lowercase)
<b>VISUAL C++ 2</b>	Ctrl+Shift+U (uppercase); Ctrl+U (lowercase)
<b>VISUAL C++ 6</b>	Ctrl+Shift+U (uppercase); Ctrl+U (lowercase)
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+U (uppercase); Ctrl+U (lowercase)
<b>WINDOWS</b>	Alt, E, V, U (uppercase); Alt, E, V, L (lowercase)
<b>MENU</b>	Edit   Advanced   Make Uppercase; Edit   Advanced   Make Lowercase

<b>COMMAND</b>	Edit.MakeUppercase; Edit.MakeLowercase
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0044

You can easily change the case of any text:

```
Dim Bubba As String
```

Just select the as much of the word you want to change case for, and press Ctrl+Shift+U to make the selected characters all uppercase:

```
Dim BUBBA As String
```

Or if you prefer, you can always make all the selected characters lowercase by pressing Ctrl+U:

```
Dim bubba As String
```

Again, you don't have to select the whole word; you can select only the characters you want to change.

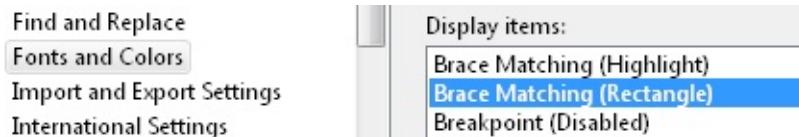
## AX.60 Brace Matching Rectangle

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Fonts and Colors   Display Items
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0050

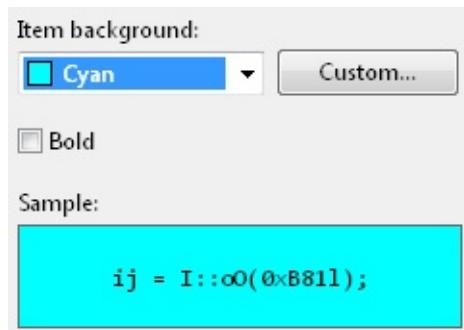
The default colors can be a pain sometimes, especially when it comes to certain colors. One of these, for me, is the Brace Matching Rectangle. The default colors are light grey:

```
static void Main
{
}
```

To change the color, first go to Tools | Options | Fonts and Colors | Display Items and choose Brace Matching (Rectangle), as shown in the following illustration:



Now choose your new color (Cyan in this example):



## A Note About VB

Visual Basic treats this a little differently. When you first close braces, it gives you the Brace Matching Rectangle color.

After the braces are in place, if you click next to them, it uses the highlighted reference colors.

## AX.61 Automatic Delimiter Highlighting

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Fonts and Colors   Display Items
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0071

You have probably seen automatic delimiter highlighting in action before. It shows up when you close parentheses, curly brackets, and other similar delimiters.

## C#

```
static void Main(string[] args)
{
}
```

## VB

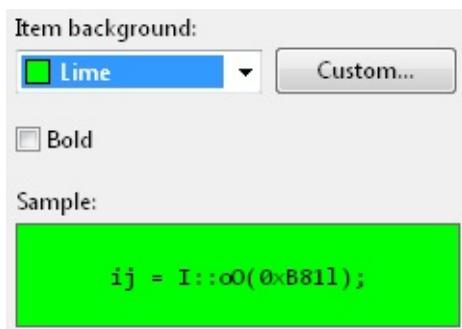
```
GetName(ByVal name As String, ByVal age As Integer)
```

## Changing Colors

You can modify the colors to suit your need by going to Tools | Options | Environment | Fonts and Colors and selecting Brace Matching (Rectangle):

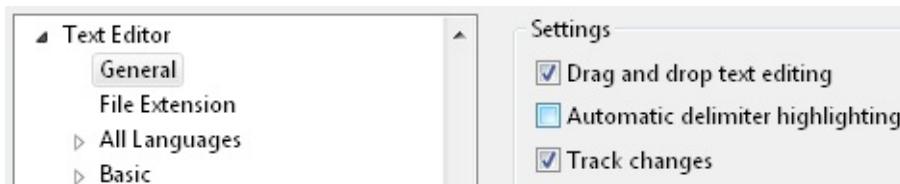


Let's change the background to lime green and click OK:



## Turning It Off

If you don't like this feature, you can go to Tools | Options | Text Editor | General and clear the Automatic Delimiter Highlighting check box, as shown in the following illustration:



## AX.62 Move or Select to the Top or Bottom of the Current View in the Editor

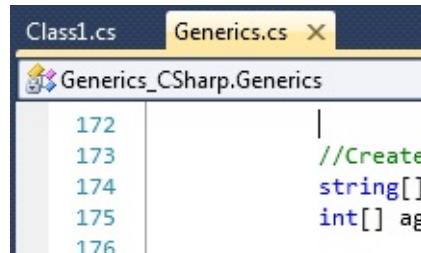
<b>DEFAULT</b>	Ctrl+PgUp (move top); Ctrl+PgDn (move bottom); Ctrl+Shift+PgUp (select top); Ctrl+Shift+PgDn (select bottom)
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	Ctrl+PgUp (move top); Ctrl+PgDn (move bottom); Ctrl+Shift+PgUp (select top); Ctrl+Shift+PgDn (select bottom)
<b>VISUAL C++ 2</b>	Ctrl+PgUp (move top); Ctrl+PgDn (move bottom); Ctrl+Shift+PgUp (select top); Ctrl+Shift+PgDn (select bottom)

<b>VISUAL C++ 6</b>	Ctrl+PgUp (move top); Ctrl+PgDn (move bottom); Ctrl+Shift+PgUp (select top); Ctrl+Shift+PgDn (select bottom)
<b>VISUAL STUDIO 6</b>	Ctrl+PgUp (move top); Ctrl+PgDn (move bottom); Ctrl+Shift+PgUp (select top); Ctrl+Shift+PgDn (select bottom)
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.ViewTop; Window.ViewBottom;Edit.ViewTopExtend;Edit.ViewBottomExtend
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0056

This tip comes in handy when you want to travel from one end of your screen to the other. For example, when you are working with documents, you might find yourself at the bottom of the screen and want to get to the top:

```
201 }  
202 | }  
203 | }  
204 }
```

Just press Ctrl+PgUp, and you are taken to the top of the screen as close to the current column position as possible:



Using Ctrl+PgDn takes you to the bottom of the screen. You can also use Ctrl+Shift+Pg[Up or Dn] to select everything from the current cursor position to the top or bottom of the screen.

## AX.63 Format the Current Document or Selection

<b>DEFAULT</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+F (selection)
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+F (selection)
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+D (document); Ctrl+E, Ctrl+D (document); Ctrl+E, D (document); Ctrl+K, Ctrl+F (selection); Ctrl+E, Ctrl+F (selection); Ctrl+E, F (selection)

<b>VISUAL C++ 2</b>	[no shortcut] (document); Ctrl+Shift+F (selection); Ctrl+Alt+I (selection)
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+F (selection); Alt+F8 (selection)
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+D (document); Ctrl+K, Ctrl+F (selection); Alt+F8 (selection)
<b>WINDOWS</b>	Alt,E, V, A (document); Alt,E, V, F (selection)
<b>MENU</b>	Edit   Advanced   Format Document; Edit   Advanced   Format Selection
<b>COMMAND</b>	Edit.FormatDocument; Edit.FormatSelection
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0057

Let's say you have some code that isn't formatted properly.

```

6  namespace ConsoleApplication24
7  {
8  class Program
9  {
10 static void Main(string[] args)
11 {
12
13 Console.WriteLine("blah");
14 }
15 }
16 }
17

```

Now you want it to look good. Just select the code, and then go to Edit | Advanced | Format Selection to get the result shown in the following illustration:

```

6  namespace ConsoleApplication24
7  {
8  class Program
9  {
10 static void Main(string[] args)
11 {
12
13     Console.WriteLine("blah");
14 }
15 }
16 }
17

```

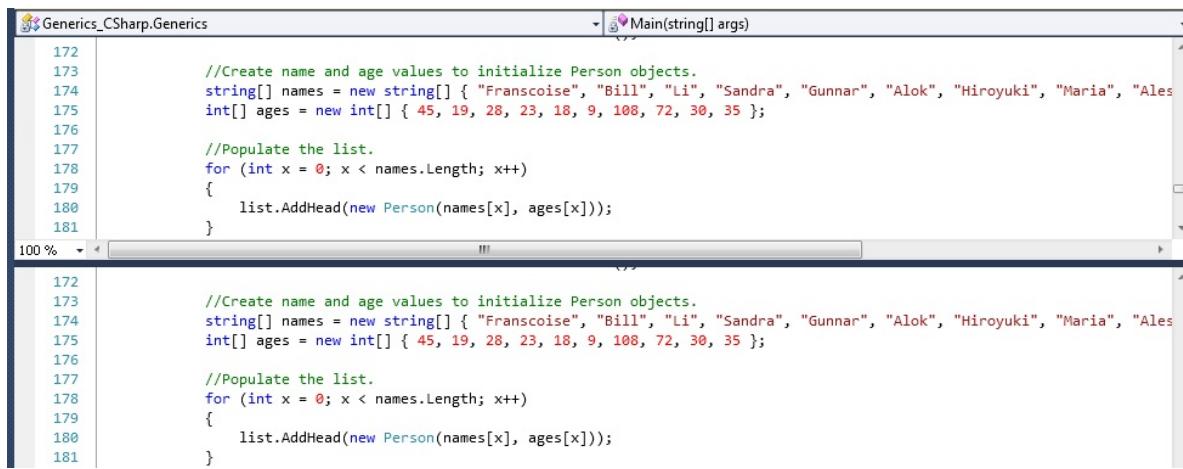
If you want to fix the formatting for the entire document you are currently working in, just go to Edit | Advanced | Format Document. This operation formats

everything in the current open document for you, without your having to select specific areas.

## AX.64 Use F6 to Jump Between Split Windows

<b>DEFAULT</b>	F6; Shift+F6
<b>VISUAL BASIC 6</b>	F6; Shift+F6
<b>VISUAL C# 2005</b>	[no shortcut]
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	F6; Shift+F6
<b>VISUAL STUDIO 6</b>	F6; Shift+F6
<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Window.NextSplitPane; Window.PreviousSplitPane
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0005

If you have split windows (Window | Split), you can easily move the cursor between the panes without using your mouse: Just press F6:



The screenshot shows a Visual Studio window with two vertically split panes. Both panes display the same C# code for initializing a list of Person objects. The code includes line numbers from 172 to 181. The top pane shows the full code, while the bottom pane shows a portion of it, likely due to scroll position or a different view mode.

```

172
173     //Create name and age values to initialize Person objects.
174     string[] names = new string[] { "Franscoise", "Bill", "Li", "Sandra", "Gunnar", "Alok", "Hiroyuki", "Maria", "Ales
175     int[] ages = new int[] { 45, 19, 28, 23, 18, 9, 108, 72, 30, 35 };
176
177     //Populate the list.
178     for (int x = 0; x < names.Length; x++)
179     {
180         list.AddHead(new Person(names[x], ages[x]));
181     }

```

## AX.65 Turn Off Single-Click URL Navigation in the Editor

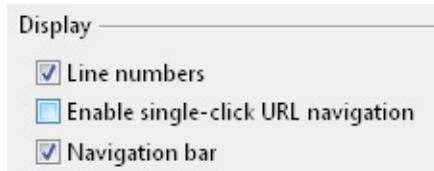
<b>WINDOWS</b>	Alt,T, O

<b>MENU</b>	Tools   Options   Text Editor   All Languages   General   Display
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0060

In most languages, by default, single-click URL navigation is turned on, which allows you to use Ctrl+Click to follow a link from the Editor:



This feature can be turned off by going to Tools | Options | Text Editor | All Languages [or your language] | General | Display and clearing the Enable Single-Click URL Navigation check box:



## AX.66 Hide the Vertical and/or Horizontal Scroll Bars

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   General   Display
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0058

This isn't a commonly used option but can be useful in certain situations. If you want more real estate on your screen, you can go to Tools | Options | Text Editor | General | Display and clear the Vertical Scroll Bar and/or Horizontal Scroll Bar check boxes to make the scroll bars go away.

Before:

```
172 //Create name and age values to initialize Person
173 string[] names = new string[] { "Franscoise", "Bi
174 int[] ages = new int[] { 45, 19, 28, 23, 18, 9, 1
175
176 //Populate the list.
177 for (int x = 0; x < names.Length; x++)
178 {
179     list.AddHead(new Person(names[x], ages[x]));
180 }
181
182
183 Console.WriteLine("Unsorted List:");
100 %
```

After:

```
166
167 static void Main(string[] args)
168 [
169     //Declare and instantiate a new generic SortedList<
170     //Person is the type argument.
171     SortedList<Person> list = new SortedList<Person>();
172
173     //Create name and age values to initialize Person ob
174     string[] names = new string[] { "Franscoise", "Bill"
175     int[] ages = new int[] { 45, 19, 28, 23, 18, 9, 108
176
177     //Populate the list.
178     for (int x = 0; x < names.Length; x++)
```

Of course, you no longer have your scroll bars, which can be somewhat annoying.

## AX.67 How to Convert Tabs to Spaces and Vice Versa

<b>DEFAULT</b>	[no shortcut]
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	[no shortcut]
<b>VISUAL C++ 2</b>	[no shortcut]

<b>VISUAL C++ 6</b>	[no shortcut]
<b>VISUAL STUDIO 6</b>	Ctrl+Q (tabify); Ctrl+Shift+Q (untabify)
<b>WINDOWS</b>	Alt,E, V, T (tabify); Alt,E, V, B (untabify)
<b>MENU</b>	Edit   Advanced   Tabify Selected Lines; Edit   Advanced   Untabify Selected Lines
<b>COMMAND</b>	Edit.TabifySelectedLines; Edit.UntabifySelectedLines; Edit.ConvertSpacesToTabs; Edit.ConvertTabsToSpaces
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEdit0028

Some people prefer spaces; others prefer tabs. You can have it any way you want it with this next item. You can convert spaces to tabs and convert tabs to spaces on your selected lines. You can do this in a couple of ways, and each has different results.

## Tabify/Untabify

If all you want to do is convert leading spaces to tabs (or vice versa), you would use the Tabify/Untabify commands. First, pick a line with some leading spaces, as shown in the following illustration.

### NOTE

You don't have to select the entire line for this to work; as long as any part of the line is selected, it performs the action.

```

11  .....
12  .....Person..... bob = new Person();
13  .....bob.ToString();
14  .....

```

Now go to Edit | Advanced | Tabify Selected Lines.

You should get the leading spaces converted to tabs, as shown in the following illustration:

```

11  .....
12  > > .....Person..... bob = new Person();
13  > > .....bob.ToString();
14  > > .....

```

To change leading tabs to spaces, you would use the Untabify Selected Lines

command.

## ConvertSpacesToTabs/ConvertTabsToSpaces

OK, so what if you want to convert *all* spaces to tabs? Well, you have to use commands that have no shortcut or menu items. The commands you are interested in are Edit.ConvertTabsToSpaces and Edit.ConvertSpacesToTabs.

The following illustration shows what ConvertSpacesToTabs does to our example.

### NOTE

For these commands, you have to select everywhere you want to convert, because the command does not automatically convert the entire line.

```
11 | -----{  
12 | → → ...Person→ → bob⇒=·new·Person();⇒= ...  
13 | → → ...bob.ToString();⇒ → → → → ...  
14 | → → } }
```

As you can see, almost all spaces are converted to tabs. Because spaces are converted to tabs in increments of 4 (default), if you have, say, 6 spaces, it results in a tab and 2 spaces left over. That is why you see some leftover spaces in the example. If you select these same lines and run ConvertTabsToSpaces, it inserts spaces instead of tabs.

## AX.68 Delete Horizontal White Space

<b>DEFAULT</b>	Ctrl+K, Ctrl+\
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+\
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+\; Ctrl+E, Ctrl+\; Ctrl+E, \
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+\
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+\
<b>WINDOWS</b>	Alt,E, V, H
<b>MENU</b>	Edit   Advanced   Delete Horizontal White Space
<b>COMMAND</b>	Edit.DeleteHorizontalWhiteSpace
<b>VERSIONS</b>	2005, 2008, 2010

CODE	vstipEdit0037
------	---------------

Want to get rid of some extra spaces? First find a line that has some extra white space that you want to get rid of:

.....Ti( );.....

Put your cursor in the extra white space:

.....Ti( );.....|.....

Go to Edit | Advanced | Delete Horizontal White Space or press Ctrl+K, Ctrl+\, and the extra space is gone:

.....Ti( );|.....

Interestingly, this also works between items as well. Just find some space and put your cursor in it:

.....{.....|.....}.....

Then get rid of the extra space:

.....{ }.....

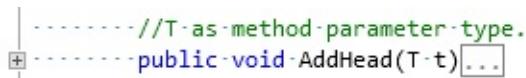
#### NOTE

One space always remains when you use this feature between items.

## AX.69 Expanding Your Code with Outlining

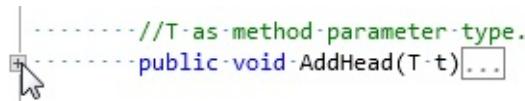
<b>DEFAULT</b>	Ctrl+M, Ctrl+M
<b>VISUAL BASIC 6</b>	Ctrl+M, Ctrl+M
<b>VISUAL C# 2005</b>	Ctrl+M, Ctrl+M; Ctrl+M, M
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+M, Ctrl+M
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt,E, O, T
<b>MENU</b>	Edit   Outlining   Toggle Outlining Expansion
<b>COMMAND</b>	Edit.ToggleOutliningExpansion
<b>VERSIONS</b>	2005, 2008, 2010

By default, outlining is enabled in Visual Studio. Suppose you encounter a collapsed area of code, as shown in the following illustration, and you want to see all the code that is collapsed in that area:



You have three ways to expand it:

- Click the plus sign to expand the area.



- Click anywhere in the area to be expanded, and press Ctrl+M, Ctrl+M.
- Click anywhere in the area to be expanded, and go to Edit | Outlining | Toggle Outlining Expansion on the menu bar.

## AX.70 Collapsing or Expanding All Your Code with Outlining

<b>DEFAULT</b>	Ctrl+M, Ctrl+L
<b>VISUAL BASIC 6</b>	Ctrl+M, Ctrl+L
<b>VISUAL C# 2005</b>	Ctrl+M, Ctrl+L; Ctrl+M, L
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+M, Ctrl+L
<b>VISUAL STUDIO 6</b>	[no shortcut]
<b>WINDOWS</b>	Alt+E, O, L
<b>MENU</b>	Edit   Outlining   Toggle All Outlining
<b>COMMAND</b>	Edit.ToggleAllOutlining
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0031

You can easily collapse or expand all your code with outlining. For example, suppose you have code that is expanded, as shown in the following illustration:

```

using namespace System;

int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hello World");

    Stuff();

    return 0;
}

void Stuff()
{
    // test
}

```

You want it all collapsed, so you have two options:

- Press Ctrl+M, Ctrl+L.
- Go to Edit | Outlining | Toggle All Outlining on the menu bar.

The result is collapsed code:

```

+ Declarations
+ int main(array<System::String ^> ^args) { ... }

+ void Stuff() { ... }

```

Just repeat one of the steps to reverse the process, and all your code is expanded again. This is particularly useful if you are using a feature (certain Find operations) that can't look inside collapsed code.

## AX.71 Turn Off or Turn On Outlining

<b>DEFAULT</b>	Ctrl+M, Ctrl+P (stop outlining)
<b>VISUAL BASIC 6</b>	Ctrl+M, Ctrl+P (stop outlining)
<b>VISUAL C# 2005</b>	Ctrl+M, Ctrl+P (stop outlining); Ctrl+M, P (stop outlining)
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+M, Ctrl+P (stop outlining)
<b>VISUAL STUDIO 6</b>	[no shortcut]

<b>WINDOWS</b>	Alt,E, O, P (stop outlining); Alt, E, O, U (start outlining)
<b>MENU</b>	Edit   Outlining   Stop Outlining; Edit   Outlining   Start Automatic Outlining (Not Available in C++ 2005 and 2008)
<b>COMMAND</b>	Edit.StopOutlining; Edit.OutliningStartAutomaticOutlining
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0033

If you don't like the outlining feature in Visual Studio, you can turn it off one of two ways:

- Press Ctrl+M, Ctrl+P on your keyboard.
- Go to Edit | Outlining | Stop Outlining on your menu bar.

To turn outlining back on, go to Edit | Outlining | Start Automatic Outlining on your menu bar. Unfortunately, no keyboard shortcut is available for turning outlining back on.

#### NOTE

Start Automatic Outlining is not available in C++ 2005/2008. To get it back in C++ 2005/2008, just close and reopen the file you are working on.

## AX.72 Understanding Virtual Space

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   All Languages   General   Settings
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0023

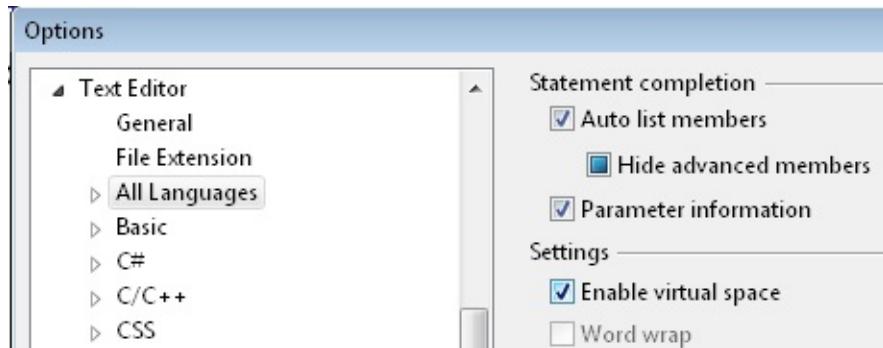
Virtual space is a little difficult to understand if you aren't familiar with older editors. We used to have (and some people still have) editors that treat everywhere on a line as editable space.

Let me explain: Without virtual space, the line ends where the code ends.

If I move my cursor to the end of any line and press my Right Arrow key, it goes to the next line. This is the way editors have been for a while now, and this isn't really new information.

However, this wasn't always the case. There was a time when you could type

anywhere you wanted, anytime you wanted, without restriction. Some text editors still allow this today. Virtual space allows you to go back to the old style of editing, which is preferred by some. Go to Tools | Options | Text Editor | All Languages | General | Settings, and select the Enable Virtual Space check box to turn this feature on:



#### NOTE

Enable Virtual Space and Word Wrap are mutually exclusive options, so you have to choose one or the other.

After you select the Enable Virtual Space option, you can type anywhere on a line, regardless of whether or not the code ends:

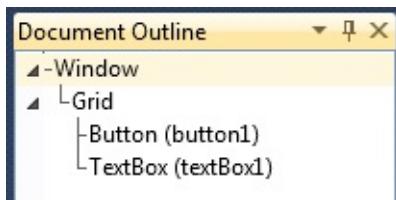
```
int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hello World");
    Stuff();
    return 0;
}
```

## AX.73 Document Outline: WPF and Silverlight Projects

<b>DEFAULT</b>	Ctrl+Alt+T
<b>VISUAL BASIC 6</b>	Ctrl+Alt+T
<b>VISUAL C# 2005</b>	Ctrl+Alt+T; Ctrl+W, Ctrl+U; Ctrl+W, U
<b>VISUAL C++ 2</b>	[no shortcut]

<b>VISUAL C++ 6</b>	Ctrl+Alt+D
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+T
<b>WINDOWS</b>	Alt,V, E, D
<b>MENU</b>	View   Other Windows   Document Outline
<b>COMMAND</b>	View.DocumentOutline
<b>VERSIONS</b>	2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipTool0117

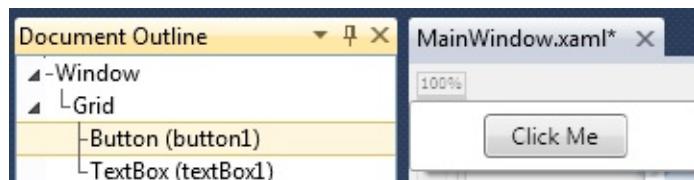
When working with WPF and XAML, it can sometimes get tricky finding items. This is where the Document Outline feature comes in handy. In this example, I've created a WPF Application and put a few controls on it. The following illustration shows what I get when I pull up the Document Outline (Ctrl+Alt+T):



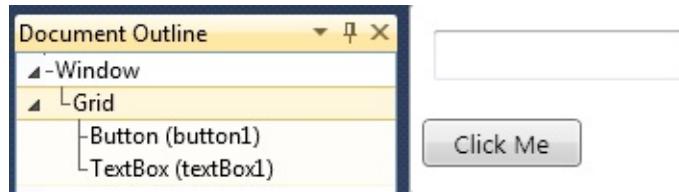
You can also get to the Document Outline by clicking the Document Outline button located in the lower-left corner of the screen by default:



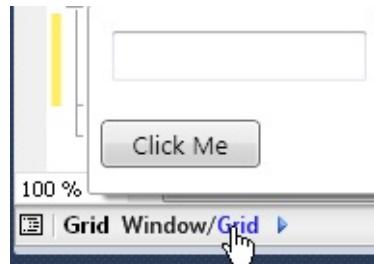
Notice how it shows each control and the parent/child relationships. If the experience stopped there, it would be OK, but it actually gets even better when you put your mouse pointer over any item in the list, as shown in the following illustration:



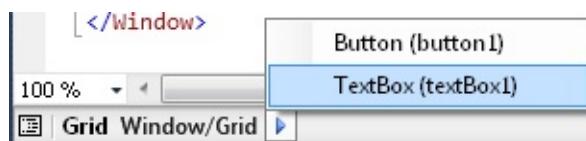
Putting the mouse pointer over a parent shows a preview of the parent and all the children:



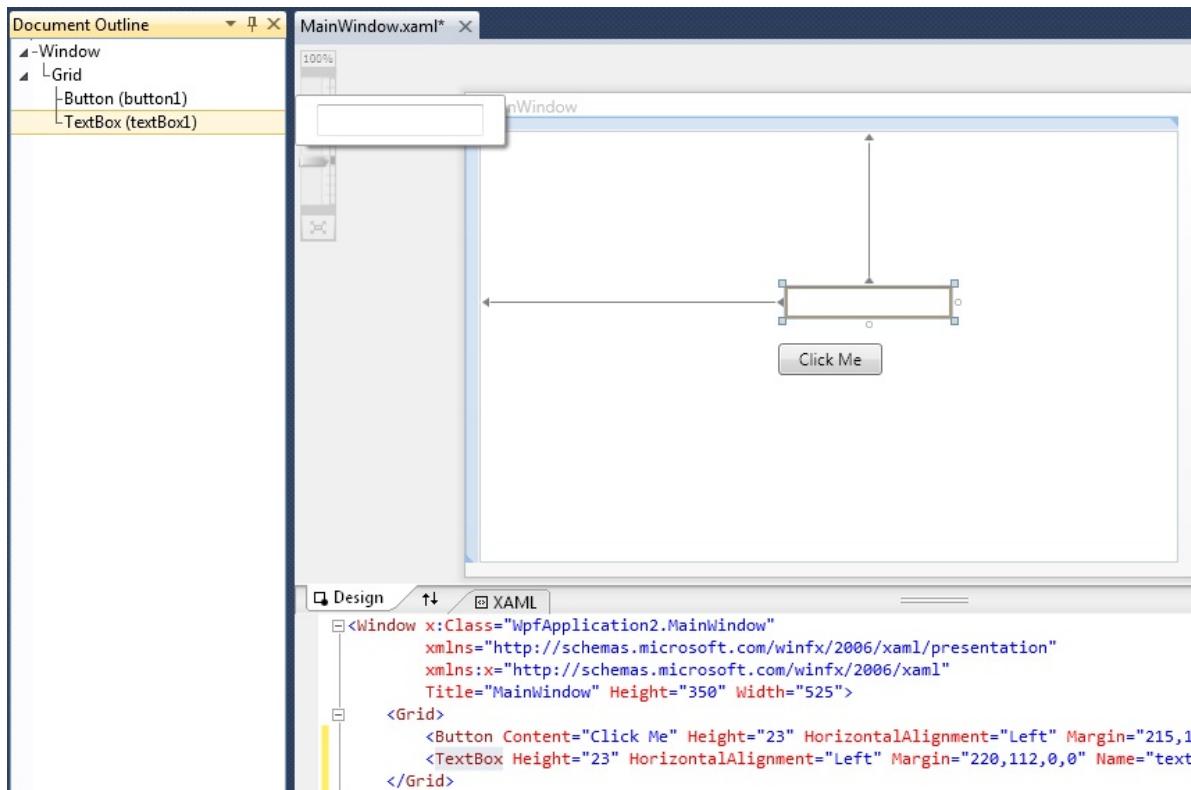
You can also get a preview from the outline presented at the lower part of the screen:



And you can dig into the details if needed:



Also, when you click on any item in the Document Outline, it is selected in both XAML and Design view:



To sum it up, the Document Outline can be used when working with WPF to do the following:

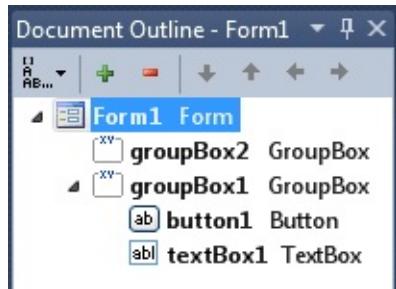
- View the logical structure of elements in your XAML.
- View a thumbnail preview of an element in a pop-up window.
- Navigate to specific elements, in Design view and in XAML view.
- Put user input focus on deeply nested elements that might be hard to select on the design surface itself.
- Locate controls that might be visually hidden by other controls.

## AX.74 Document Outline: Windows Form Projects

<b>DEFAULT</b>	Ctrl+Alt+T
<b>VISUAL BASIC 6</b>	Ctrl+Alt+T
<b>VISUAL C# 2005</b>	Ctrl+Alt+T; Ctrl+W, Ctrl+U; Ctrl+W, U
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+Alt+D
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+T
<b>WINDOWS</b>	Alt,V, E, D
<b>MENU</b>	View   Other Windows   Document Outline
<b>COMMAND</b>	View.DocumentOutline
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0118

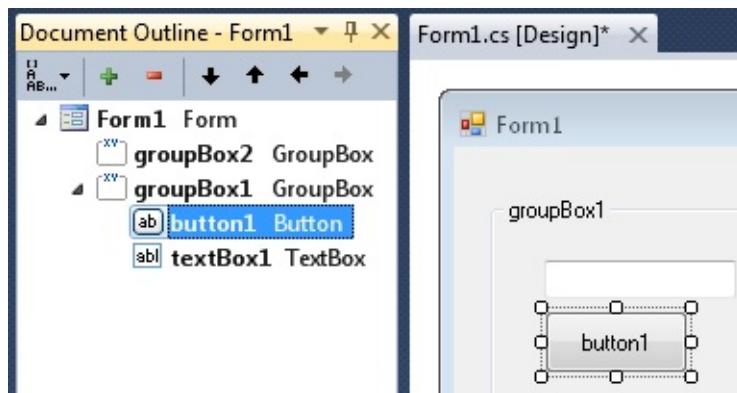
The Document Outline is used to get a bird's eye view of items in your project. Let's look at using it with Windows Form projects.

For this example, I've created a new Windows Forms Application and put a few sample controls on it. The following illustration shows what the Document Outline (Ctrl+Alt+T) looks like:



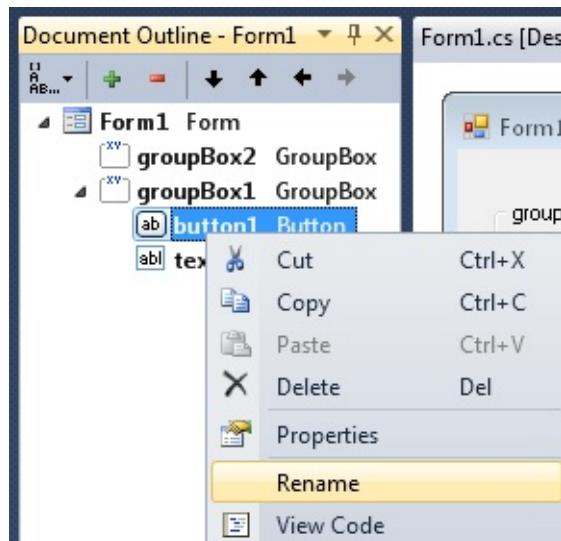
## Selection

When you click any item in the list, that item is selected on the form in Design view:



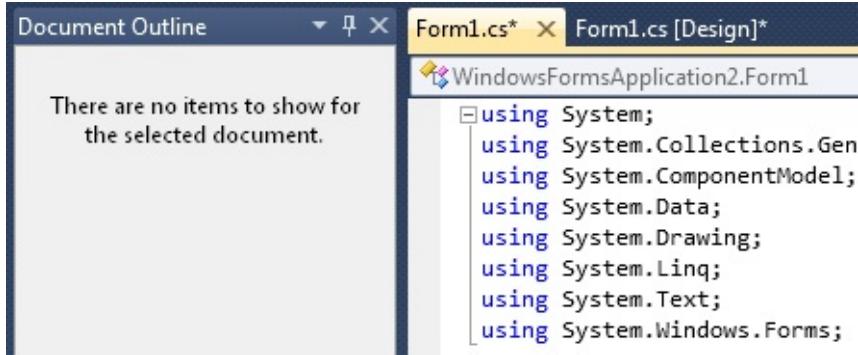
## Context Commands

Additionally, you can access a variety of commands by right-clicking any item, including the ability to rename the control from the Document Outline:



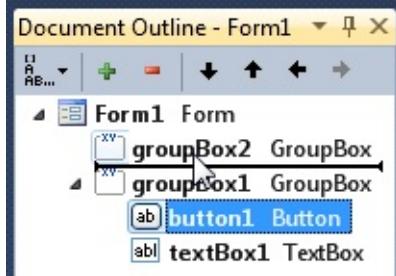
## View Code

You can also press F7 with any item selected to view the code for it, but remember that the Document Outline does not work in code view:



## Relocate Items

The ability to move items from one container to another is supported as well:

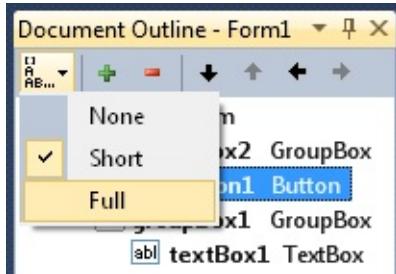


## Toolbar Controls

The toolbar supports a variety of functions as described in the following sections.

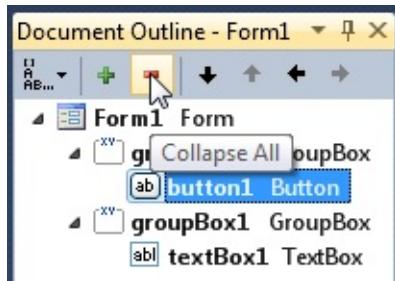
### Name display

You can select different name display styles:



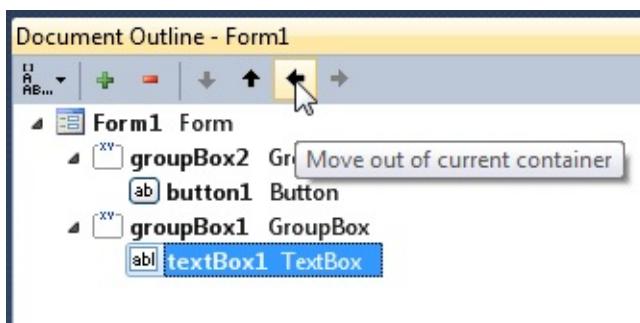
### Expand/collapse

You can expand or collapse the entire outline:



## Moving around

The toolbar even supports moving around within and between containers:



## AX.75 Change the Tooltip Font Size

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Fonts and Colors   Show settings For
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0046

Ever want to change the size of your tooltip font? Here is what it looks like by default:

```
Console.WriteLine();
class System.Console
Represents the standard input,
```

To change it, just go to Tools | Options | Fonts and Colors | Show Settings For. From there, select Editor Tooltip from the drop-down list, as shown in the following illustration:



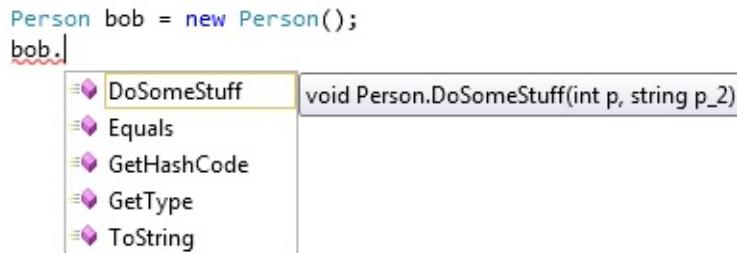
Now that you have the Editor Tooltip settings, change the font to a bigger size and

then click OK. Now you should see a bigger font size on your tooltips.

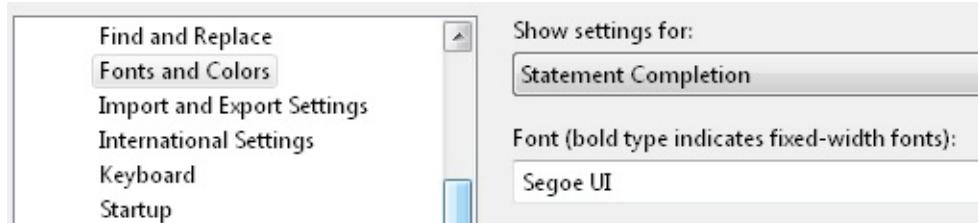
## AX.76 Change the Statement Completion Font Size

WINDOWS	Alt,T, O
MENU	Tools   Options   Fonts and Colors
VERSIONS	2005, 2008, 2010
CODE	vstipEdit0055

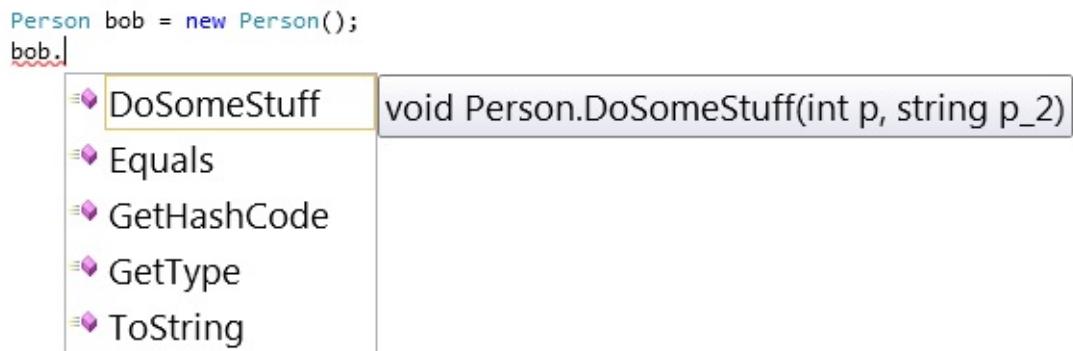
Are you like me and think that the statement completion font is annoyingly small and hard to read?



Well, you can easily increase the font size by going to Tools | Options | Fonts And Colors | Show Settings For | Statement Completion, as shown in the following illustration:



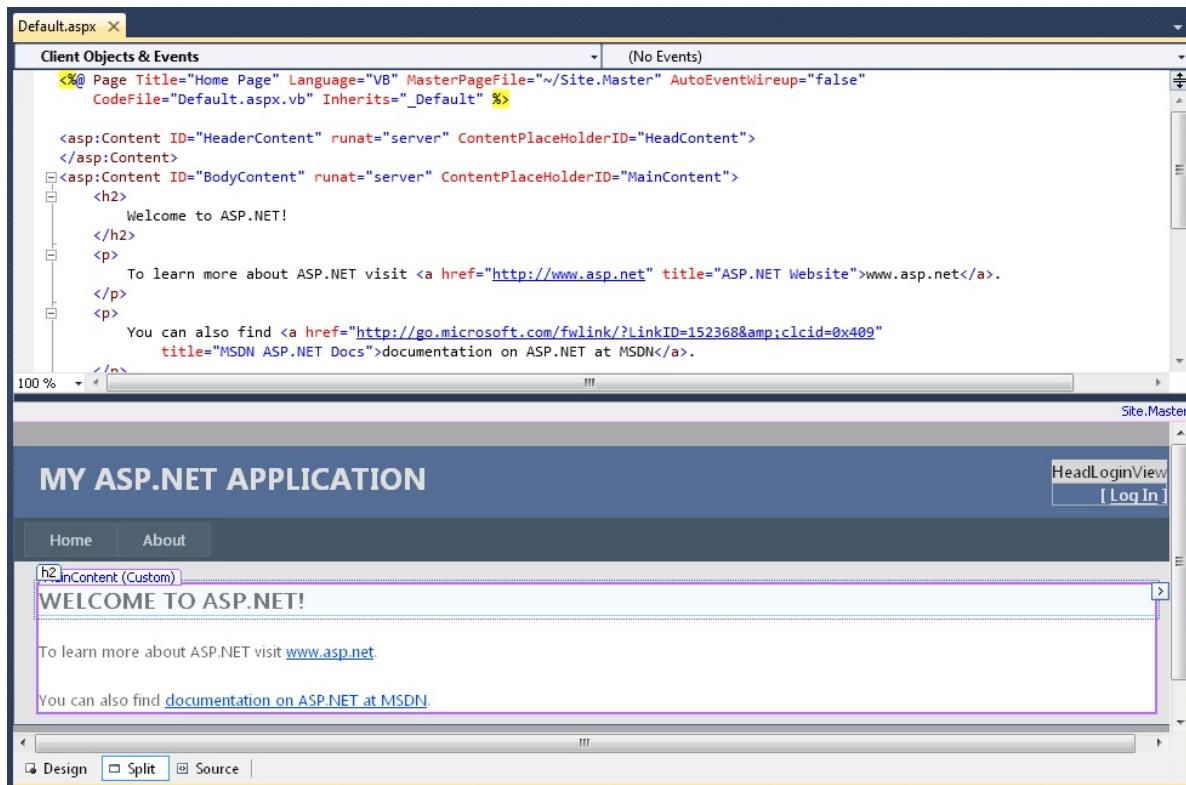
From this dialog box, just make the font any size you like. Click OK to exit the dialog box, and now you should have statement completion in a size you can read:



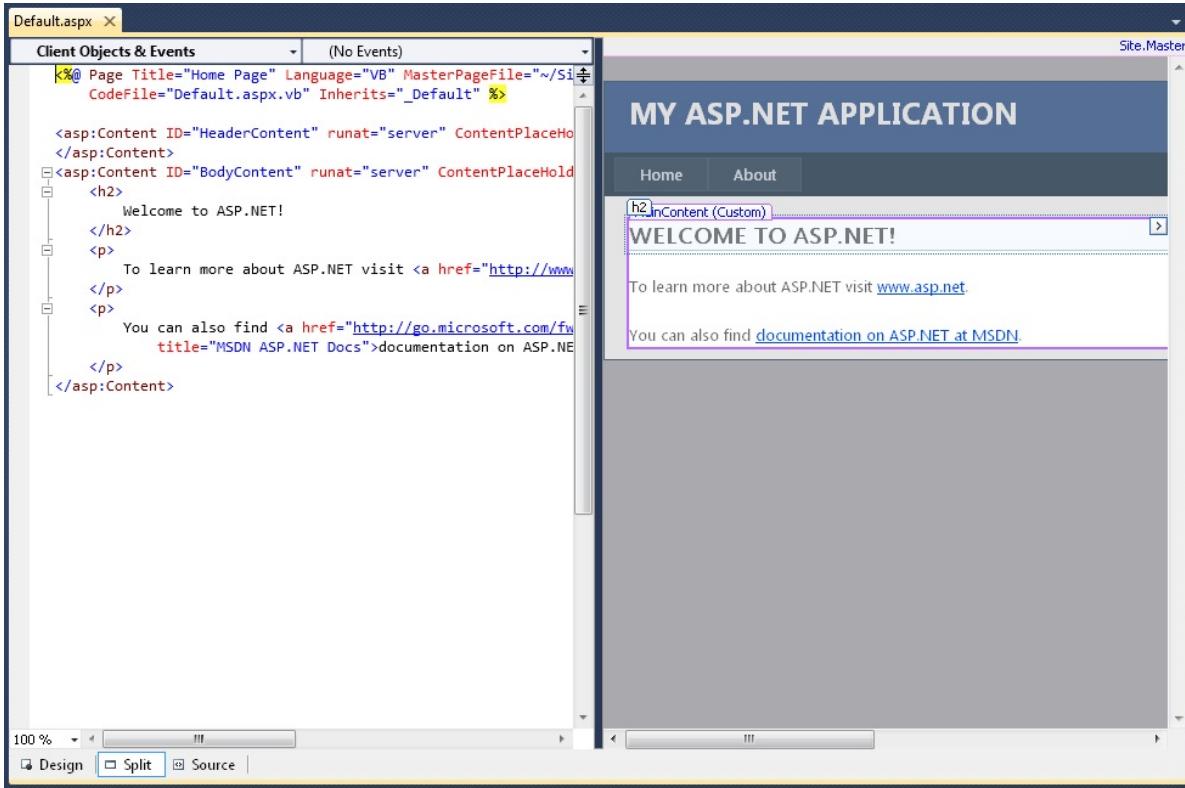
## AX.77 Vertical Split View for Web Projects

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEdit0081

The default way Split view handles panes is to tile them horizontally:



However, you can change this by going to Tools | Options | HTML Designer | General and selecting Split Views Vertically. Now the Split view tiles the panes vertically:



#### NOTE

When you perform this action, you might have to close and reopen some files to see it take effect.

## AX.78 Open JScript Braces on a New Line

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Text Editor   JScript   Formatting
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEdit0087

By default, Visual Studio formats JScript functions and control blocks with the open brace on the same line as the declaration:

```
function Blah() {  
  
};
```

Without getting into the big debate about whether this is good or bad, let's assume you prefer to have your braces inline vertically:

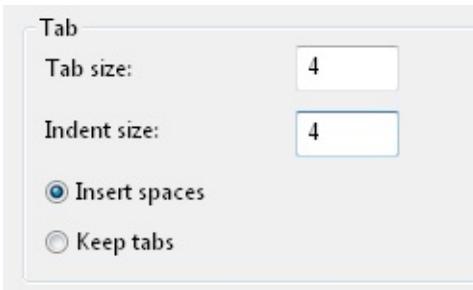
```
function Blah()  
{  
  
};
```

Go to Tools | Options | Text Editor | JScript | Formatting, and choose whether you want the open brace on a new line for functions, control blocks, or both.

## AX.79 Insert Spaces vs. Keep Tabs

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   [Language]   Tab
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0072

Some people like tabs in their code, and others are partial to spaces. You can specify what you want by going to Tools | Options | Text Editor | [Language] | Tab:



### Tab Size

Use this setting to specify the distance in spaces between tab stops. The default is four spaces. Every time you hit the Tab key, it advances the number of spaces specified.

### Indent Size

Use this setting to specify the size in spaces of an automatic indentation. The default is four spaces. Tab characters, space characters, or both are inserted according to the specified size. When the editor automatically indents your code, it uses this setting to determine how much space to use.

## Insert Spaces

Indent operations insert only space characters, not Tab characters. For example, if the indent size is set to 5, five space characters are inserted whenever you press the Tab key or click the Increase Indent button on the formatting toolbar:

..... spaces

## Keep Tabs

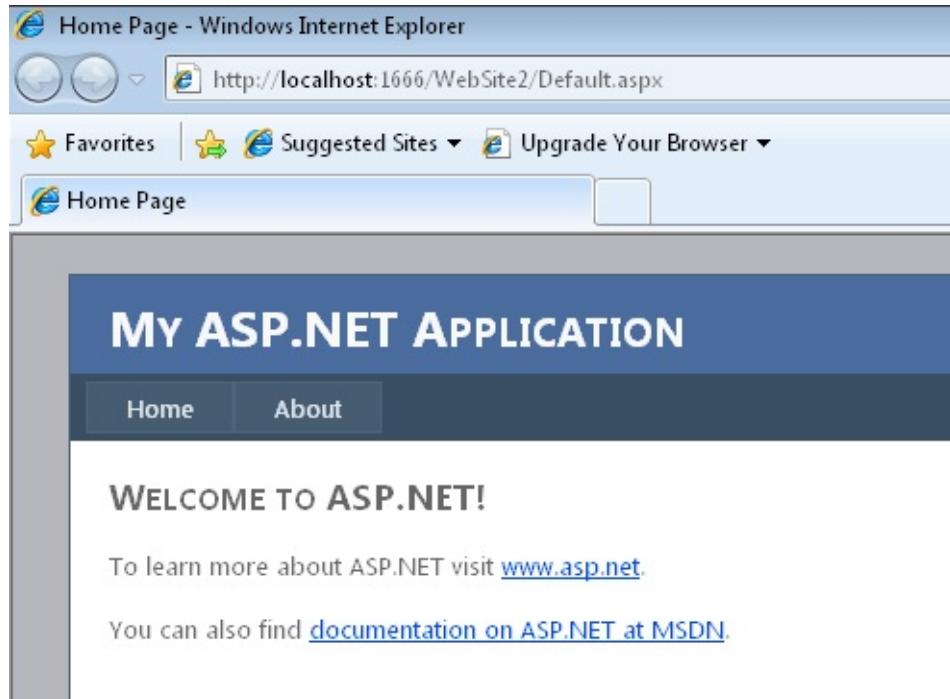
Indent operations insert Tab characters. Each Tab character comprises the number of spaces specified in the Tab Size setting. If the indent size is not an even multiple of the Tab size, space characters are added to make up the difference:

..... tab

## AX.80 View in Browser

<b>DEFAULT</b>	Ctrl+Shift+W
<b>VISUAL BASIC 6</b>	Ctrl+Shift+W
<b>VISUAL C# 2005</b>	[no shortcut]
<b>VISUAL C++ 2</b>	Ctrl+Shift+W
<b>VISUAL C++ 6</b>	Ctrl+Shift+W
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+W
<b>WINDOWS</b>	Alt,F, B
<b>MENU</b>	File   View in Browser
<b>COMMAND</b>	File.ViewinBrowser
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0119

You can quickly view your current page in your browser by pressing Ctrl+Shift+W. This automatically opens up your default browser (see vstipEnv0057, [03.11 Using Additional Browsers for Web Development](#), page 96):



## AX.81 Detect When a File Is Changed Outside the Environment

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options Environment   Documents
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0073

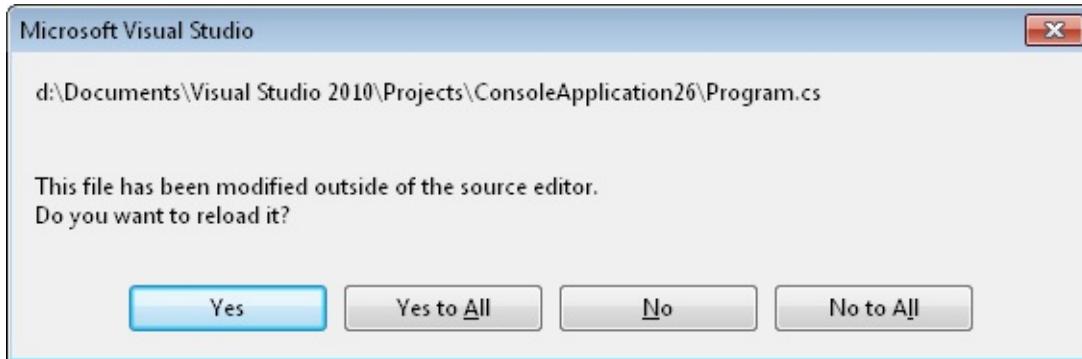
If you go to Tools | Options | Environment | Documents, you can see two interesting options:

- Detect when file is changed outside the environment
- Auto-load changes, if saved

### Detect When File Is Changed Outside The Environment

When this option is selected, a message immediately notifies you of changes to an

open file that have been made by an editor outside the IDE. The message enables you to reload the file from storage:



## Auto-Load Changes, If Saved

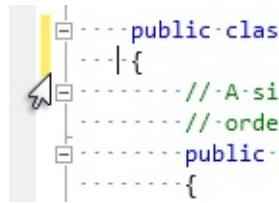
When you have the Detect When File Is Changed Outside The Environment options selected and an open file in the IDE changes outside the IDE, a warning message is generated by default. However, if the Auto-Load Changes, If Saved option is selected, no warning appears and the document is reloaded in the IDE to pick up the external changes.

As you can see, the default is to detect the changes but not auto-load them. This is generally a good strategy because any changes you load from outside the editor should be reviewed, to avoid erasing work you have already in the editor.

## AX.82 Turn Off the Selection Margin

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0034

For those who aren't familiar with it, the selection margin is the area between line numbers and the outline indicators. It is used to show code changes and to enable you to select an entire line of code with one click. The following illustration shows the area in action:



If you don't have use for the change tracking and line selection, you can easily turn this feature off by going to Tools | Options | Text Editor | General | Display and clearing the Selection Margin check box:



Now the selection margin is gone:



An interesting side effect in Visual Studio 2005 and Visual Studio 2008 is that hiding the selection margin also hides the outline area.

Before:

```

7  class Program
8  {
9      static void
10     {
11     }

```

After:

```

7  class Program
8  {
9      static void
10     {

```

## AX.83 Reuse the Same Editor Window When Opening Files

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Documents

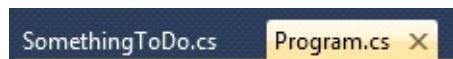
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0027

Normally, when you open up a document (in this example “Program.cs”), it creates a new tab.

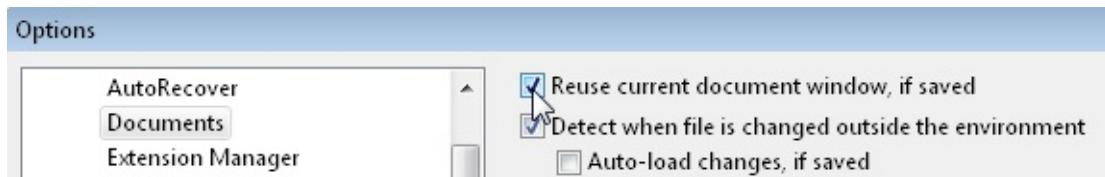
Before:



After:

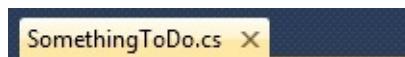


You can reuse the same document window if you want by going to Tools | Options | Environment | Documents and selecting the Reuse Current Document Window, If Saved check box:



Now when you open a document, you see the following results.

Before:



After:



The caveat here is that you must have a saved document for this to work. If the document was not saved, a new document window would be created even with this option turned on.

## AX.84 Sharing Snippets with Your Team

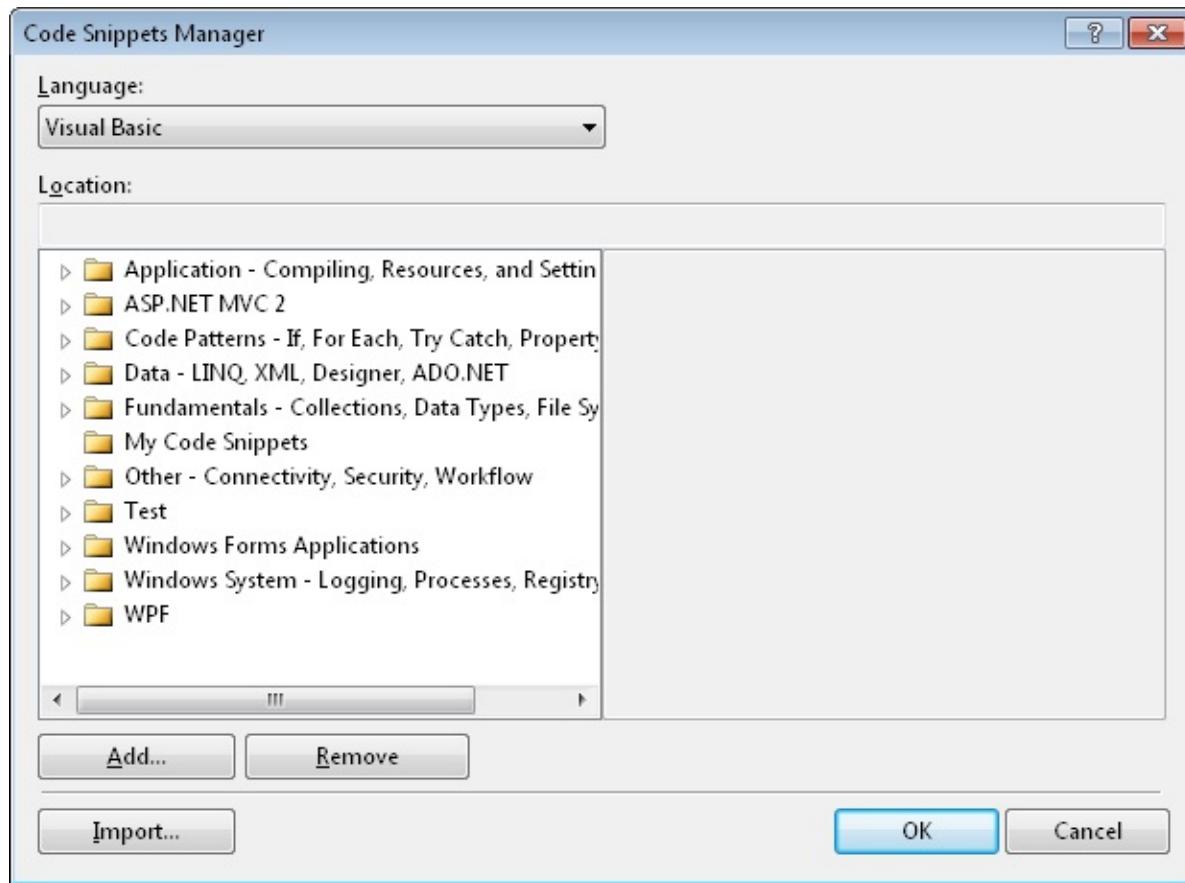
<b>DEFAULT</b>	Ctrl+K, Ctrl+B
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+B
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+B

<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+B
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+B
<b>WINDOWS</b>	Alt,T, T
<b>MENU</b>	Tools   Code Snippets Manager
<b>COMMAND</b>	Tools.CodeSnippetsManager
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0075

Sometimes you might want to share special snippets with your team for others to use. It's easy to do, and it's a great way to ensure common code constructs remain the same.

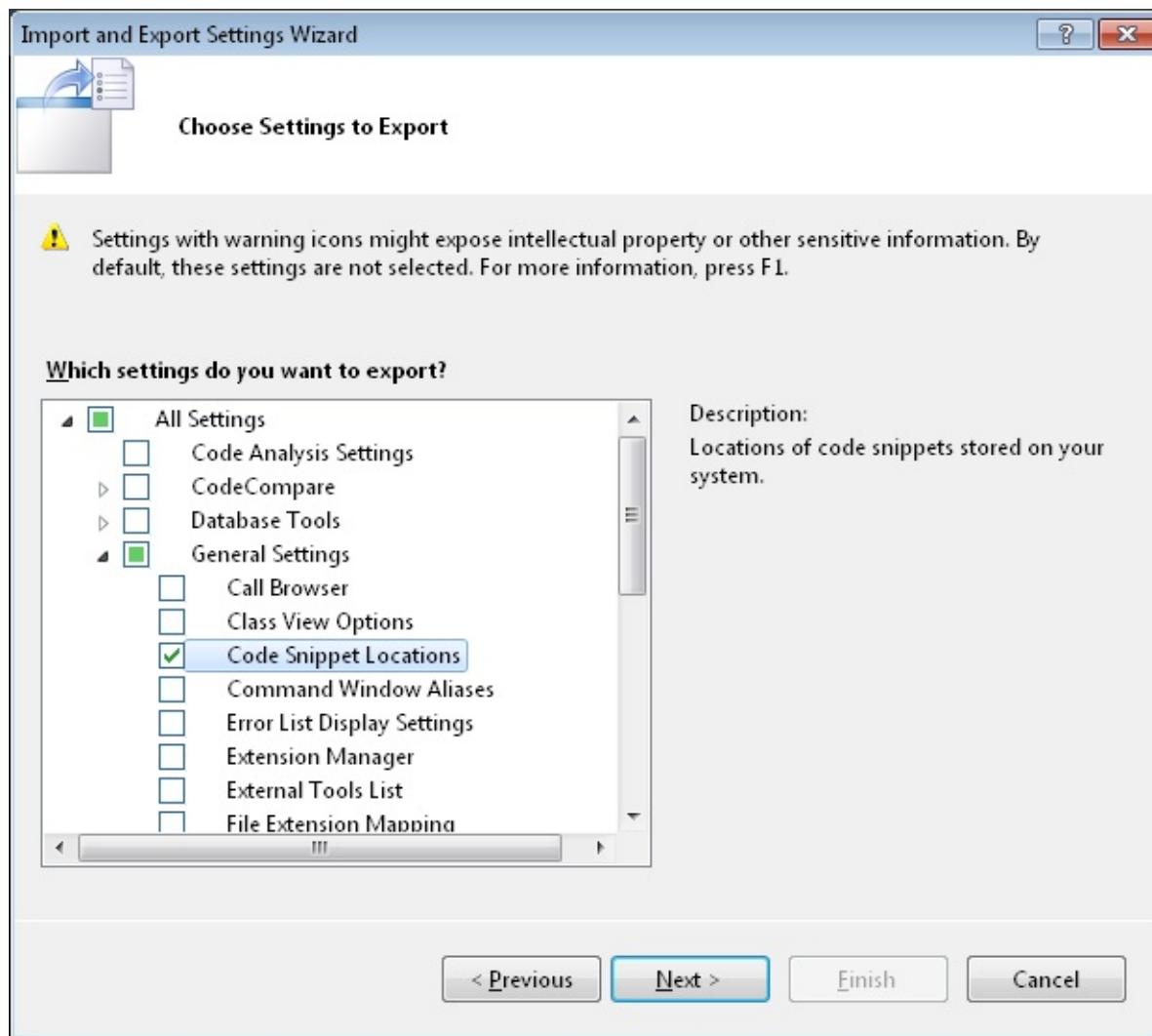
First create a network directory that all the team members can read from, and put all your special .snippet files there. If you don't have experience working with snippet files, take a look at vstipTool0016, [06.54 Create New Code Snippets from Existing Ones](#), page 271.

Next have each team member open the Code Snippets Manager (Ctrl+K, Ctrl+B):



Ask your team members to click Add and select the network directory you created earlier, and everyone can use the shared snippets.

If you don't want to have everyone doing this manually, you can do these steps yourself and export just your code snippet locations, as shown in the following illustration:



Then have everyone import the .vssettings file you created to get the shared location. For more information about exporting, see vstipEnv0021, [01.03 Exporting Your Environment Settings](#), on page 6.

## AX.85 Swap the Current Anchor Position

<b>DEFAULT</b>	Ctrl+K, Ctrl+A
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+A
<b>VISUAL C# 2005</b>	Ctrl+K, Ctrl+A; Ctrl+E, Ctrl+A; Ctrl+E, A
<b>VISUAL C++ 2</b>	Ctrl+Shift+X
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+A
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+A

<b>WINDOWS</b>	[no shortcut]
<b>COMMAND</b>	Edit.SwapAnchor
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0068

When you select text, the “anchor” is the cursor location at the end of the selection. By default, the anchor is to the right of the selection. However, you can use Ctrl+K, Ctrl+A to swap the anchor from the right side to the left side, as shown in the following illustrations:

```
iAnotherParam = 6;
```

```
iAnotherParam = 6;
```

OK, so why would you want to do this? Well, those who use Emacs like this feature quite a bit, and it has been around a long, long time. One great benefit comes when you need to expand a selection from left to right. By swapping the anchor, you can hold down your Shift key and use your Left Arrow key to expand the selection:

```
AnotherParam = 6;
```

## AX.86 Guidelines: A Hidden Feature for the Visual Studio Editor

<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0015

Guidelines are used when you want visible column indicators to help you keep things lined up in the editor:

```

namespace Generics_CSharp
{
    //Type parameter T in angle brackets.
    public class myList<T> : IEnumerable<T>
    {
        protected Node head;
        protected Node current = null;

        // Nested type is also generic on T
        protected class Node
        {
            public Node next;
            //T as private member datatype.
            private T data;
            //T used in non-generic constructor.
            public Node(T t)
            {

```

## Visual Studio 2010

This feature has been removed in Visual Studio 2010. With that said, if you are using Visual Studio 2010, an extension, created by Paul Harrington, is available at <http://visualstudiogallery.msdn.microsoft.com/en-us/0fbf2878-e678-4577-9fdb-9030389b338c>.

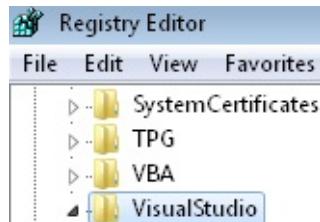
## Visual Studio 2005/2008

To add the guidelines to the user interface, you need to follow these steps:

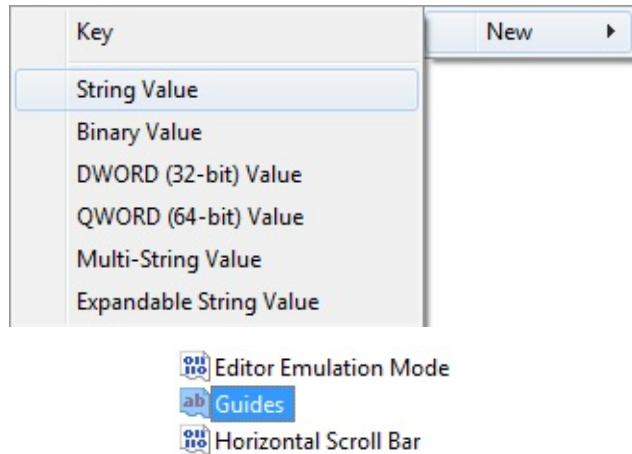
1. Shut down Visual Studio.
2. Go to [HKEY\_CURRENT\_USER]\Software\Microsoft\VisualStudio\<version>\Text Editor in the registry (regedit.exe).

### WARNING

Editing the registry can cause serious problems if you don't know what you are doing, so edit it at your own risk.



3. Create a string value called Guides:



#### 4. Set Guides to the following:

RGB(x,y,z) n1,...,n13, where x,y,z are the RGB color values representing the color you want for the guides; and n is the column number position at which you want the guides to appear.

You can have at most 13 guidelines. For example, RGB(255,0,0) 5, 80 places a red guideline at column numbers 5 and 80:

Editor Emulation Mode	REG_DWORD	0x00000000 (0)
Guides	REG_SZ	RGB(255,0,0) 5, 80

#### 5. Open Visual Studio, and open a file in the Editor:

```
//Type parameter T in angle brackets.
public class MyList<T> : IEnumerable<T>
{
    protected Node head;
    protected Node current = null;

    // Nested type is also generic on T
    protected class Node
    {
        public Node next;
        //T as private member datatype.
        private T data;
        //T used in non-generic constructor.
        public Node(T t)
        {
            next = null;
            data = t;
        }
        public Node Next
        {
            get { return next; }
            set { next = value; }
        }
    }
}
```

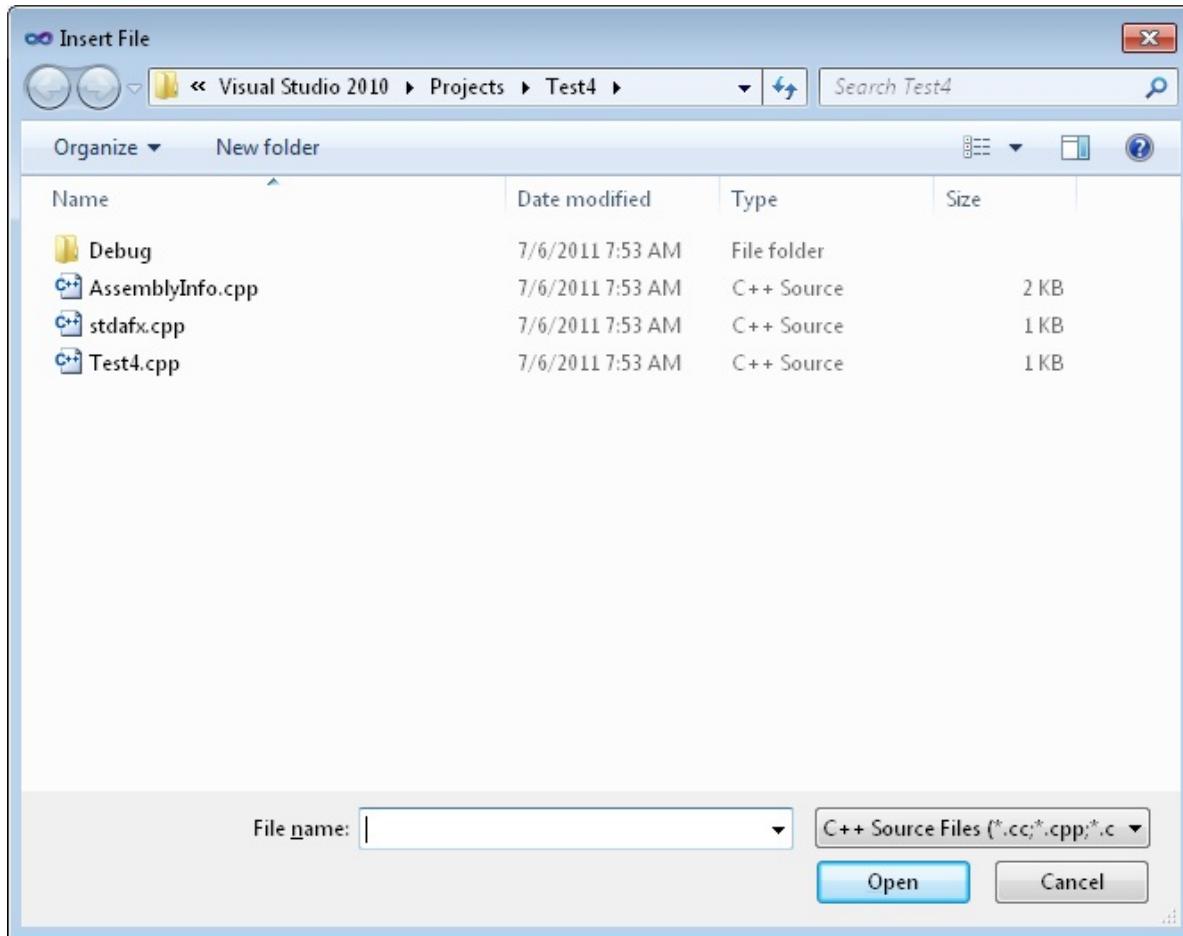
## Removing guidelines

To delete the guidelines, just delete the Guides value you created above and then close and reopen Visual Studio.

## AX.87 Insert File as Text

<b>WINDOWS</b>	Alt,E, X
<b>MENU</b>	Edit   Insert File As Text
<b>COMMAND</b>	Edit.InsertFileAsText
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0021

Another classic item that tends to get overlooked in the wake of snippets and T4 templates is the Insert File As Text feature. Let's say you have a chunk of code in a file, and you want it in another file too. Just go to Edit | Insert File As Text to see the following dialog box:



Choose your file, and it inserts the contents of that file as though you had just typed the text in yourself.

#### NOTE

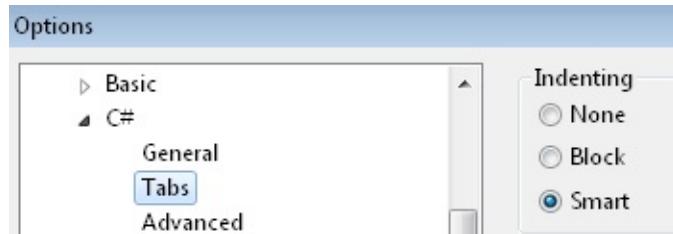
When performing this action, you might need to change the file type to look for.

## AX.88 Indenting: Smart vs. Block vs. None

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Text Editor   [Language]   Tabs   Indenting
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0049

Indenting is something we all deal with in the Editor all the time. In this tip, I

reveal what each of the available indenting choices does for you. If you go to Tools | Options | Text Editor | [Language] | Tabs | Indenting, you can see the options shown in the following illustration:



## Smart

Let's begin with the option that most people have by default: Smart indenting. Create a new method, and press Enter after the curly brace:

```
void Blah()  
{  
|
```

Notice that Smart indenting pays attention to where it is and automatically indents after the opening curly brace. This is the “smart” part of the indenting because it knew that you pressed Enter after an opening curly brace and assumed you wanted to indent.

## Block

Now let's look at Block indenting:

```
void Blah()  
{  
|
```

In this case, the cursor maintains its current indent level and doesn't “smart” indent based on context. Block indenting is the old indenting style that is preferred by people who want control over when indenting happens.

## None

Of course, if you really want total control, you can choose None in the Options dialog box and turn off any indenting at all:

```
void Blah()  
{  
|
```

The cursor returns to column 1 every time you press Enter.

## AX.89 Change CSS Formatting

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Text Editor   CSS   Formatting
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0094

People are very picky about how their code is styled. Fortunately, Visual Studio offers you the ability to format your CSS code the way you like it. Just go to Tools | Options | Text Editor | CSS | Formatting:

### NOTE

In Visual Studio 2005, the menu path is Tools | Options | Text Editor | CSS | Format.



### Style

You have three style options available:

#### Expanded (Default)

Provides the most readability by adding extra space in the styles. The selector and initial brace appear on their own lines, declarations are indented on subsequent lines, and the closing brace is aligned with the matching opening brace:

```
BODY
{
    color: red;
    font-family: Arial;
}
```

## Semi-expanded

Provides a trade-off between readability and compactness by reducing space. The selector and initial brace ({} ) are positioned on the same line, declarations are indented on subsequent lines, and the closing brace (}) is aligned with the matching opening brace:

```
BODY {
    color: red;
    font-family: Arial;
}
```

## Compact rules

Provides maximum amount of reduced space. The selector and declaration are positioned on the same line:

```
BODY { color: red; font-family: Arial; }
```

## Capitalization

This option is pretty straightforward and provides casing instructions for the properties.

### Lowercase (Default)

```
BODY
{
    color: red;
    font-family: Arial;
}
```

### Uppercase

```
BODY
{
    COLOR: red;
    FONT-FAMILY: Arial;
}
```

### As entered

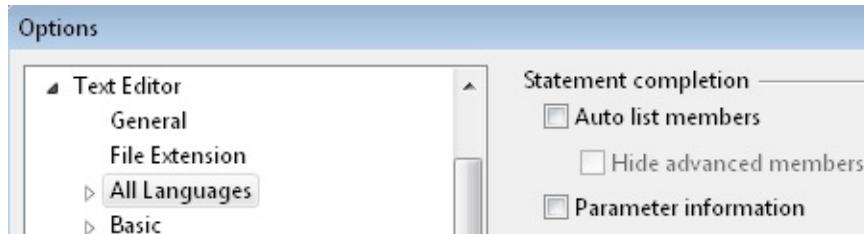
Leaves the casing alone and doesn't modify the user input.

## AX.90 How to Turn Off Automatic IntelliSense

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   All Languages [or specific language]   General   Statement completion
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0054

So IntelliSense might not be for everyone. It can sometimes annoy people when it automatically pops up. You can disable automatic IntelliSense and still have it come up only when you want it to.

Go to Tools | Options | Text Editor | All Languages [or specific language] | General | Statement Completion, and clear the Auto List Members check box:



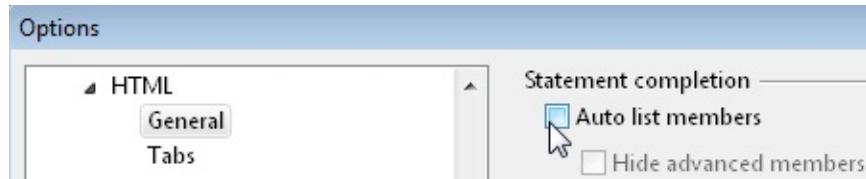
Now anytime you want IntelliSense, just press Ctrl+J to bring it up.

## AX.91 Disable HTML, CSS, or JScript IntelliSense

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Text Editor   HTML   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0085

Personally, I love IntelliSense everywhere I write code.

However, some people don't like it in some places, and one place I find where that is especially true is in the HTML editor. You can turn off IntelliSense by going to Tools | Options | Text Editor | HTML | General and clearing the Auto List Members check box:

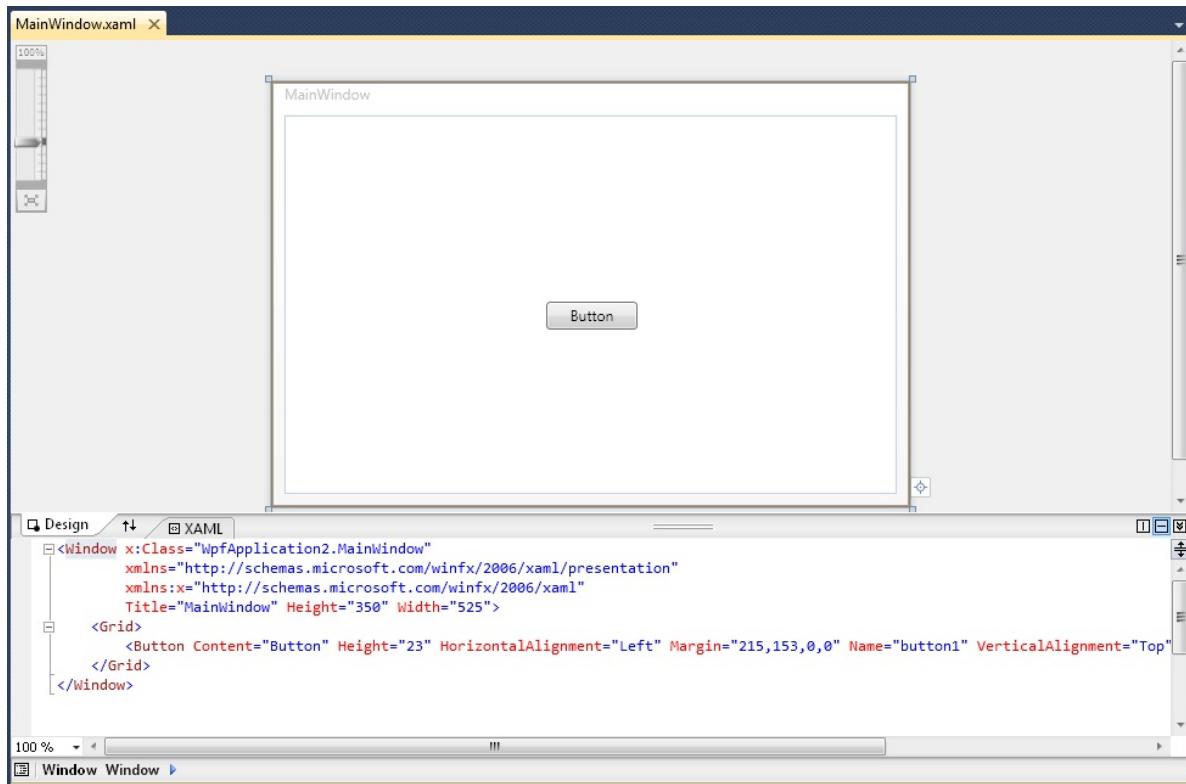


This action disables IntelliSense for the HTML text editor. Additionally, if you want to do the same for CSS or Jscript, just go to Tools | Options | Text Editor | [CSS or JScript] | General and perform the same step.

## AX.92 Design and XAML on Different Document Tabs

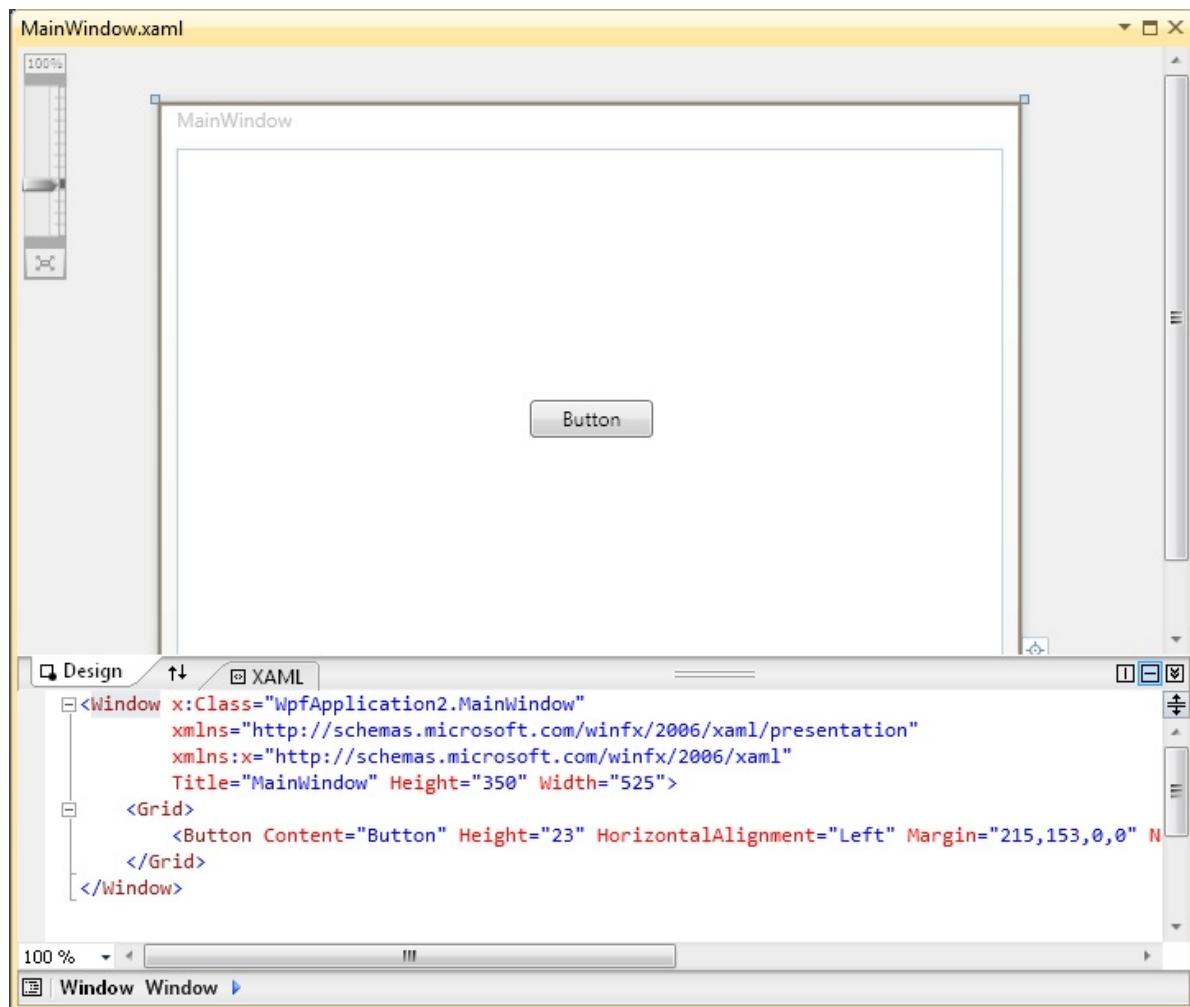
VERSIONS	2010
CODE	vstipTool0009

Now that you can free your document windows and put them on multiple monitors, many people are asking how to do this with code-behind files. Assume you have the following XAML document open:

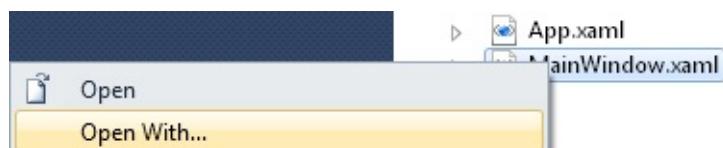


But you want the Design view in one window and the XAML view in another so that you can work on each document in a separate monitor. Well, if you click and

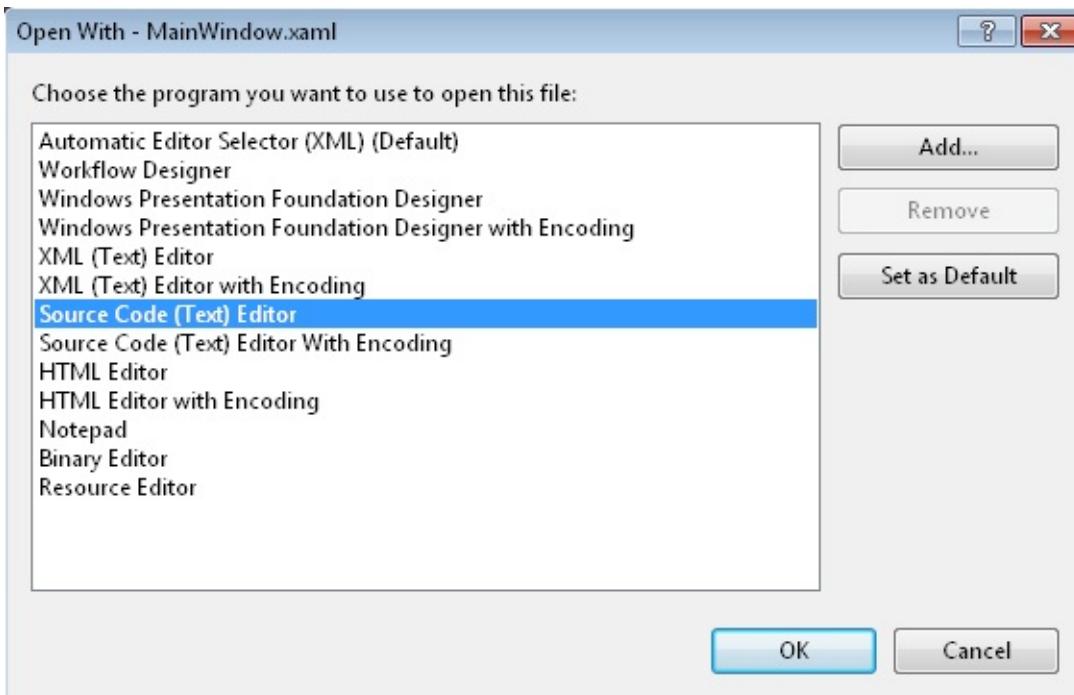
drag the document tab, both the design and the XAML go together:



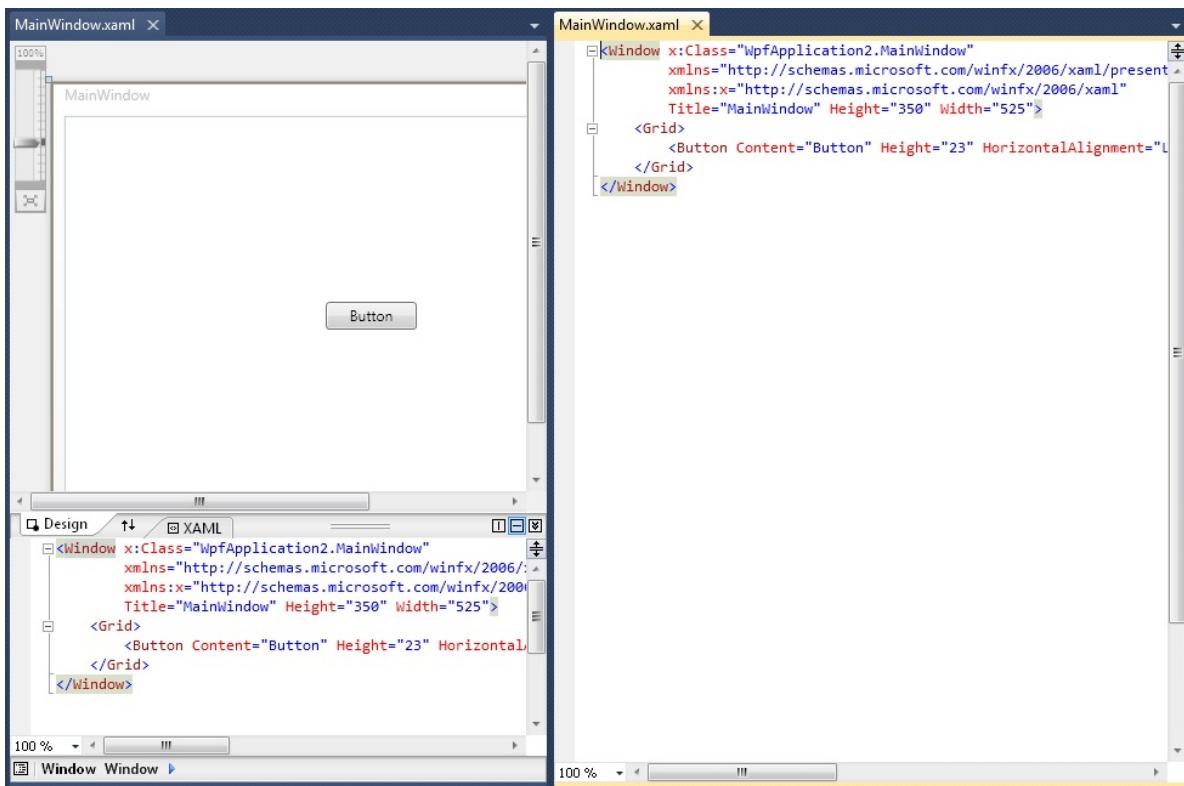
So what do you do? As it turns out, the solution is pretty easy. With the design view in one document, go to Solution Explorer, right-click the XAML file, and choose Open With:



Now, in the Open With dialog box, select Source Code (Text) Editor and click OK:



You now have two XAML windows, one for your designer and one for the XAML:



Now you can put them on two different monitors or whatever you feel like doing.

# AX.93 Using Generate from Usage

<b>WINDOWS</b>	Alt,E, I, G, T
<b>MENU</b>	Edit   IntelliSense   Generate   New Type
<b>COMMAND</b>	Edit.GenerateNewType; EditorContextMenus.CodeWindow.Generate.GenerateNewType
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipEdit0011

I'm a big fan of Test Driven Development (TDD), so I absolutely love this tip because it is a big step toward enabling TDD activities in Visual Studio. The idea behind it is simple; it allows you to use classes and members before you define them. This was created so that you can write your tests and use classes/members that haven't been created yet per the tenets of TDD ("red, green, refactor"). The C# and VB implementations are slightly different, so let's take a look at those differences.

VB

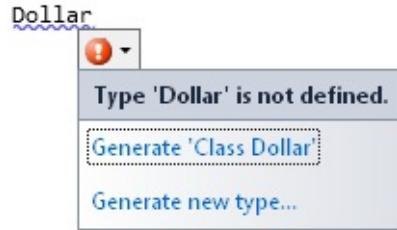
Start by using a class that you haven't created yet:

```
Sub Main()  
  
    Dim myDollar As New Dollar  
  
End Sub
```

Obviously, you will get an error:

```
Sub Main()
    Dim myDollar As New Dollar
End Sub
```

But wait. When you click the Error Correction Options, you get something new:



Click Generate ‘Class Dollar’, and you get a new file called Dollar.vb with a class stub inside:

```
Class Dollar  
  
End Class
```

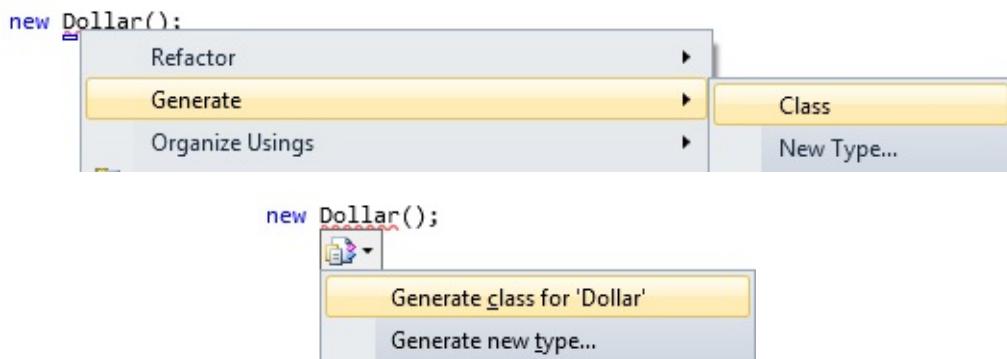
The error is gone, and you are ready to start using the class. You can repeat this process for new members that you create for the new class, as you use them.

## C#

Start by using a class that doesn't exist:

```
static void Main(string[] args)  
{  
    Dollar myDollar = new Dollar();  
}
```

Here you have a couple of options. You can right-click and choose Generate | Class, or you can click the smart tag and choose Generate class for ‘Dollar’. They both do the same thing, and you can see both options in the following illustrations:

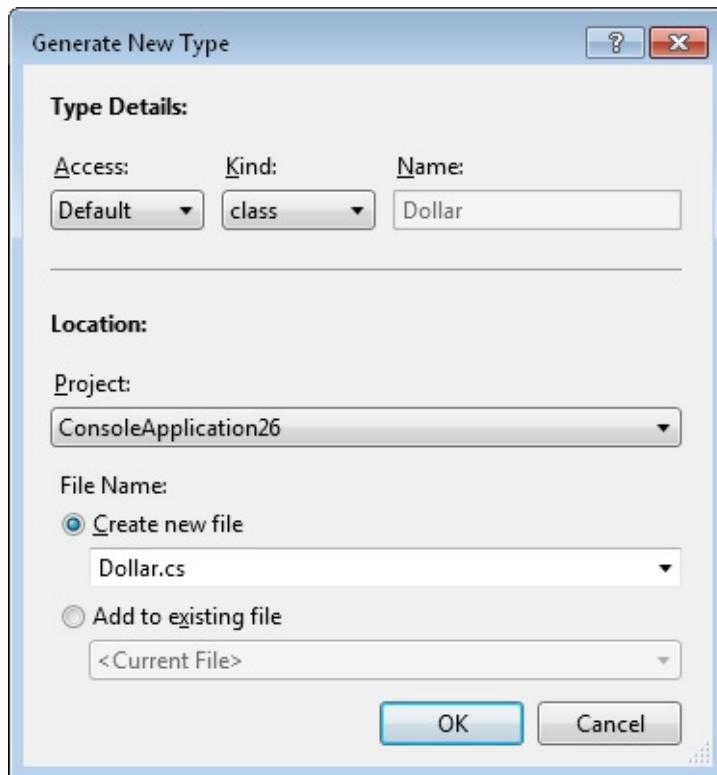


You get a new file called Dollar.cs with a class stub inside:

```
class Dollar  
{  
}
```

The error is gone, and you are ready to start using the class. You can repeat this process for new members that you create for the new class, as you use them.

When you get the hang of this and actually start using this feature for TDD activities, you would not use the examples I provided but would instead choose the Generate New Type option. The dialog box is the same for VB and C#.



Notice that you have the ability to set Access, Kind, and—most importantly—Location. It's the Location option that TDD folks will use to put the classes into the proper project outside of their test projects.

## AX.94 IntelliSense Suggestion Mode

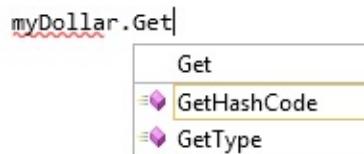
<b>DEFAULT</b>	Ctrl+Alt+Spacebar
<b>WINDOWS</b>	Alt,E, I, T
<b>MENU</b>	Edit   IntelliSense   Toggle Completion Mode
<b>COMMAND</b>	Edit.ToggleCompletionMode
<b>VERSIONS</b>	2008, 2010
<b>CODE</b>	vstipEdit0012

IntelliSense comes in two modes: Completion and Suggestion. You are already familiar with IntelliSense completion mode; it's the traditional mode that we have all used for years. But if you are into Test Driven Development (TDD), completion mode can be very annoying at times.

TDD developers often use classes and members before they exist. It's not fun when you go to type the name of something that doesn't exist and you get IntelliSense. Especially because you sometimes accidentally get an option you didn't want:



The solution is suggestion mode. Just press **Ctrl+Alt+Spacebar** to go into this mode:

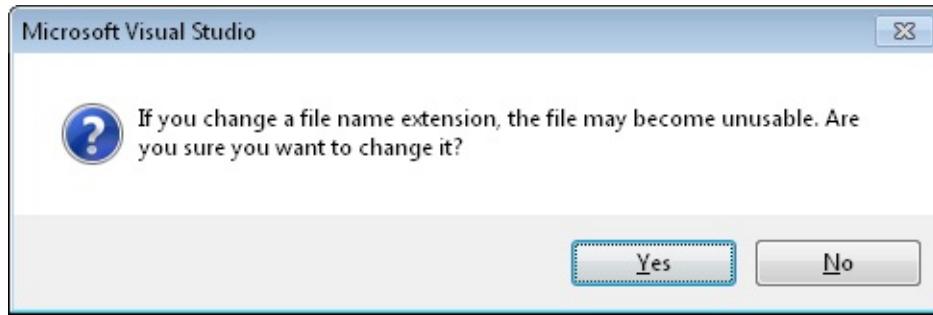


Now you get the best of both worlds: You can type in a name that doesn't exist and have quick access to the completion mode options as well. The risk of getting an option you don't want is reduced.

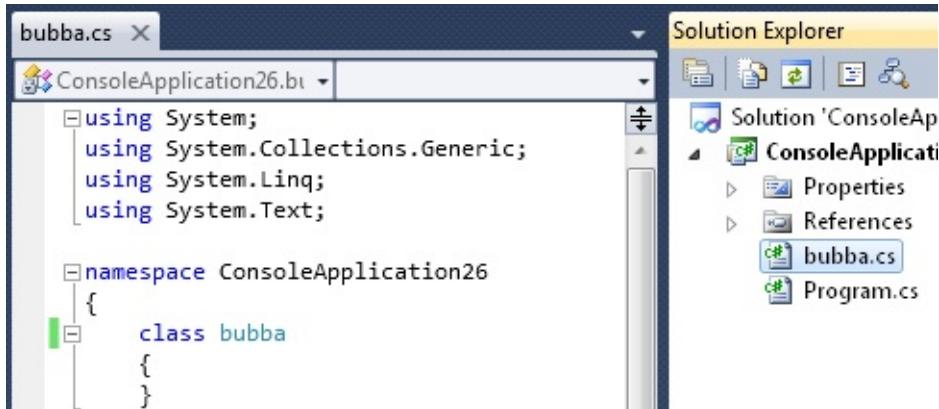
## AX.95 Turn Off Automatic Symbol Renaming When You Rename a File in Solution Explorer

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Projects And Solutions   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0011

When you start to rename a class file in Solution Explorer, you receive the following prompt:



When you click Yes, both the class name in all your code and the file are renamed:



If you find yourself doing this a lot, you can get rid of the prompt to rename by going to Tools | Options | Projects And Solutions | General and clearing the Prompt For Symbolic Renaming When Renaming Files check box.

From now on, when you do your renaming, the prompt does not appear and everything is renamed automatically.

## AX.96 Mark Methods and Types as Hidden from IntelliSense and the Object Browser

VERSIONS	2005, 2008, 2010
CODE	vstipTool0088

When you make assemblies for others to use, you might sometimes want to have hidden methods, properties, and other elements. To hide an item, you use the System.ComponentModel namespace.

**C#**

**C#**

```
using System.ComponentModel;
```

## VB

```
Imports System.ComponentModel
```

## Hiding

To hide a property or method, you use the EditorBrowsable(EditorBrowsableState.Never) attribute.

## C#

```
public class Person
{
    public int Age { get; set; }

    [EditorBrowsable(EditorBrowsableState.Never)]
    public string Name { get; set; }

    [EditorBrowsable(EditorBrowsableState.Never)]
    public void DoStuff()
    {
        //todo
    }
}
```

## VB

```
Public Class Person

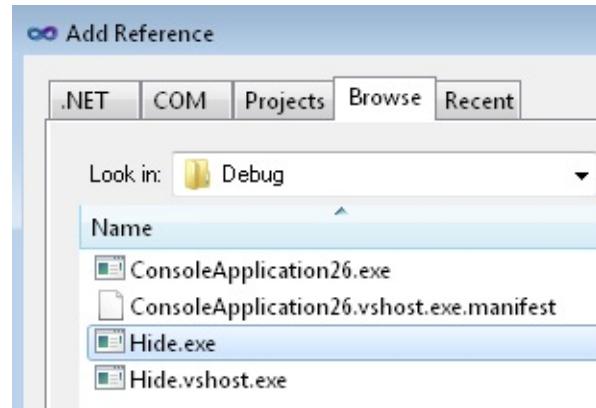
    Public Property Age As Integer

    <EditorBrowsable(EditorBrowsableState.Never)>
    Public Property Name As String
    _____
    <EditorBrowsable(EditorBrowsableState.Never)>
    Public Sub DoStuff()
        'todo stuff
    End Sub

End Class
```

## Use

Testing this is a little tricky. You have to use the assembly externally because hiding is meant to hinder external users, *not* the creator of the assembly. Just making a reference to another project isn't enough. You actually have to go to the Browse tab and set a reference to the external assembly that is created:



## Result

When you test it, the hidden items should not be visible.

C#

```
class Program
{
    static void Main(string[] args)
    {
        Person bob = new Person();

        bob.
    }
}
```

A screenshot of the C# code editor. The cursor is at the end of the line 'bob.'. An intellisense dropdown appears, showing the 'Age' property highlighted. Other members listed are Equals, GetHashCode, GetType, and ToString. A tooltip on the right says 'int Person.Age'.

VB

```
Sub Main()
    Dim bob As New Person
    bob.
End Sub
```

A screenshot of the VB code editor. The cursor is at the end of the line 'bob.'. An intellisense dropdown appears, showing the 'Age' property highlighted. Other members listed are Common and All. A tooltip on the right says 'Public Property Age As Integer'.

## Additional Tips from Chapter 7

### AX.97 Set or Remove a Breakpoint

<b>DEFAULT</b>	F9
<b>VISUAL BASIC 6</b>	F9
<b>VISUAL C# 2005</b>	F9
<b>VISUAL C++ 2</b>	F9; Ctrl+Shift+F9
<b>VISUAL C++ 6</b>	F9
<b>VISUAL STUDIO 6</b>	F9
<b>WINDOWS</b>	Alt,D, G
<b>MENU</b>	Debug   Toggle Breakpoint
<b>COMMAND</b>	Debug.ToggleBreakpoint
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0016

Setting a simple breakpoint is easy. Just find some code:

```
11 static void Main(string[] args)
12 {
13     Console.WriteLine("boogie fever");
14     Console.WriteLine("disco inferno");
15 }
16
```

Now set (or remove) a breakpoint either by pressing F9 or by selecting Debug | Toggle Breakpoint on your menu bar:

```
11 static void Main(string[] args)
12 {
13     Console.WriteLine("boogie fever");
14     Console.WriteLine("disco inferno");
15 }
16
```

If you prefer using the mouse, you can set a breakpoint by positioning your pointer in the margin next to the line on which you want to set the breakpoint:



```
11 static void Main(string[] args)
12 {
13     Console.WriteLine("boogie fever");
14     Console.WriteLine("disco inferno");
15 }
16
```

Then click your left mouse button:



```
11 static void Main(string[] args)
12 {
13     Console.WriteLine("boogie fever");
14     Console.WriteLine("disco inferno");

```

At Program.cs, line 14 character 1 ('CS\_Clean\_Console.Program.Main(string[] args)', line 4)

```
16
```

All the breakpoints you have set show up in the Breakpoints window as a list that you can work with later on.

## AX.98 Enable or Disable a Breakpoint

<b>DEFAULT</b>	Ctrl+F9
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>COMMAND</b>	Debug.EnableBreakpoint
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0017

Sometimes you want to keep a breakpoint around but need to temporarily disable it. This is easy to do and requires only a slight muscle memory modification from setting and removing a breakpoint. Just put your cursor on the line with the breakpoint you want to disable, then press Ctrl+F9 to disable the breakpoint (notice that the glyph in the margin changes):

```

11  static void Main(string[] args)
12  {
13      Console.WriteLine("boogie fever");
14      Console.WriteLine("disco inferno");
15  }

```

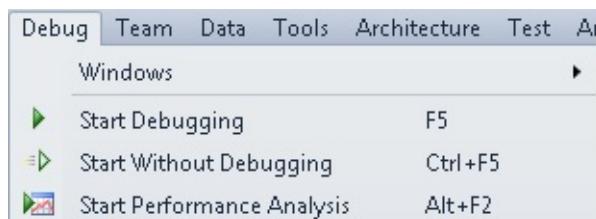
To re-enable it, press Ctrl+F9 again.

All your disabled breakpoints are easily visible in the Breakpoints window alongside your enabled breakpoints so that you can work with them later.

## AX.99 Start Debugging vs. Start Without Debugging

<b>DEFAULT</b>	F5 (start); Ctrl+F5 (start w/o debug)
<b>VISUAL BASIC 6</b>	F5 (start); Ctrl+F5 (start w/o debug); Ctrl+Alt+Break (stop)
<b>VISUAL C# 2005</b>	F5 (start); Ctrl+F5 (start w/o debug); Shift+F5 (stop)
<b>VISUAL C++ 2</b>	F5 (start); Ctrl+F5 (start w/o debug); Alt+F5 (stop)
<b>VISUAL C++ 6</b>	F5 (start); Ctrl+F5 (start w/o debug); Shift+F5 (stop)
<b>VISUAL STUDIO 6</b>	F5 (start); Ctrl+F5 (start w/o debug); Shift+F5 (stop)
<b>WINDOWS</b>	Alt,D, S (start); Alt,D, H (start w/o debug)
<b>MENU</b>	Debug   Start Debugging; Debug   Start Without Debugging
<b>COMMAND</b>	Debug.Start; Debug.StartWithoutDebugging
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0037

There seems to be a great deal of confusion as to what actually happens when you use Start Debugging (F5) versus Start Without Debugging (Ctrl+F5):



## Starting with Debugging

Let's start with the basics: When you press F5 (Start Debugging) in Visual Studio, it launches your application, attaches the debugger, and lets you do all the "normal" things you would expect.

According to the documentation (see "Debugger Roadmap" at [http://msdn.microsoft.com/en-us/library/k0k771bt\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/k0k771bt(v=VS.100).aspx)), here is what the debugger does:

*"The Visual Studio debugger is a powerful tool that allows you to observe the run-time behavior of your program and locate logic errors. The debugger works with all Visual Studio programming languages and their associated libraries. With the debugger, you can break, or suspend, execution of your program to examine your code, evaluate and edit variables in your program, view registers, see the instructions created from your source code, and view the memory space used by your application."*

## The debugger: Release builds

One popular misconception is that the debugger doesn't come into play for release builds. This isn't true. Set a breakpoint in some code for a release build, and press F5 to see whether it stops there. Of course it does!

The debugger is attached. Now, what about things that aren't happening?

### NOTE

Release builds are optimized versions of your code, so make sure to test your breakpoint on a piece of code that is used.

For example, you can't use the `System.Diagnostics.Debug` class methods to send messages to the Output window because the compiler strips them out for release builds:

```
System.Diagnostics.Debug.WriteLine  
    ("This is a debug message.");
```

## Starting Without Debugging

This is exactly what it sounds like. When you choose the Start Without Debugging option, the application starts without the debugger attached. That's it! Nothing else happens. It just doesn't attach the debugger. Otherwise, everything else is the same. So, the practical implications of this are obvious: Without the debugger attached, when the application runs, it does not hit breakpoints, emit debug messages, and so on.

So now let's deal with the biggest myth about choosing the Start Without Debugging option.

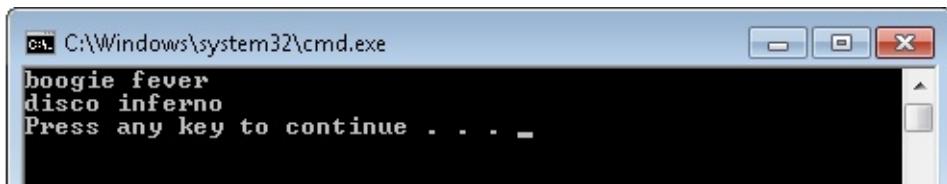
## Myth: Start Without Debugging creates a release build

Not true. It uses the build you are currently on. So if you are using a debug build and you press Ctrl+F5, Visual Studio runs that debug build. The easiest way to test this is to use conditional compilation to see whether you are using a debug build:

```
#if DEBUG
    Console.WriteLine("boogie fever");
    Console.WriteLine("disco inferno");

    System.Diagnostics.Debug.WriteLine
        ("This is a debug message.");
#endif
```

In this example, the Console statements run, but the System.Diagnostics.Debug statement does not run because the debugger is not attached. So you get the following output:



## Finally

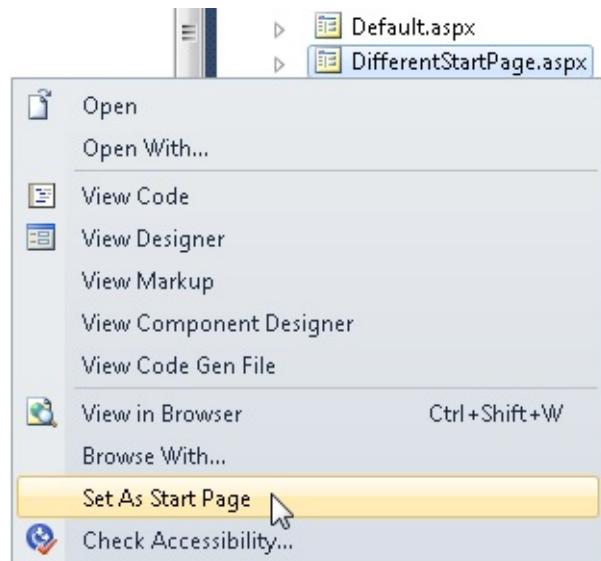
OK, so the obvious question is “Why have this option?” Well, the answer is simple: So that you can run the application without having the overhead of the debugger attached, to do a quick “Smoke Test” to see whether the code runs. As you learn more about this feature, you might find other reasons for wanting to run without the debugger attached.

Now you have a better idea of the difference between the Start Debugging and Start Without Debugging options.

## AX.100 Set As Start Page

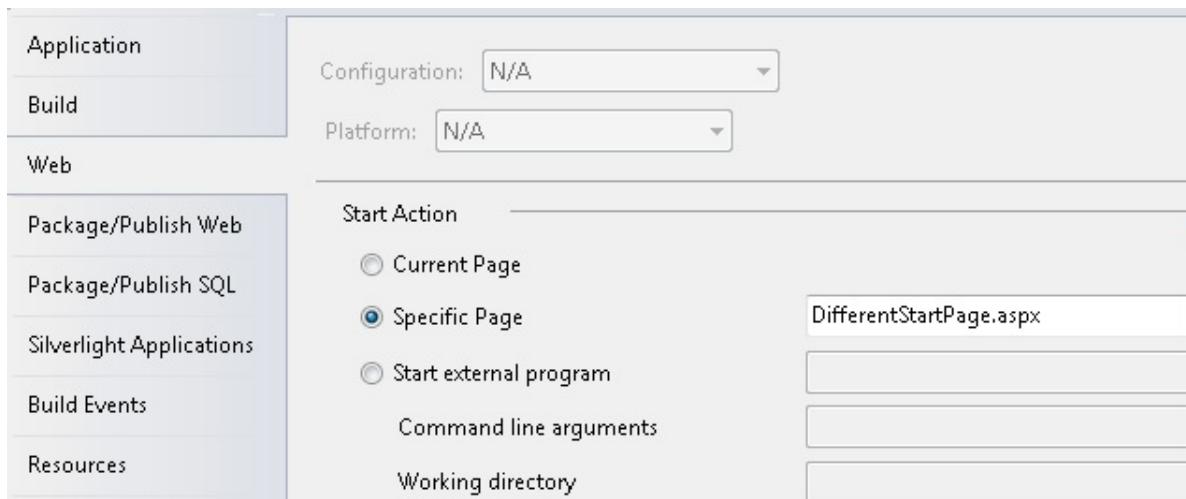
<b>WINDOWS</b>	ALT, P, P, P, Enter
<b>MENU</b>	Project   Set As Start Page
<b>COMMAND</b>	Project.SetAsStartPage
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0027

When you create a new web project, the default start page is set for you automatically and is the first page you see when you start debugging. However, you might find that you want a different page to start with instead. It's easy to change the start page. Simply right-click the new start page in Solution Explorer, and select Set As Start Page:

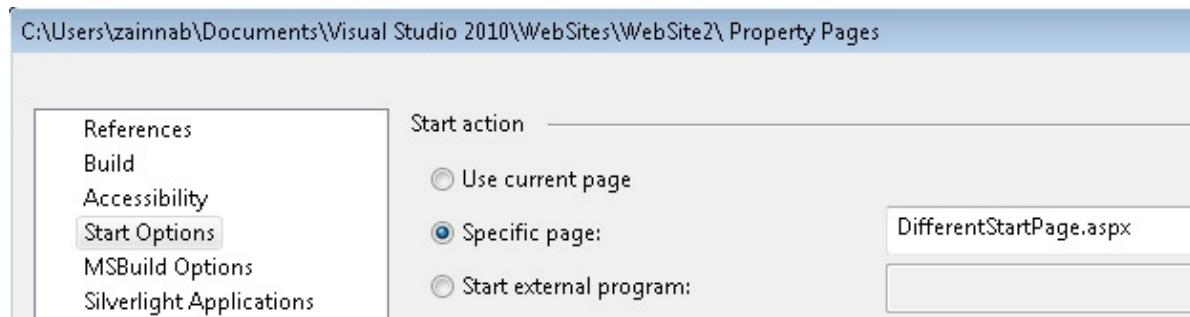


The changes take effect immediately, and the new page becomes the start page until you change it.

In case you are curious, this actually changes the Specific Page value in your Web Application properties dialog box (Web Tab, Start Action):



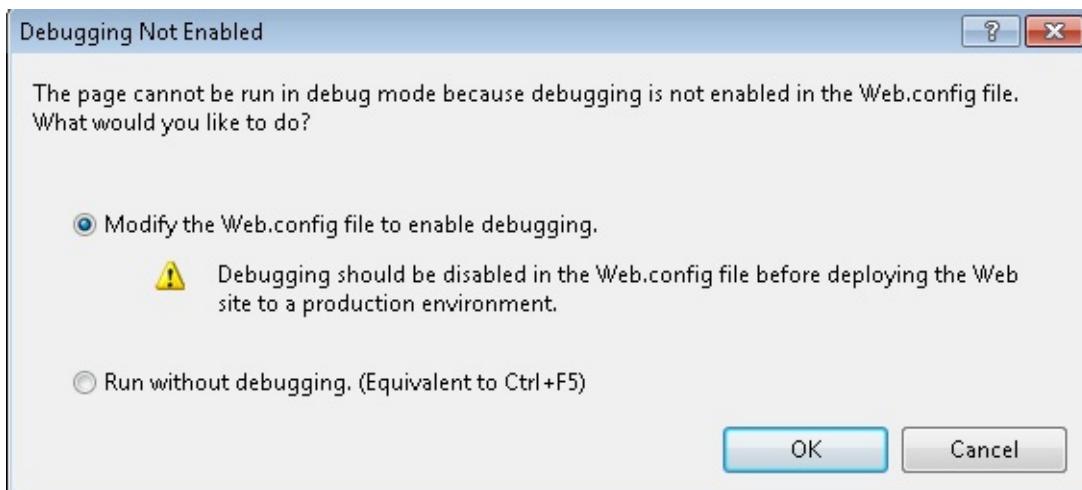
For websites, you can find this in the project properties dialog box under Start Options:



## AX.101 Enable Debugging in Web.Config

VERSIONS	2008, 2010
CODE	vstipProj0026

If you have ever created a web project in Visual Studio, you have undoubtedly encountered the following dialog box:



A quick read tells you that it defaults to the Modify The Web.config File To Enable Debugging option:

```
<system.web>
  <compilation debug="true"
    targetFramework="4.0"/>
```

A couple of things need to be pointed out here:

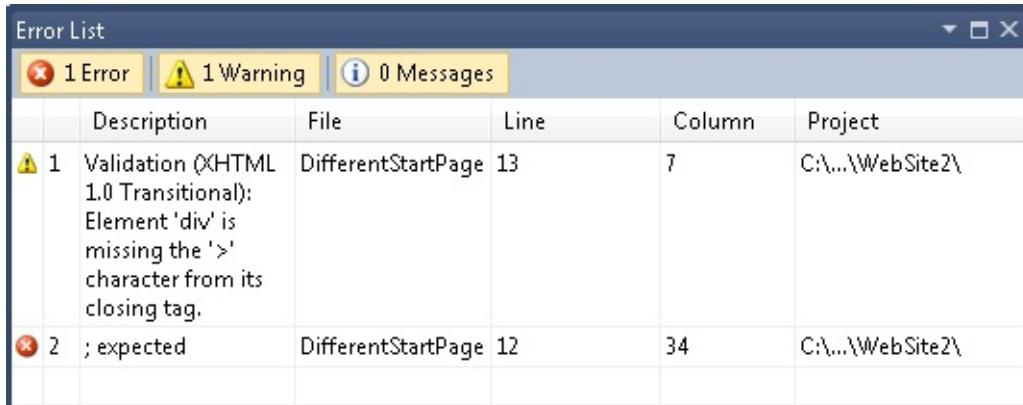
- You can avoid this dialog box by editing your Web.config file manually and setting **debug** to true.
- No dialog box comes up to enable you to turn this off again before you deploy, so be aware that you should ensure that it is turned off before you go to

production.

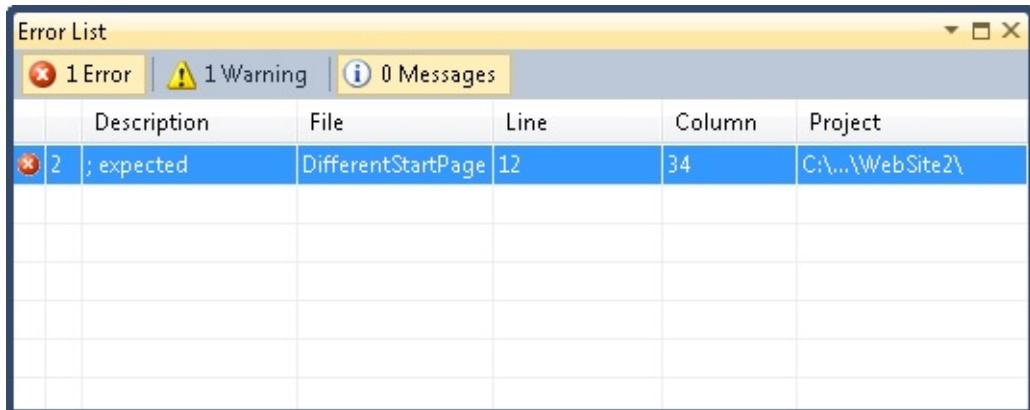
## AX.102 View the Error List Window

<b>DEFAULT</b>	Ctrl+\, E; Ctrl+\, Ctrl+E
<b>VISUAL BASIC 6</b>	Ctrl+\, E; Ctrl+\, Ctrl+E; Ctrl+W, Ctrl+E
<b>VISUAL C# 2005</b>	Ctrl+\, E; Ctrl+\, Ctrl+E; Ctrl+W, Ctrl+E; Ctrl+W, E
<b>VISUAL C++ 2</b>	Ctrl+\, E; Ctrl+\, Ctrl+E
<b>VISUAL C++ 6</b>	Ctrl+\, E; Ctrl+\, Ctrl+E
<b>VISUAL STUDIO 6</b>	Ctrl+\, E; Ctrl+\, Ctrl+E
<b>WINDOWS</b>	Alt,V, I
<b>MENU</b>	View   Error List
<b>COMMAND</b>	View.ErrorList
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0018

You can view the Error List window by pressing Ctrl+\, E inside Visual Studio:



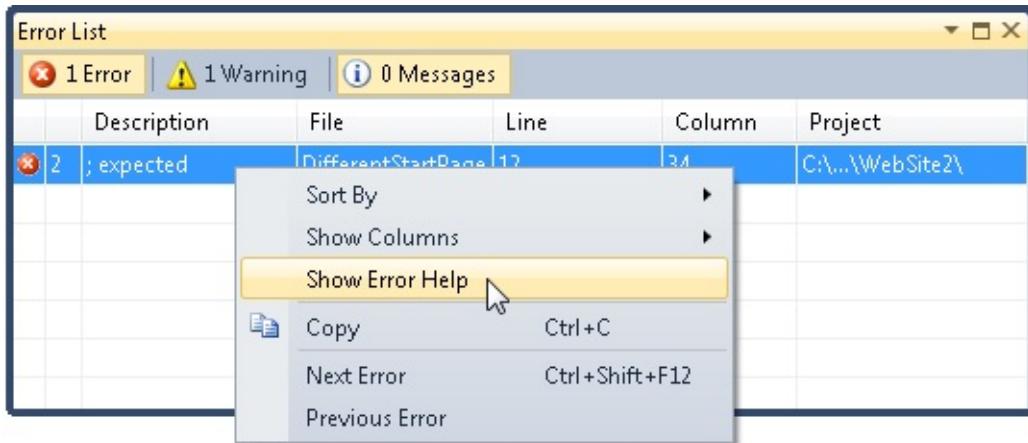
Also, you can filter by type, using the toggle buttons at the top (Errors, Warnings, Messages) to show or hide particular messages:



## AX.103 Show Error Help from Errors List Window

<b>DEFAULT</b>	F1
<b>VISUAL BASIC 6</b>	F1
<b>VISUAL C# 2005</b>	F1
<b>VISUAL C++ 2</b>	F1
<b>VISUAL C++ 6</b>	F1
<b>VISUAL STUDIO 6</b>	F1
<b>WINDOWS</b>	F1
<b>MENU</b>	[Context Menu]   Show Error Help
<b>COMMAND</b>	Help.F1Help
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0020

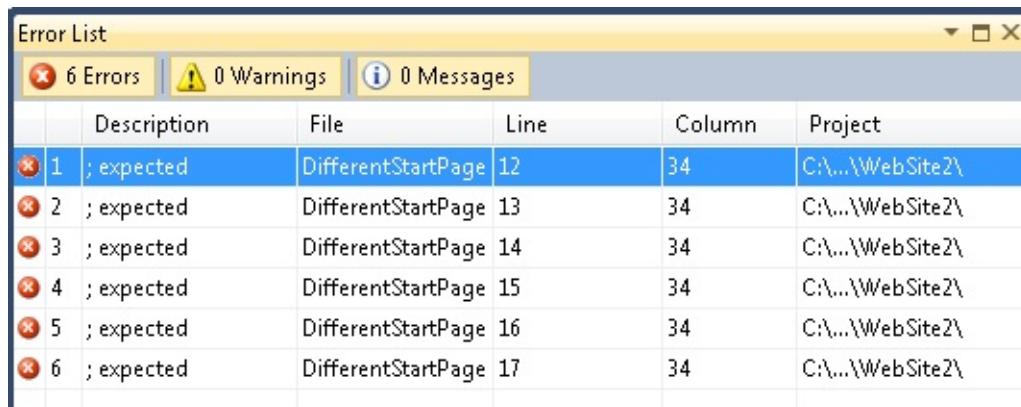
You can get help on any error in the Errors List window by right-clicking any error and choosing Show Error Help. This opens the documentation, if available, with details about the specific error:



## AX.104 Hide or Show Error List When the Build Finishes with Errors

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Projects and Solutions   Default
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0022

By default, the Error List window appears when build errors have occurred:

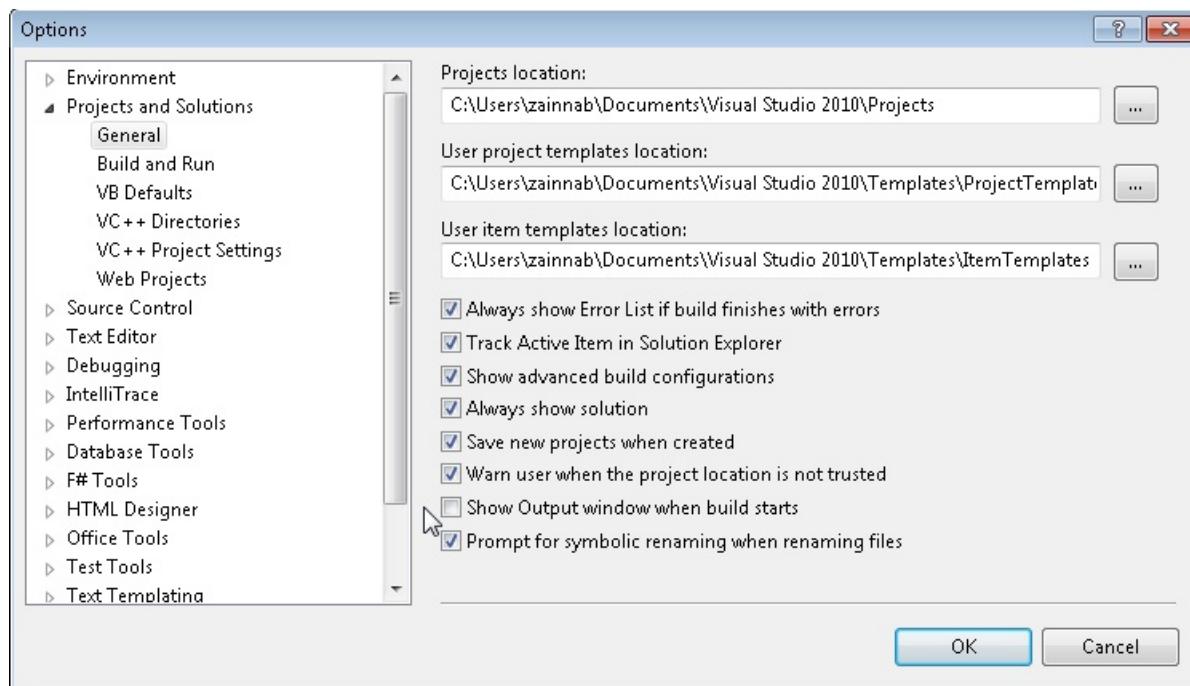


You can change this behavior by going to Tools | Options | Projects And Solutions | General and clearing the Always Show Error List If Build Finishes With Errors check box.

## AX.105 Show the Output Window During Build

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Projects and Solutions   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0045

When you do a build, by default, it always shows the Output window. If you don't want the Output window to show up every time you do a build, you can go to Tools | Options | Projects And Solutions | General and clear the Show Output Window When Build Starts check box:



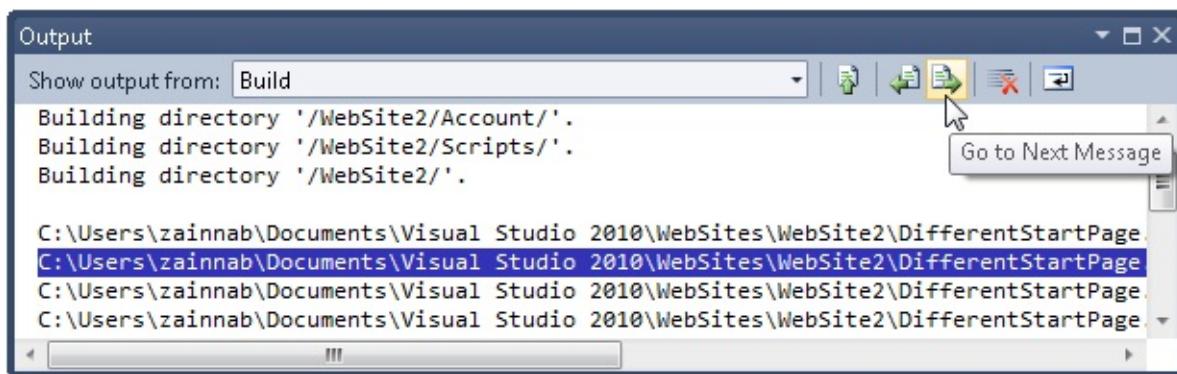
Of course, you can bring up the Output window anytime by pressing Ctrl+Alt+O.

## AX.106 Navigate Among Errors in the Output Window

<b>DEFAULT</b>	F8 (next); Shift+F8 (previous)
<b>VISUAL BASIC 6</b>	[no shortcut]
<b>VISUAL C# 2005</b>	F8 (next); Shift+F8 (previous)
<b>VISUAL C++ 2</b>	F4 (next); Shift+F4 (previous)

<b>VISUAL C++ 6</b>	F8 (next); F4 (next)
<b>VISUAL STUDIO 6</b>	F8 (next); F12 (next); Shift+F8 (previous); Shift+F12 (previous)
<b>COMMAND</b>	Edit.GoToNextLocation; Edit.GoToPrevLocation;
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0043

Did you know that you can use toolbar buttons on the Output window to navigate between errors?



Also, assuming the Output window is active, you can use F8 (next) and Shift+F8 (previous) to navigate among the errors as well.

The cursor in the editor automatically follows you as you go through the errors, places the cursor where you can make changes so that you just start typing, and has an indicator in the far left margin to show your current position:

```

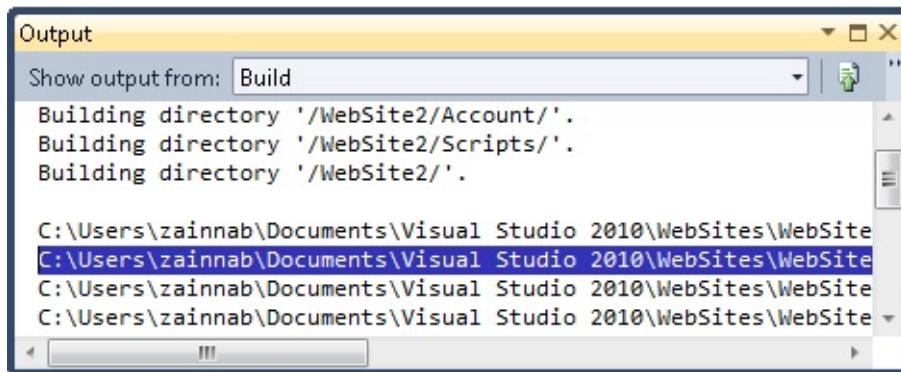
12 | Console.WriteLine("blah") ~
13 | Console.WriteLine("blah") ~
14 | Console.WriteLine("blah") ~
15 | Console.WriteLine("blah") ~
16 | Console.WriteLine("blah") ~
17 | Console.WriteLine("blah") ~

```

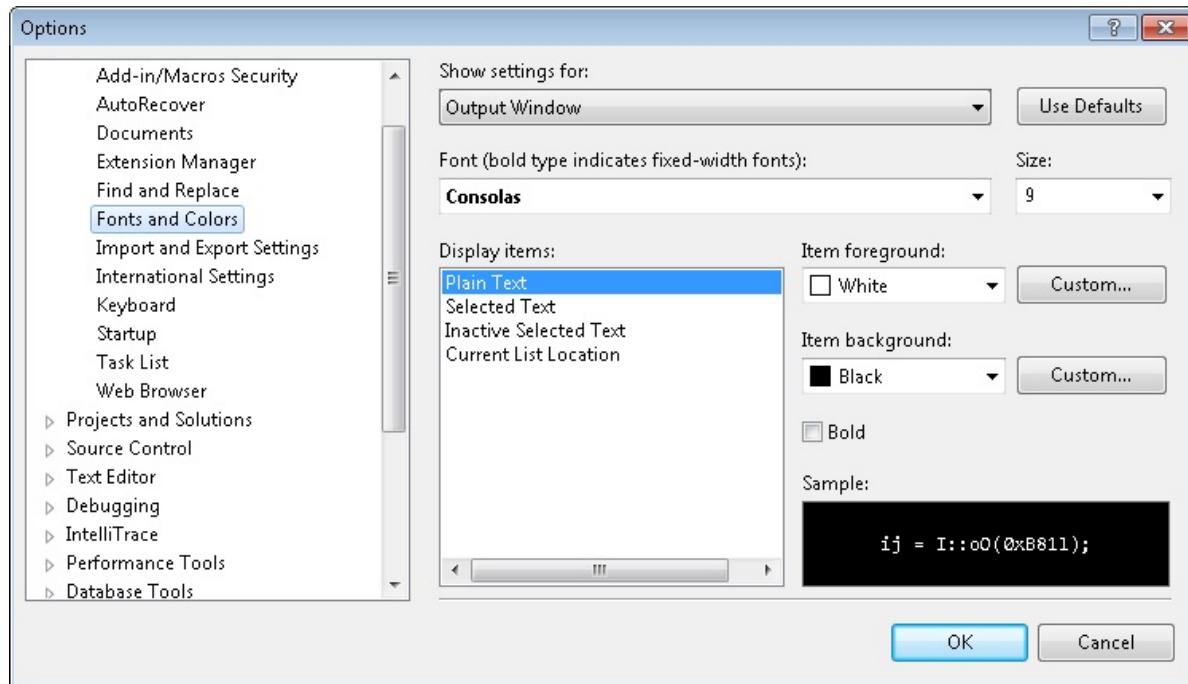
## AX.107 Customize the Output Window

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Environment   Fonts and Colors   Show settings for   Output Window
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010

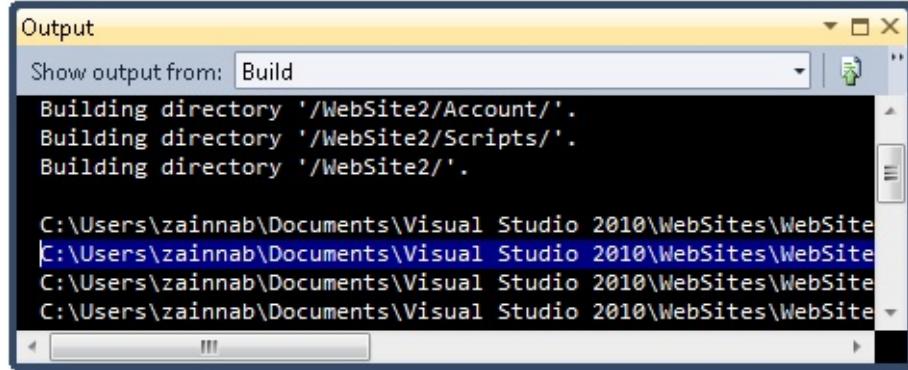
Working with the Output window is fairly common. Personalizing the window to your needs can definitely improve your work day. The Output window can be modified from its original configuration:



Just go to Tools | Options | Fonts And Colors | Show Settings For | Output Window. You can change the font type, color, and size for a variety of display items, as shown in the following illustration:



Here's the result in the Output window:



Obviously, you'll want to experiment with combinations that suit you.

## AX.108 Step Out of or Over a Method

<b>DEFAULT</b>	Shift+F11 (step out); F10 (step over)
<b>VISUAL BASIC 6</b>	Shift+F11 (step out); Ctrl+Shift+F8 (step out); F10 (step over); Shift+F8 (step over)
<b>VISUAL C# 2005</b>	Shift+F11 (step out); F10 (step over)
<b>VISUAL C++ 2</b>	Shift+F7 (step out); F10 (step over)
<b>VISUAL C++ 6</b>	Shift+F11 (step out); F10 (step over)
<b>VISUAL STUDIO 6</b>	Shift+F11 (step out); F10 (step over)
<b>WINDOWS</b>	Alt,D, T (step out); Alt, D, O (step over)
<b>MENU</b>	Debug   Step Out; Debug   Step Over
<b>COMMAND</b>	Debug.StepOut; Debug.StepOver
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0035

When debugging your code, you often come across a call to another method:

```
Sub Main()  
  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
  
    SomeMethodHere()  
  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
End Sub
```

At this point, if you want to see what is in the method, you can step into it by pressing F11:

```
Sub SomeMethodHere()  
    Console.WriteLine("awesome code here")  
    Console.WriteLine("awesome code here")  
End Sub
```

## Step Out

Once inside, you might decide you don't really need to go through all the code. You can, at any time, step out by pressing Shift+F11, which finishes execution of the current method and returns you to the original call:

```
Sub Main()  
  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
  
    SomeMethodHere()  
  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
End Sub
```

From here, you can continue on as you normally would with your debugging.

## Step Over

You have another option when you get to a method call:

You can decide that the method should just run without having to look at what is inside it. If you want to run the method and move to the next line of code, just press F10 to step over that method:

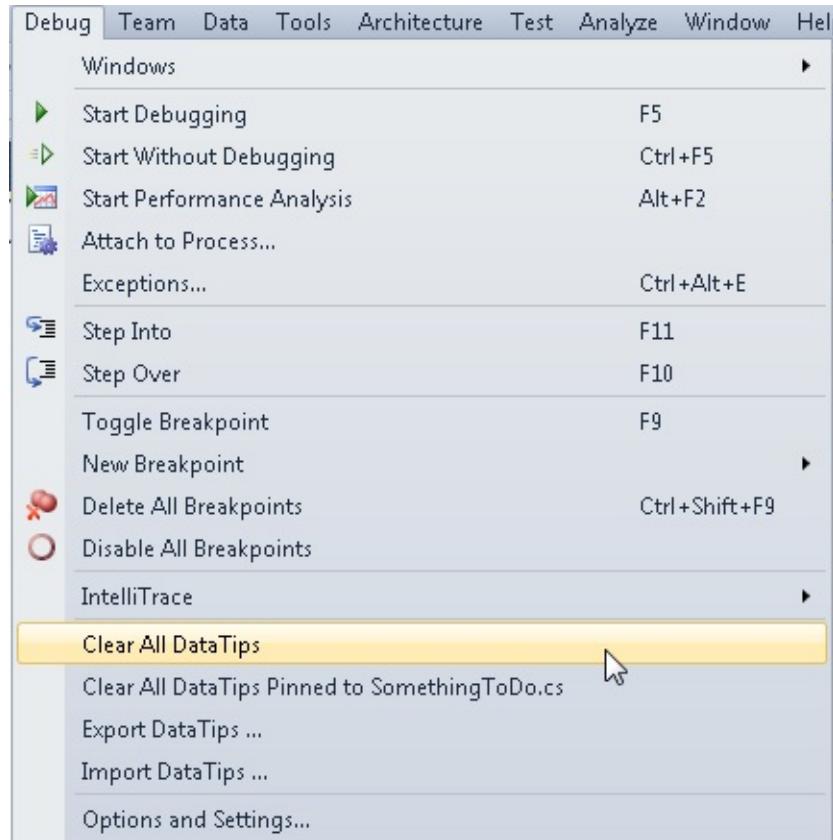
```
Sub Main()  
  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
  
    SomeMethodHere()  
  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
    Console.WriteLine("blah")  
End Sub
```

You can now navigate more effectively by using these techniques to determine how deep you want to get into method calls.

## AX.109 Clearing Your DataTips

<b>WINDOWS</b>	Alt,D, A, A, ENTER (clear all)
<b>MENU</b>	Debug   Clear All DataTips; Debug   Clear All DataTips Pinned to [file]
<b>COMMAND</b>	Debug.ClearAllDataTips
<b>VERSIONS</b>	2010
<b>CODE</b>	vstipDebug0014

As you create your DataTips, you might find that it is necessary to clear them out when you are done. You can clear your DataTips in a couple of ways from the Debug menu:



## Clear All DataTips

This command does what it says and clears *all* (pinned and floating) DataTips in the solution.

## Clear All DataTips Pinned to [File Name]

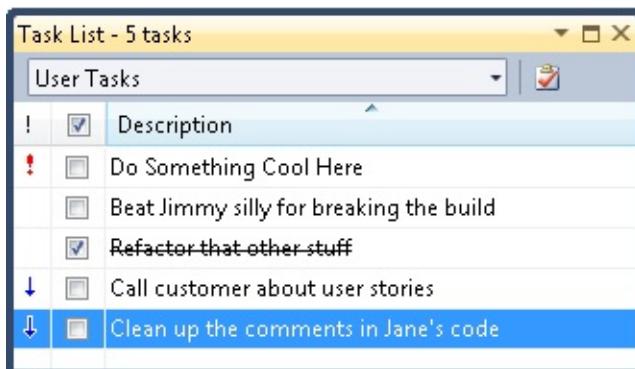
Clear only the pinned DataTips in [file name]. This option appears only if you have at least one pinned DataTip in a file. The referenced file is the one you have currently open in the editor.

## AX.110 Create User Tasks in the Task List

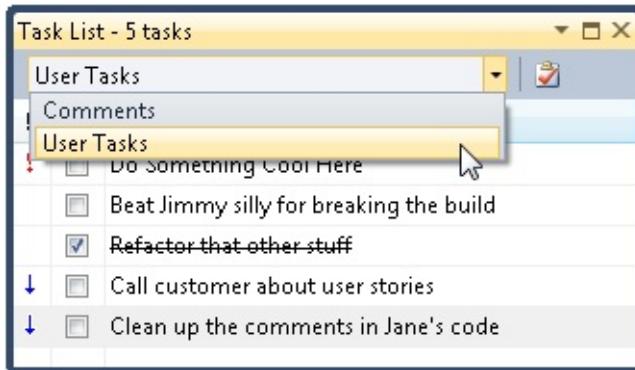
<b>DEFAULT</b>	Ctrl+\, Ctrl+T; Ctrl+\, T
<b>VISUAL BASIC 6</b>	Ctrl+\, Ctrl+T; Ctrl+\, T; Ctrl+ALT+K
<b>VISUAL C# 2005</b>	Ctrl+\, Ctrl+T; Ctrl+\, T; Ctrl+W,Ctrl+T; Ctrl+W, T
<b>VISUAL C++ 2</b>	Ctrl+\, Ctrl+T; Ctrl+\, T
<b>VISUAL C++ 6</b>	Ctrl+\, Ctrl+T; Ctrl+\, T

<b>VISUAL STUDIO 6</b>	Ctrl+\, Ctrl+T; Ctrl+\, T; Ctrl+Alt+K
<b>WINDOWS</b>	Alt, V, K
<b>MENU</b>	View   Task List
<b>COMMAND</b>	View.TaskList
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0027

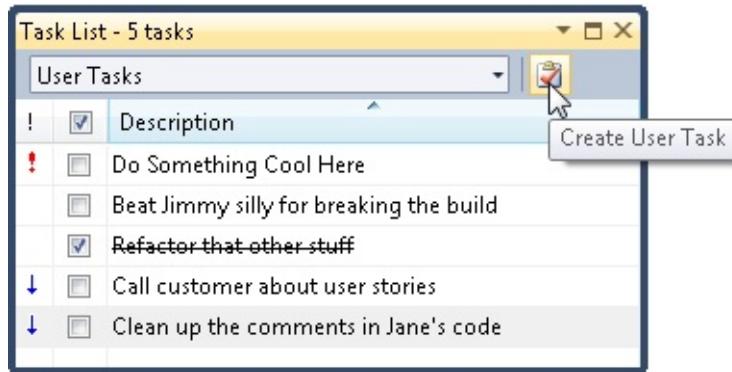
Creating tasks is useful when you need reminders for getting work done. User Tasks is a checklist of “to do” items that are stored in the Solution User Options (.suo) file. They are per-user settings and are not typically checked into source control:



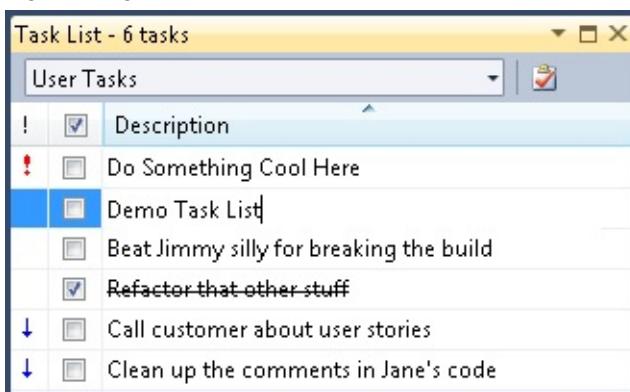
To begin, make sure that User Tasks is selected from the drop-down list in the Task List (Ctrl+\, T):



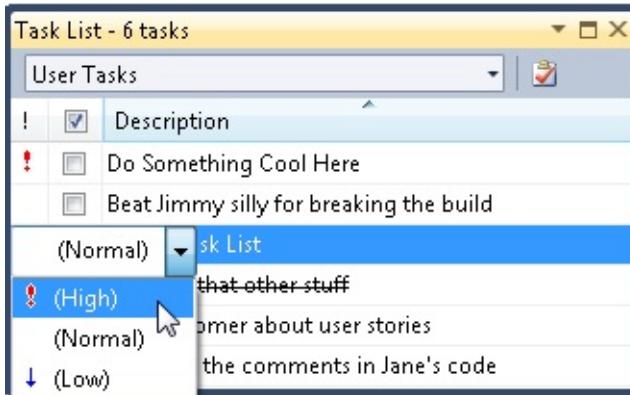
Next, click the Create User Task button:



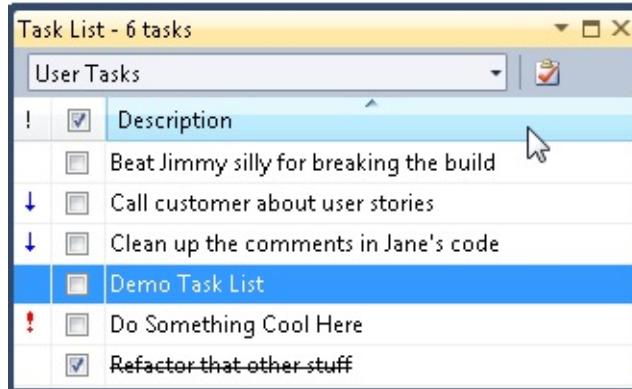
Now you can enter any task you want:



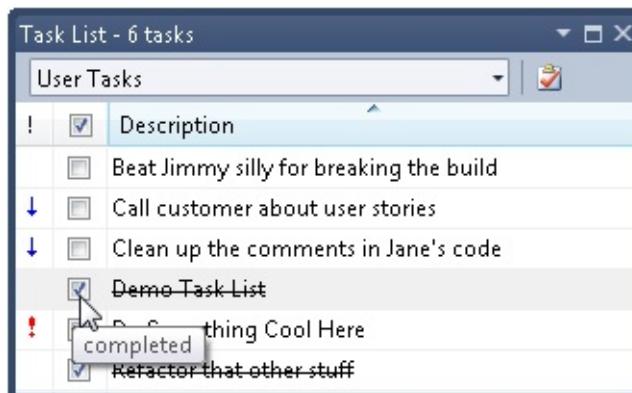
You can easily edit any task by double-clicking or pressing Enter while in it. Also, you can set priority levels on them by clicking in the box to the far left of any task:



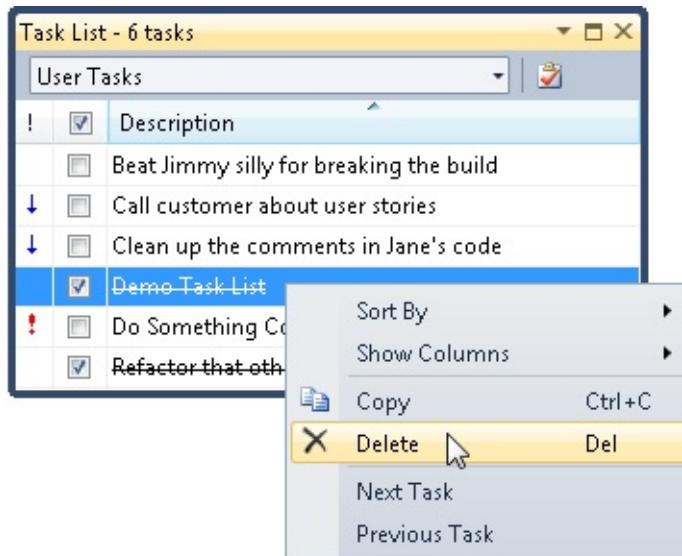
Naturally, you can sort them by any column, such as Description or Priority:



When you are done with a task, you can just click the check box to mark it complete:



Or you can right-click the task and delete it:



#### NOTE

A command called `CreateUserTask` is available, but it doesn't accept any arguments and creates only a

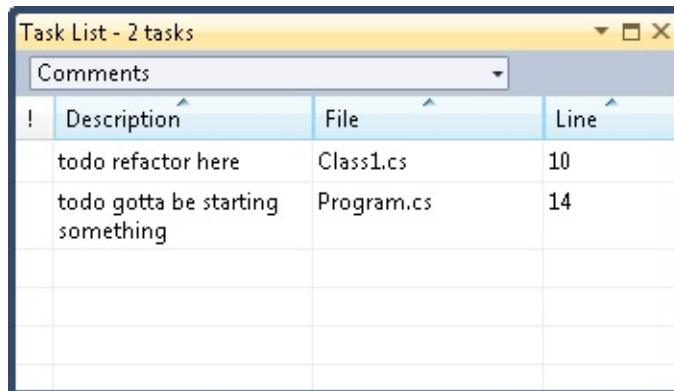
blank task. For this reason, I didn't list it in the summary information.

## AX.111 Show the Full File Path in the Task List

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0033

Sometimes you have comments and/or shortcuts with descriptions and/or file names that are the same but in different solutions. Knowing the full path to the file can help you determine which item you are looking at in the Task List. This tip can help you with the items that are part of the Comments and Shortcuts areas.

Normally, an item in the Task List doesn't have the full path:



You can enable the full path by going to Tools | Options | Environment | Task List | Task List Options and clearing the Hide Full File Paths check box:

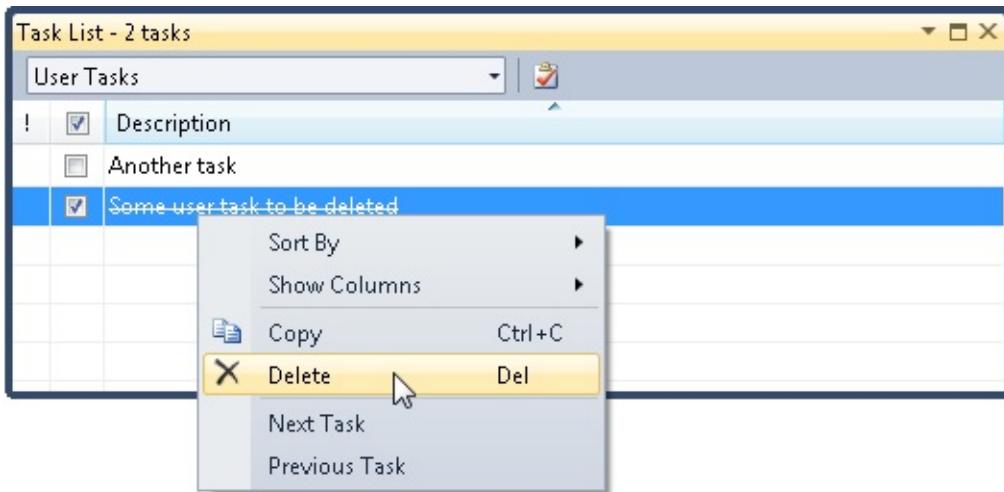
Now you can see the full file path in the Task List and clearly see where a comment or shortcut resides:

Task List - 2 tasks						
Comments						
	Description	File				
!	Description	File				
!	todo gotta be starting something	C:\Users\zainnab\... Console\Program.cs	14			
!	todo refactor here	C:\Users\zainnab\...Clean Console\Class1.cs	10			

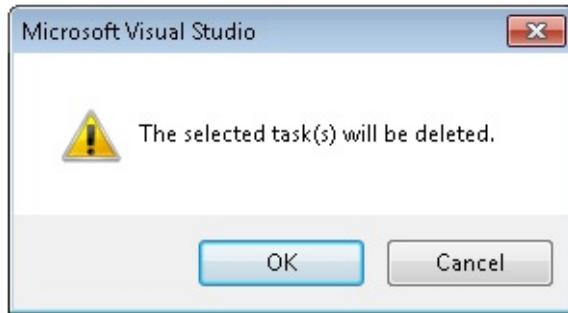
## AX.112 Disable the Prompt for Deleting Items from the Task List

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options   Environment   Task List   Task List
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0031

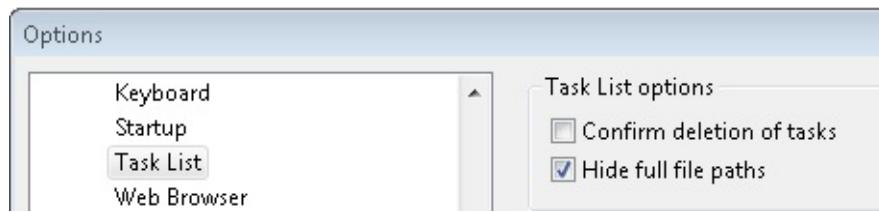
When dealing with the Task List, removing items is quite common. To delete an item from the Task List, simply right-click any item and choose Delete:



The following prompt appears:



You can turn the prompt dialog box off by going to Tools | Options | Environment | Task List | Task List Options and clearing the Confirm Deletion Of Tasks check box:



Now when you delete tasks, you have no dialog box to contend with. Of course, this means that you can also now accidentally delete a task without anything to stop you, so use this option at your own risk.

## AX.113 Navigate Task List Entries with the Keyboard

<b>DEFAULT</b>	F8 (next location); Shift+F8 (prev location); [no shortcut] (next task); [no shortcut] (prev task)
<b>VISUAL BASIC 6</b>	[no shortcut] (next location); [no shortcut] (prev location); Ctrl+Shift+F12 (next task); [no shortcut] (prev task)
<b>VISUAL C# 2005</b>	F8 (next location); Shift+F8 (prev location); [no shortcut] (next task); [no shortcut] (prev task)
<b>VISUAL C++ 2</b>	F4 (next location); Shift+F4 (prev location); [no shortcut] (next task); [no shortcut] (prev task)
<b>VISUAL C++ 6</b>	F8 (next location); F4 (next location); [no shortcut] (next task); [no shortcut] (prev task)
<b>VISUAL STUDIO 6</b>	F8 (next location); F12 (next location); Shift+F8 (prev location); Shift+F12 (prev location); [no shortcut] (next task); [no shortcut] (prev task)
<b>COMMAND</b>	Edit.GoToNextLocation; Edit.GoToPrevLocation; View.NextTask; View.PreviousTask

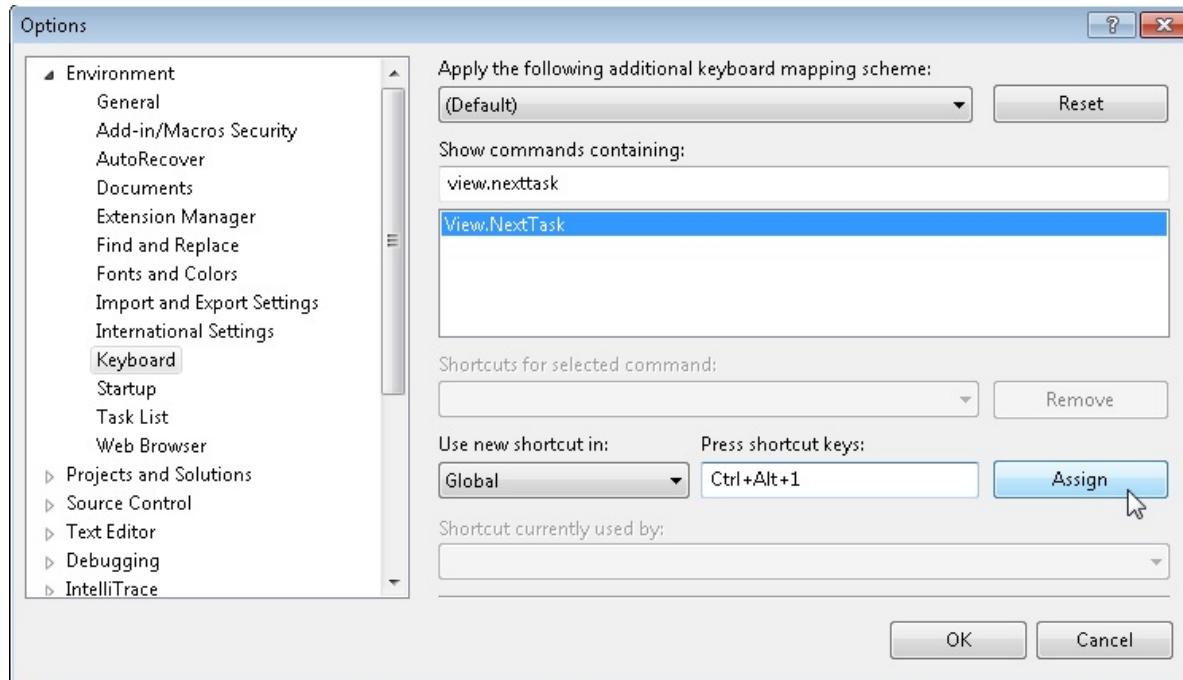
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0092

F8 and Shift+F8 are the universal “next” and “previous” keyboard shortcuts for items in tool window lists. For example, when the Task List is up, these keyboard commands move the focus between task items. If the Errors window is up, the commands move the focus between errors. However, you cannot use these commands to switch to the Task List from the Errors window.

What if you wanted a set of keyboard shortcuts that switched to the Task List and then navigated between them? Just go to Tools | Options | Environment | Keyboard, and assign shortcut keys to the View.NextTask and View.PreviousTask commands, as shown in the following illustration.

#### NOTE

For more information about assigning shortcut keys, see vstipTool0063 (03.26 Keyboard Shortcuts: Creating New Shortcuts, page 127).

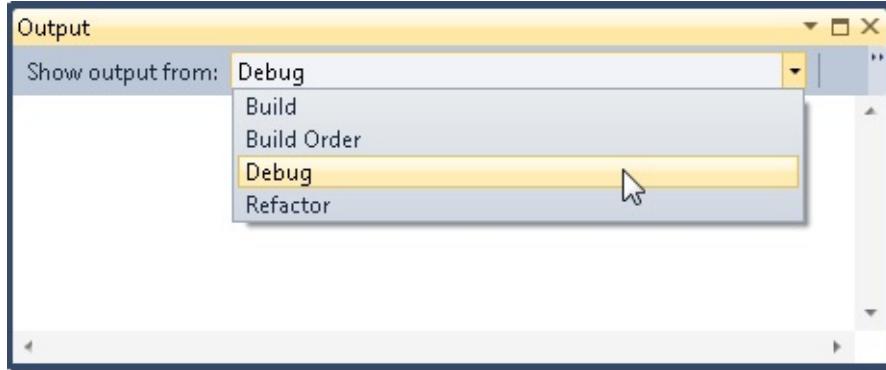


## AX.114 Navigating Between Output Window Panes with the Keyboard

<b>COMMAND</b>	Window.NextSubPane; Window.PreviousSubPane
----------------	--

<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0091

The Output window can have several panes. How many depends on your context:



You can use Window.[Next / Previous]Subpane to move between these panes. However, the commands are not bound, by default, to any keyboard mappings. That is easily solved by going to Tools | Options | Environment | Keyboard and assigning shortcut keys to the commands.

#### NOTE

For more information about assigning shortcut keys, see vstipTool0063 (03.26 Keyboard Shortcuts: Creating New Shortcuts, page 127).

## AX.115 The Watch Window: Moving Values Between Watch Windows

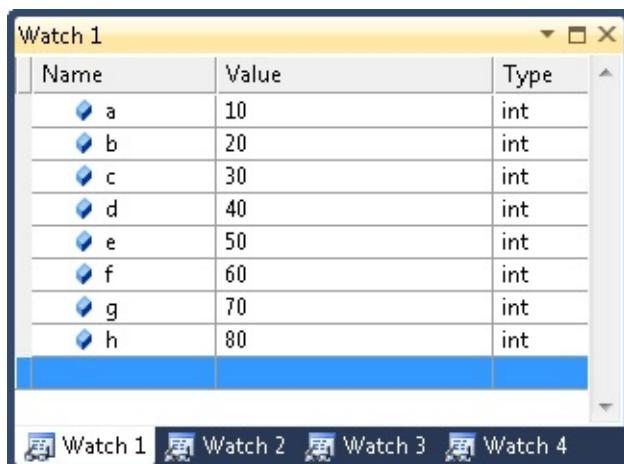
<b>DEFAULT</b>	Ctrl+Alt+W,1; Ctrl+Alt+W,[2-4]
<b>VISUAL BASIC 6</b>	Ctrl+Alt+W,1; Ctrl+Alt+W,[2-4]
<b>VISUAL C# 2005</b>	Ctrl+Alt+W,1; Ctrl+D, Ctrl+W; Ctrl+D, W; Ctrl+Alt+W,[2-4]
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+Alt+W,1; Ctrl+Alt+W,[2-4]
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+W,1; Ctrl+Alt+W,[2-4]
<b>WINDOWS</b>	Alt,D, W, W, [1-4]
<b>MENU</b>	Debug   Windows   Watch   Watch [1,2,3,4]
<b>COMMAND</b>	Debug.Watch[1,2,3,4]

<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0105

You can have up to four Watch windows in Visual Studio. The reason you get all these windows is so that you can easily organize your watches into groups. Opening a particular window is easy enough: Just press Ctrl+Alt+W and then select the number of the window you want (1, 2, 3, or 4):



The only problem is that anytime you use Add Watch or QuickWatch, the watch expression always gets added to Watch 1:

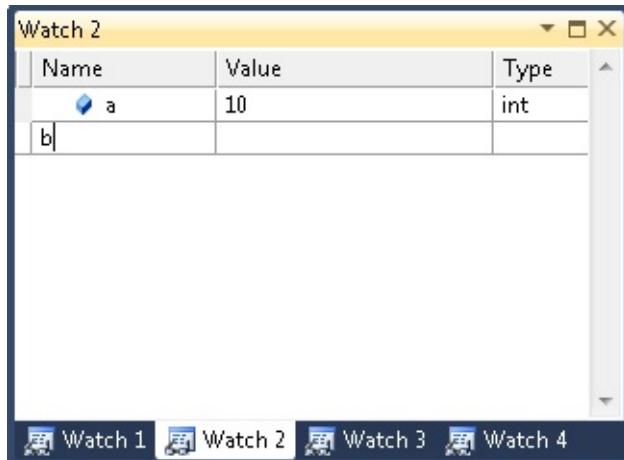


## Moving Between Windows

What if you want to move to another window? Let's look at some of your options.

### Type it in

You can just go to the window you want and type in the value you are looking for:



## Copy and paste

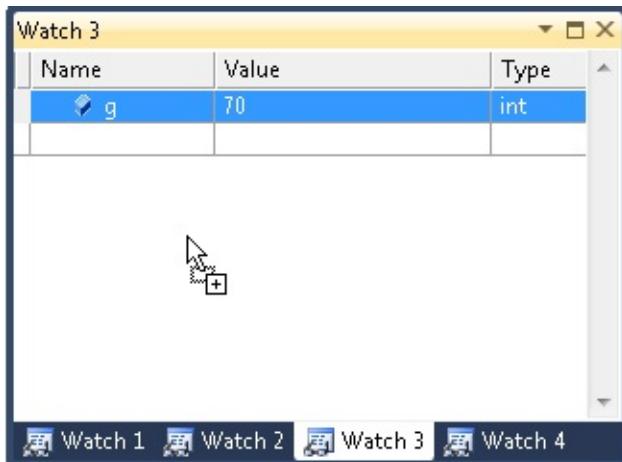
You can copy and paste the value from one window to another:

### NOTE

You can copy but not cut in the Watch window. You have to delete the original value from Watch 1 if you want to actually “move” it.

## Click and drag

You can click and drag to copy a value from one window to another:

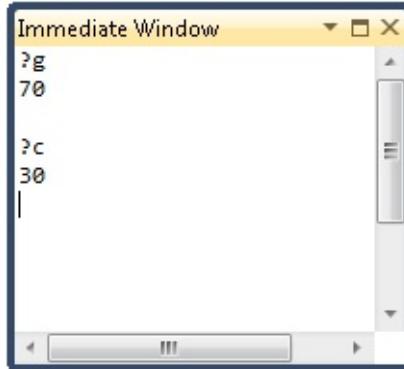


## AX.116 The Immediate Window: Simple Printing and Changing Values

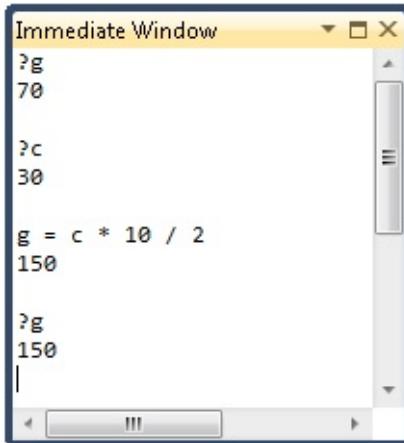
<b>DEFAULT</b>	Ctrl+Alt+I
<b>VISUAL BASIC 6</b>	Ctrl+Alt+I; Ctrl+G
<b>VISUAL C# 2005</b>	Ctrl+Alt+I; Ctrl+D, Ctrl+I; Ctrl+D, I
<b>WINDOWS</b>	Alt,D, W, I
<b>MENU</b>	Debug   Windows   Immediate
<b>COMMAND</b>	Debug.Immediate
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0094

The Immediate Window’s versatility is great, and it can give instant feedback with information you might need to help with your application. The first thing people

usually learn when using it is how to print values. Just go into debug mode, and use either **Debug.Print(variable)** or (more commonly) just **?variable**:



Additionally, you can modify values:



This is pretty basic assignment in these examples, but you can run pretty much any valid code to change values.

## AX.117 The Immediate Window: Working with Members

<b>DEFAULT</b>	Ctrl+Alt+I
<b>VISUAL BASIC 6</b>	Ctrl+Alt+I; Ctrl+G
<b>VISUAL C# 2005</b>	Ctrl+Alt+I; Ctrl+D, Ctrl+I; Ctrl+D, I
<b>VISUAL C++ 2</b>	Ctrl+Alt+I
<b>VISUAL C++ 6</b>	Ctrl+Alt+I
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+I

<b>WINDOWS</b>	Alt,D, W, I
<b>MENU</b>	Debug   Windows   Immediate
<b>COMMAND</b>	Debug.Immediate
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipTool0095

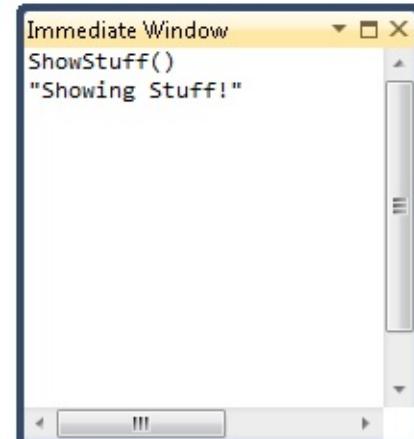
When using the Immediate Window, you work with class and object members directly. Both the traditional usage in Debug mode and the lesser-known use in Design mode is available.

## Debug

You can use any method or property as long as it is in context. So, for example, when you are in debug mode, you can call any method that is in scope:

```
class Person
{
    public string ScopeFun()
    {
        return "blah";
    }

    private string ShowStuff()
    {
        return "Showing Stuff!";
    }
}
```



## Design

### WARNING

When working with members at design time, a build will occur. This might have unintended consequences, so make sure you have experimented with this feature a bit before you use it on production code.

### NOTE

You cannot use design time expression evaluation in project types that require starting up an execution environment, including Visual Studio Tools for Office projects, web projects, Smart Device projects, and SQL projects.

A lesser-known feature is that you can work with properties and methods while in design mode. If you have static (“Shared” in VB) methods on a class, for example, you can just execute them without going into debug mode:

```
class Person
{
    public static string SomethingCool()
    {
        return "Good times!";
    }

    public string ScopeFun()
    {
        return "blah";
    }

    private string ShowStuff()
    {
        return "Showing Stuff!";
    }
}
```



For object members, you need to create an instance of the object before working with the members:

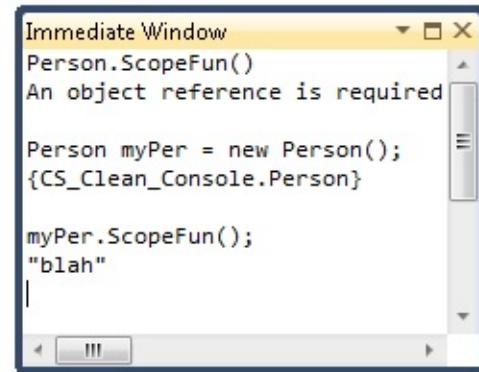
```

class Person
{
    public static string SomethingCool()
    {
        return "Good times!";
    }

    public string ScopeFun()
    {
        return "blah";
    }

    private string ShowStuff()
    {
        return "Showing Stuff!";
    }
}

```



## AX.118 The Immediate Window: Design-Time Breakpoints

<b>DEFAULT</b>	Ctrl+Alt+I
<b>VISUAL BASIC 6</b>	Ctrl+Alt+I; Ctrl+G
<b>VISUAL C# 2005</b>	Ctrl+Alt+I; Ctrl+D, Ctrl+I; Ctrl+D, I
<b>VISUAL C++ 2</b>	Ctrl+Alt+I
<b>VISUAL C++ 6</b>	Ctrl+Alt+I
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+I
<b>WINDOWS</b>	Alt,D, W, I
<b>MENU</b>	Debug   Windows   Immediate
<b>COMMAND</b>	Debug.Immediate
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipTool0096

In vstipTool0095 ([AX.117 The Immediate Window: Working with Members](#), page A179), I demonstrated that you could do design-time execution of members. I thought it would be instructive to mention that you can also use this technique to hit breakpoints in your code.

For example, assume you have an application that has a class with a static method. Now set a breakpoint on a line of code:

```
class Person
{
    public static string SomethingCool()
    {
        return "Good times!";
    }

    public string ScopeFun()
    {
        return "blah";
    }

    private string ShowStuff()
    {
        return "Showing Stuff!";
    }
}
```



Then, in design mode, execute the method from the Immediate Window.

It executes the code and stops at the breakpoint, ready for you to continue debugging:

```

class Person
{
    public static string SomethingCool()
    {
        return "Good times!";
    }

    public string ScopeFun()
    {
        return "blah";
    }

    private string ShowStuff()
    {
        return "Showing Stuff!";
    }
}

```



This is an interesting feature of design-time execution, which you can use to quickly get to an area for debugging.

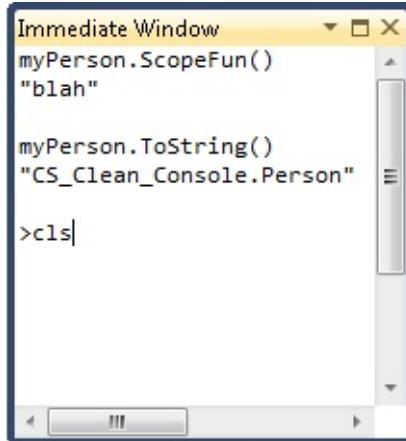
## AX.119 The Immediate Window: Running Commands

<b>DEFAULT</b>	Ctrl+Alt+I
<b>VISUAL BASIC 6</b>	Ctrl+Alt+I; Ctrl+G
<b>VISUAL C# 2005</b>	Ctrl+Alt+I; Ctrl+D, Ctrl+I; Ctrl+D, I
<b>VISUAL C++ 2</b>	Ctrl+Alt+I
<b>VISUAL C++ 6</b>	Ctrl+Alt+I
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+I
<b>WINDOWS</b>	Alt,D, W, I
<b>MENU</b>	Debug   Windows   Immediate
<b>COMMAND</b>	Debug.Immediate
<b>VERSIONS</b>	2005, 2008, 2010

CODE

vstipTool0098

You can run commands when you are in the Immediate Window. Just type > [command] to run any command. When I want to quickly clear out the Immediate Window, one of my favorite commands to run is >`cls`:

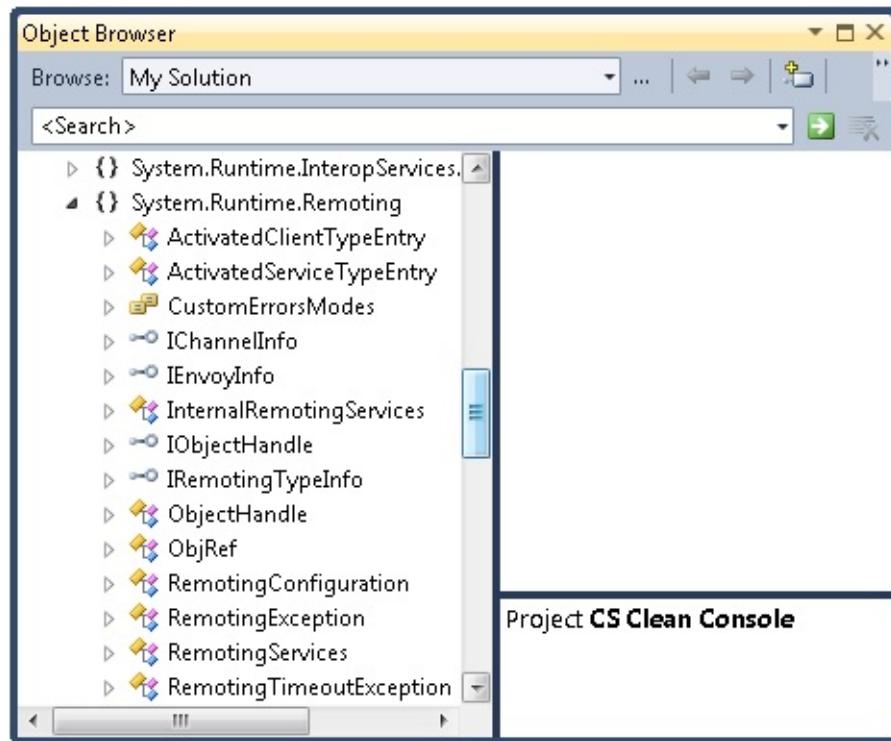


Any valid command can be run in this way, so you can run any command you want from the Immediate Window.

## AX.120 Class View and Object Browser Icons

VERSIONS	2005, 2008, 2010
CODE	vstipTool0076

You often encounter icons to represent symbols in a variety of places, such as when you use the Object Browser:



The help documentation lists out the icons for you and I have listed them in the following table for your reference. You can see them online at <http://msdn.microsoft.com/en-us/library/y47ychfe.aspx>.

#### Class View and Object Browser Icons:

Icon	Description	Icon	Description
{}	Namespace	=	Method or function
◆	Class	≡+	Operator
○—○	Interface	■	Property
◆◆	Structure	◆	Field or variable
◆◆◆	Union	⚡	Event
■■	Enum	■	Constant
□	TypeDef	■■	Enum item
●●	Module	■■■	Map item
■■■■	Intrinsic	=	External declaration
■■■■■	Delegate	■■■■■	Macro

	Exception		Template
	Map		Unknown or error
	Global		Type forwarding
	Extension method		

### Modifier Icons:

Icon	Description
<No Signal Icon>	Public. Accessible from anywhere in this component and from any component that references it.
	Protected. Accessible from the containing class or type, or those derived from the containing class or type.
	Private. Accessible only in the containing class or type.
	Internal. Accessible only from this component.
	Friend. Accessible only from the project.
	Shortcut. A shortcut to the object.

## AX.121 Output Window vs. Immediate Window

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Debugging   General
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0046

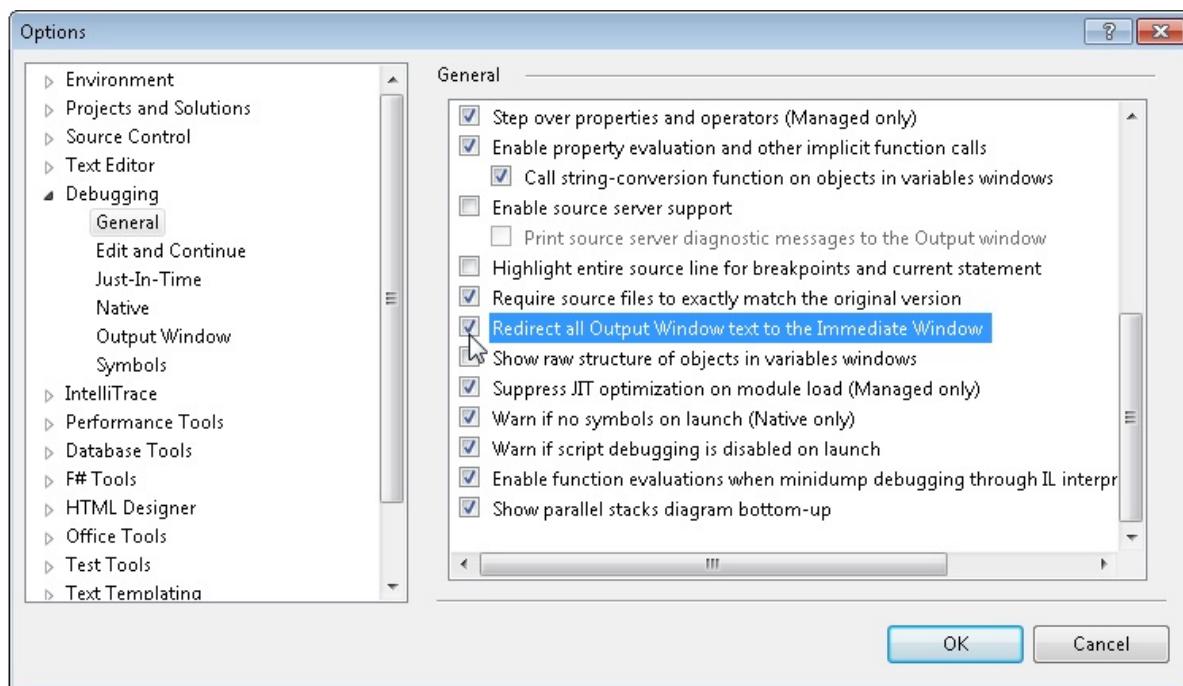
Depending on your settings, you might want to redirect Output window messages that you create to the Immediate window, or vice versa. So, consider your messages (Asserts, for example) that currently go to the Output window:

The screenshot shows the Windows Output window. The title bar says "Output". The content area displays a stack trace starting with "----- DEBUG ASSERTION FAILED -----". It includes several assert messages: "Assert Short Message", "Assert Message Two", "Assert Long Message", and "Assert Details". Below these, a detailed stack trace is shown, starting from "at Program.Main(String[] args) C:\Users\za..." and continuing through various runtime assembly and host process methods.

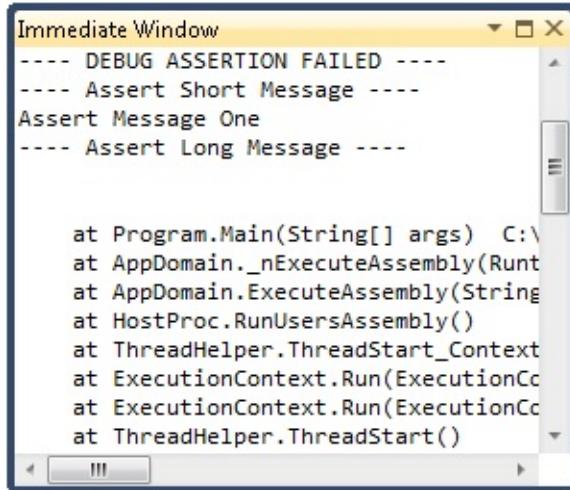
```
----- DEBUG ASSERTION FAILED -----
----- Assert Short Message -----
Assert Message Two
----- Assert Long Message -----
Assert Details

at Program.Main(String[] args) C:\Users\za...
at AppDomain._nExecuteAssembly(RuntimeAssembly assembly)
at AppDomain.ExecuteAssembly(String assemblyFile)
at HostProc.RunUsersAssembly()
at ThreadHelper.ThreadStart_Context(Object state)
at ExecutionContext.Run(ExecutionContext context)
at ExecutionContext.Run(ExecutionContext context, ContextCallback callback, Object state)
```

Go to Tools | Options | Debugging | General, and select the Redirect All Output Window Text To The Immediate Window check box:



Now your messages go to the Immediate window instead of the Output window:



#### NOTE

Not all information is redirected to the Immediate window. In this example, the results of your Assert are redirected, but some system information is always shown in the Output window.

## AX.122 The Object Browser: Settings

<b>DEFAULT</b>	Ctrl+Alt+J
<b>VISUAL BASIC 6</b>	Ctrl+Alt+J; F2
<b>VISUAL C# 2005</b>	Ctrl+Alt+J; Ctrl+W, Ctrl+J; Ctrl+W, J
<b>VISUAL C++ 2</b>	Ctrl+Alt+J; Shift+Alt+F1
<b>VISUAL C++ 6</b>	Ctrl+Alt+J
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+B; F2
<b>WINDOWS</b>	Alt,V, J
<b>MENU</b>	View   Object Browser
<b>COMMAND</b>	View.ObjectBrowser
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0080

Object Browser settings are a critical part of your browsing experience. They determine what you see and how much detail exists. This tip shows you how to use the settings to your advantage.

First, the settings button is located to the far right on the toolbar and looks like a sheet of paper with a check mark in it. There is quite a bit to look at, as you can see:



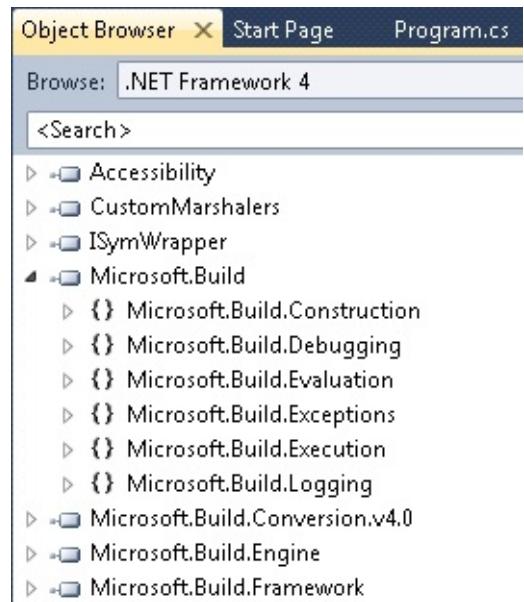
#### NOTE

The options you see are based on your context in the Object Browser, so not all options may be currently available.

## Containers vs. Namespaces

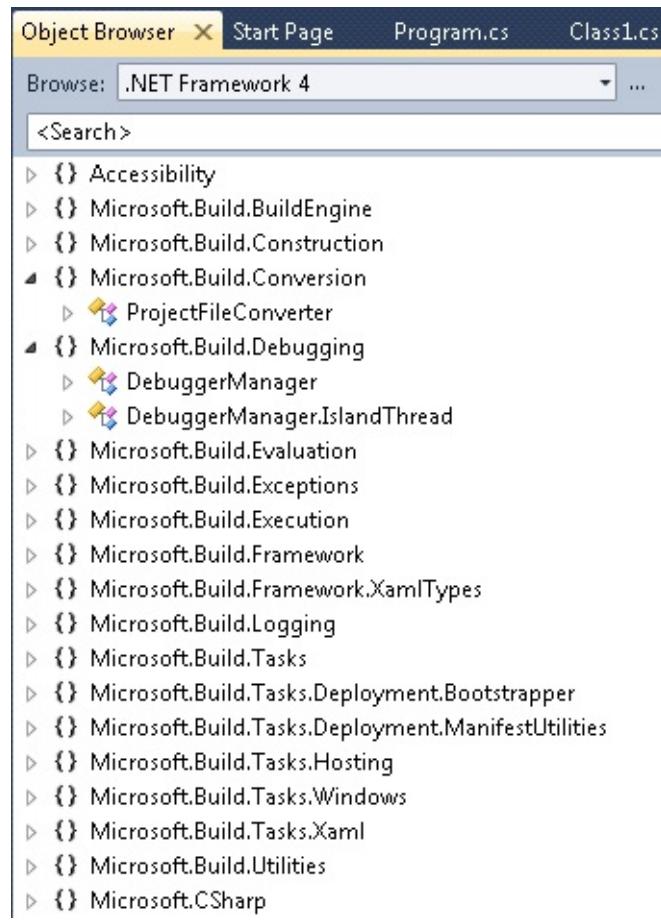
### View Containers

Sets the highest-level items in the Objects pane to physical containers, such as components, assemblies, source browser (.bsc) files, and output type libraries (.tlb). These expand to show the namespaces that are contained:

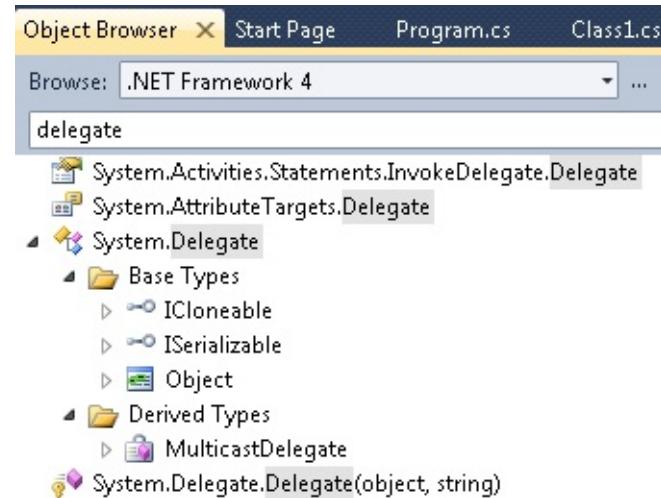


## View Namespaces

Sets the highest-level items in the Objects pane to namespaces. Namespaces stored in multiple physical containers are merged. These expand to show the items that are contained:



## Base and Derived Types



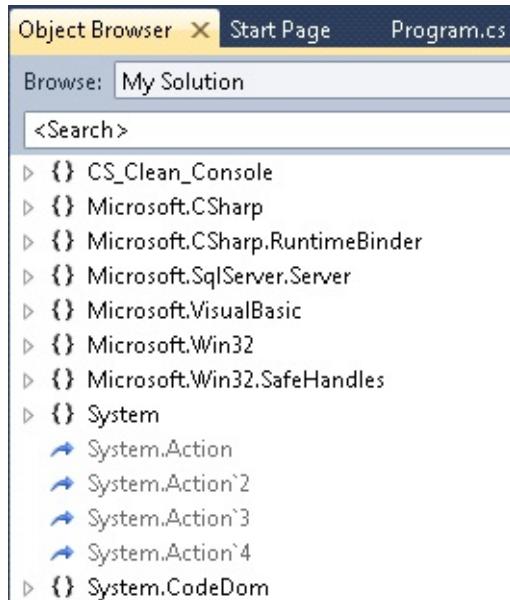
### Show Base Types

Toggles showing the Base Types folder and contents.

### Show Derived Types

Toggles showing the Derived Types folder and contents and is available only for Visual C++ projects and the .NET Framework.

## Hidden Types and Members



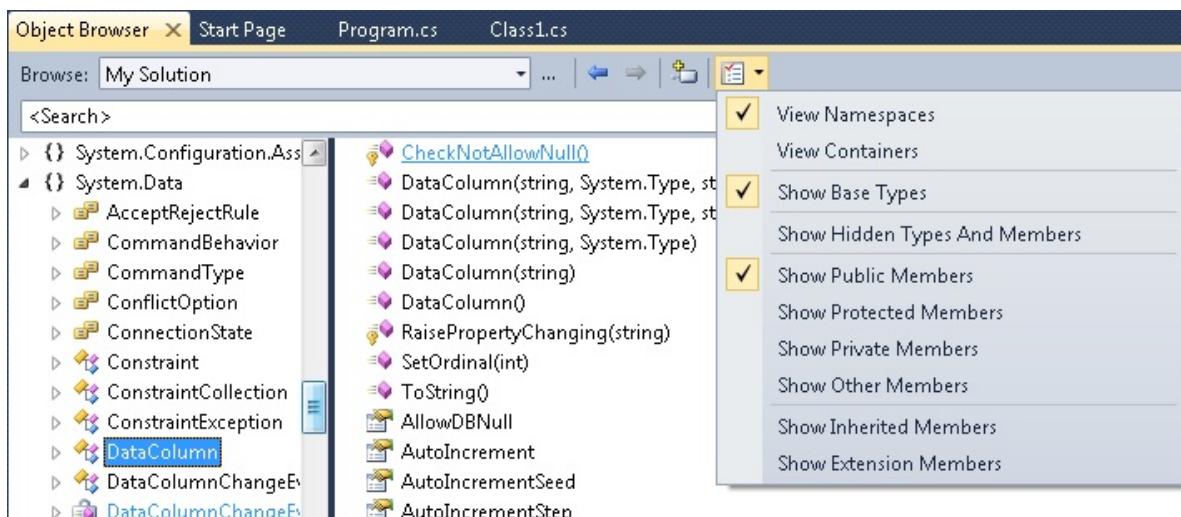
## Show Hidden Types And Members

Toggles display of hidden types in the Objects pane and hidden members in the Members pane. Hidden items show up as dimmed items in the lists.

## Public, Protected, Private, and Other Members

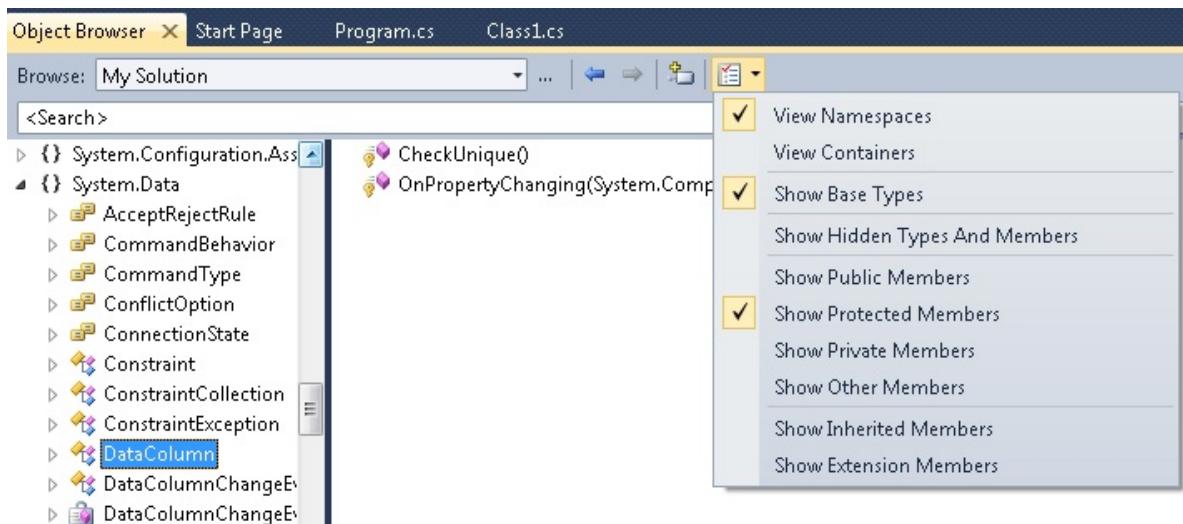
### Show Public Members

Displays members that are public or protected:



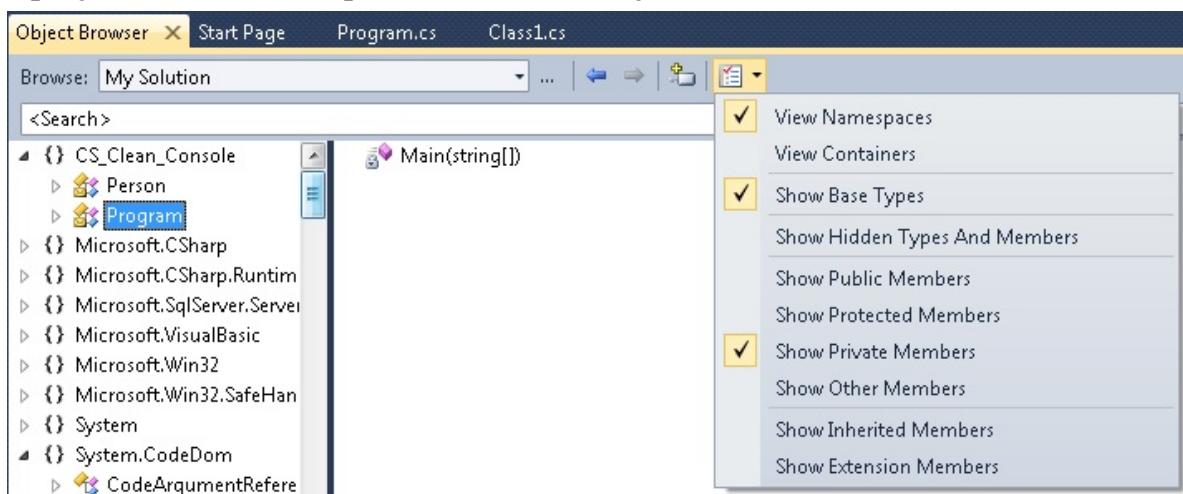
## Show Protected Members

Displays members that are protected:



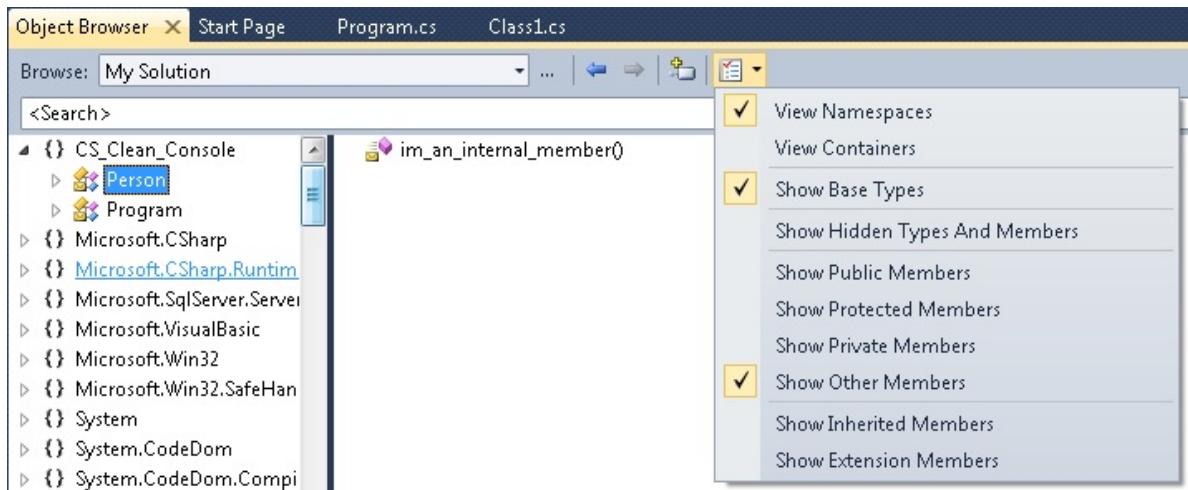
## Show Private Members

Displays members with private accessibility:



## Show Other Members

Displays members that do not fall into the category of public, protected, private, or inherited:

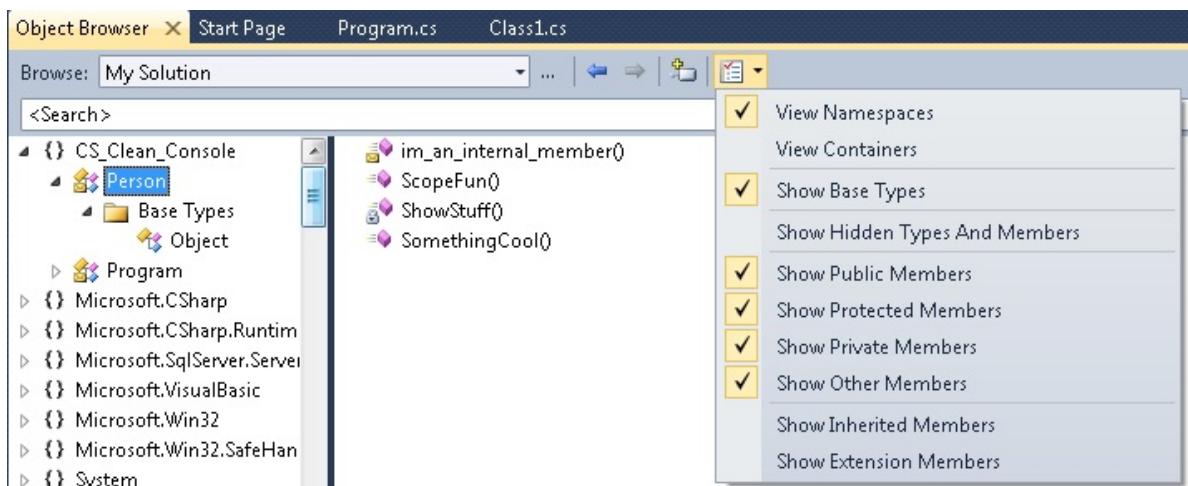


## Inherited Members and Extension Methods

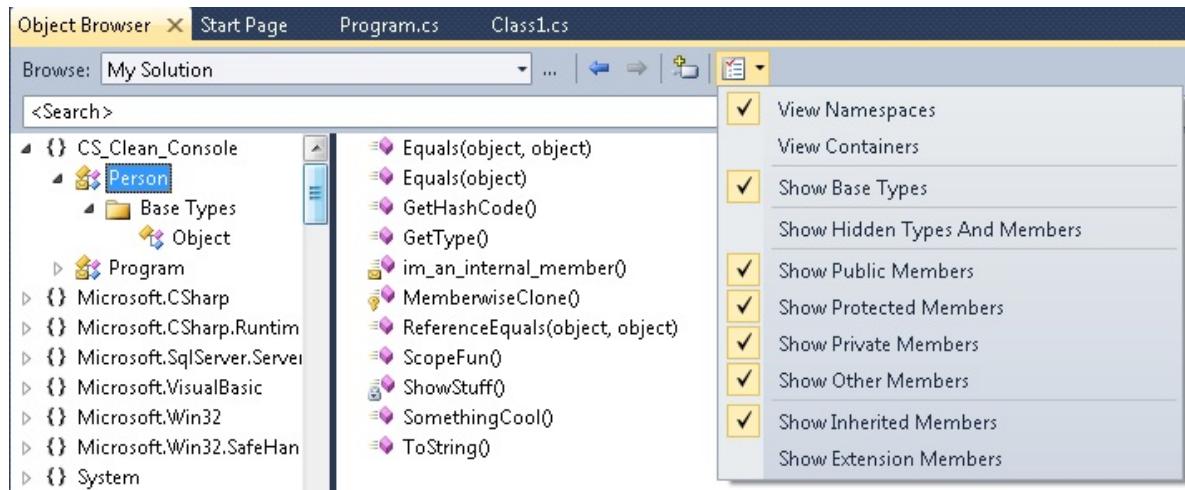
### Show Inherited Members

Toggles display of inherited members in the Members pane, as shown in the following illustrations:

Off:

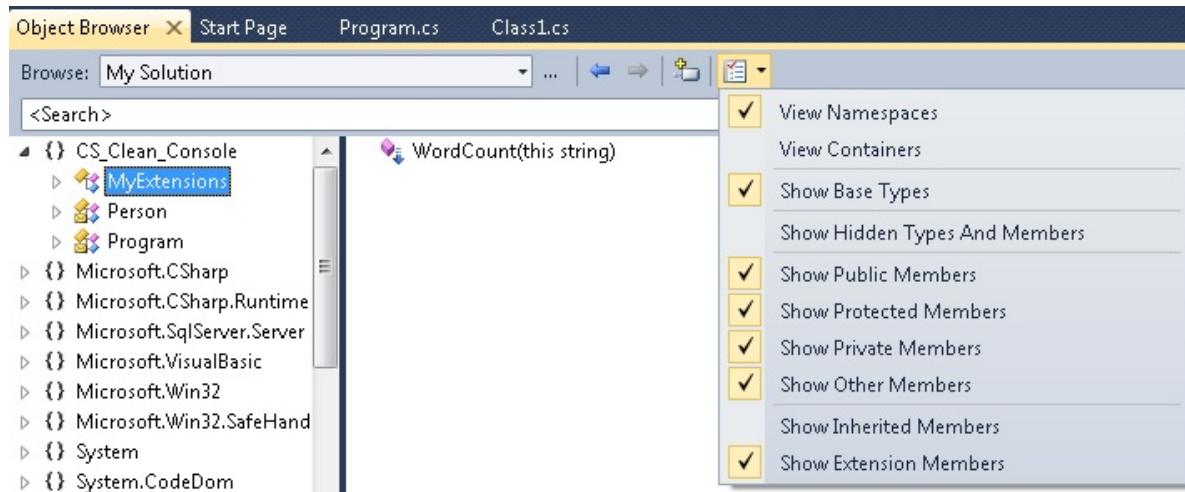


On:



## Show Extension Methods

Toggles the display of extension methods in the Members pane:

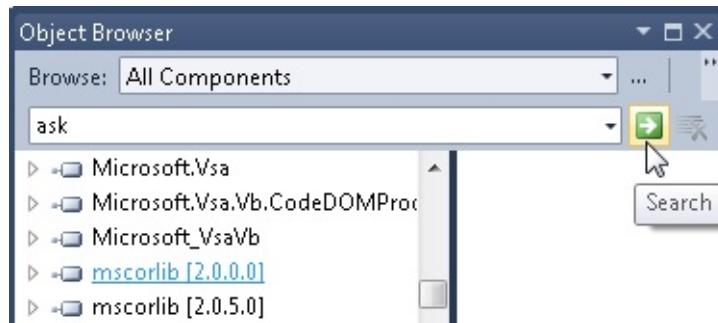


## AX.123 The Object Browser: Search

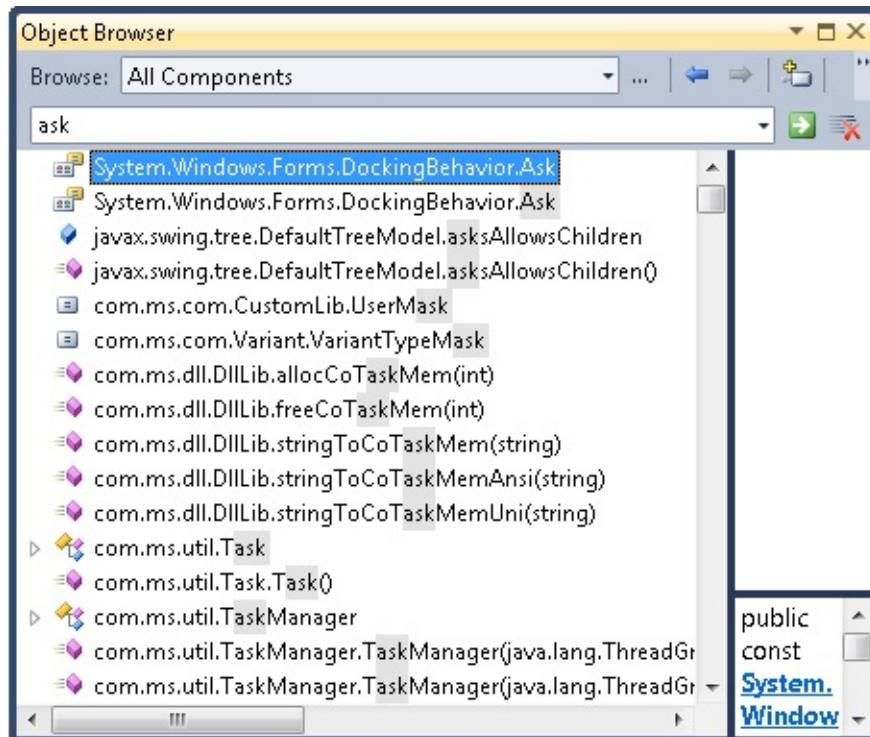
<b>DEFAULT</b>	Ctrl+K, Ctrl+R (goto to search)
<b>VISUAL BASIC 6</b>	Ctrl+K, Ctrl+R (goto to search)
<b>VISUAL C# 2005</b>	[no shortcut] (goto to search)
<b>VISUAL C++ 2</b>	[no shortcut] (goto to search)
<b>VISUAL C++ 6</b>	Ctrl+K, Ctrl+R (goto to search)
<b>VISUAL STUDIO 6</b>	Ctrl+K, Ctrl+R (goto to search)

<b>COMMAND</b>	View.ObjectBrowserGoToSearchCombo; View.ObjectBrowserClearSearch; View.ObjectBrowserSearch
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0081

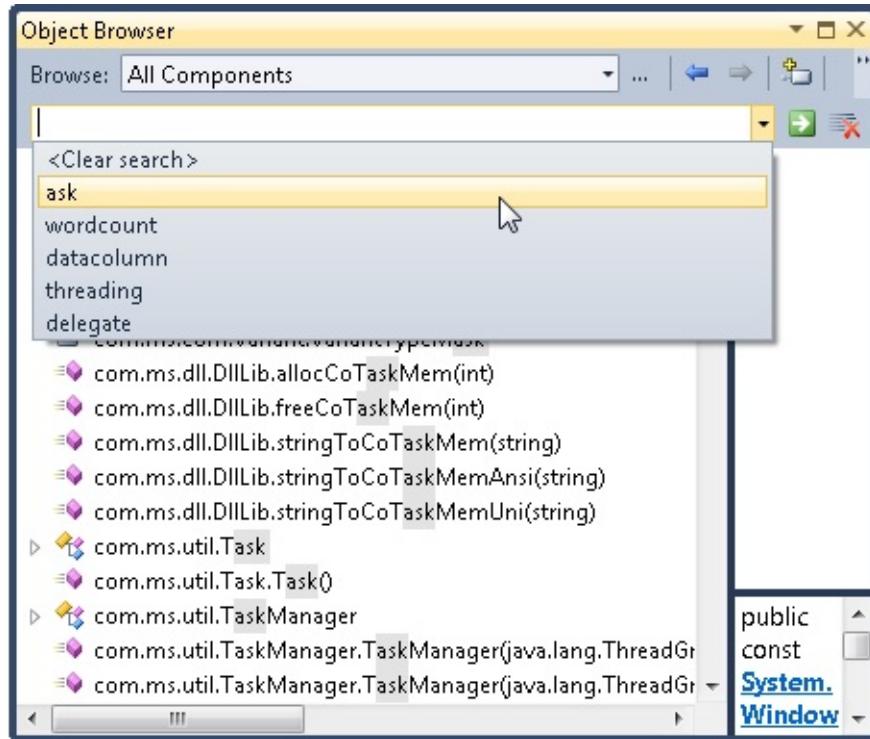
When you use the Object Browser, typically you need to find something fast. Search is a great way to find what you are looking for within the current browsing scope. To use search, just type the string you are looking for into the Search box and press Enter or click the Search button:



Searches are a “contains” operation and are not case-specific. For example, typing in the search term **ask** highlights the substring in the results. The search utility filters the Objects pane to show only those items that contain the search string:

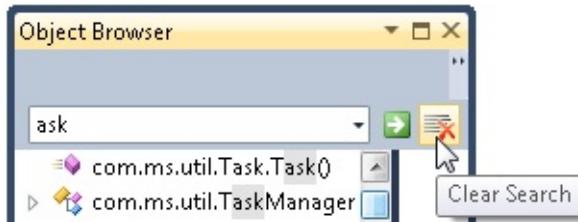


You can repeat any previous search by clicking the drop-down list arrow in the Search area:



This list persists even after Visual Studio is closed because the values are stored in the registry (HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\<version>\Object\_Browser):

You can clear any search (and remove the filter on the Objects pane) by clicking the Clear Search button to the right of the Search box:



## **View.ObjectBrowserSearch Command**

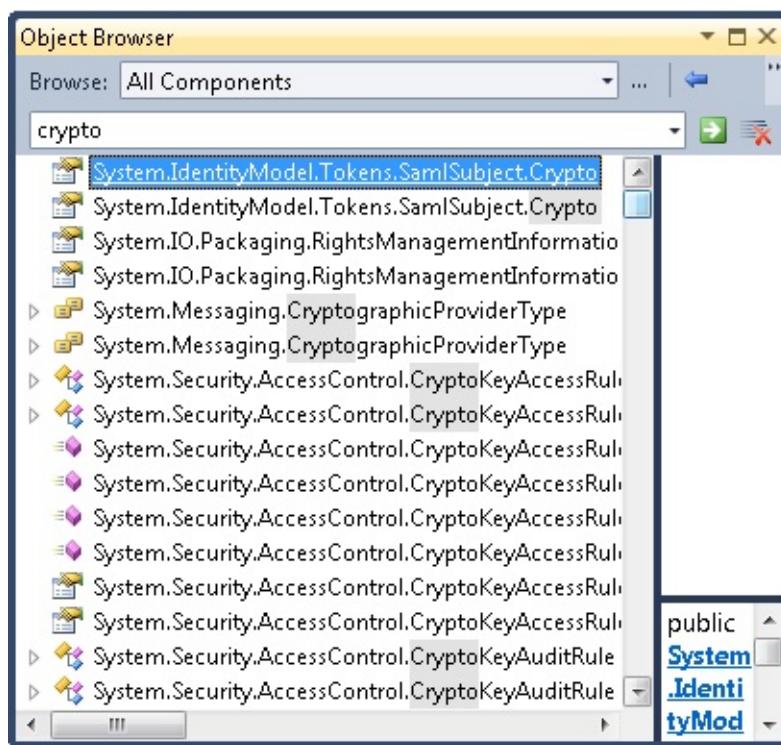
You can invoke the View.ObjectBrowserSearch command to quickly do a search by using a command. This is particularly useful if you have a search you perform frequently because you can create command aliases for your common search strings. See vstipTool0068 ([03.20 Understanding Commands: Aliases](#), page 113) for more information about command aliases.

The command is relatively straightforward. Just type in

**View.ObjectBrowserSearch [search string]:**



This yields a result based on the current Object Browser settings:

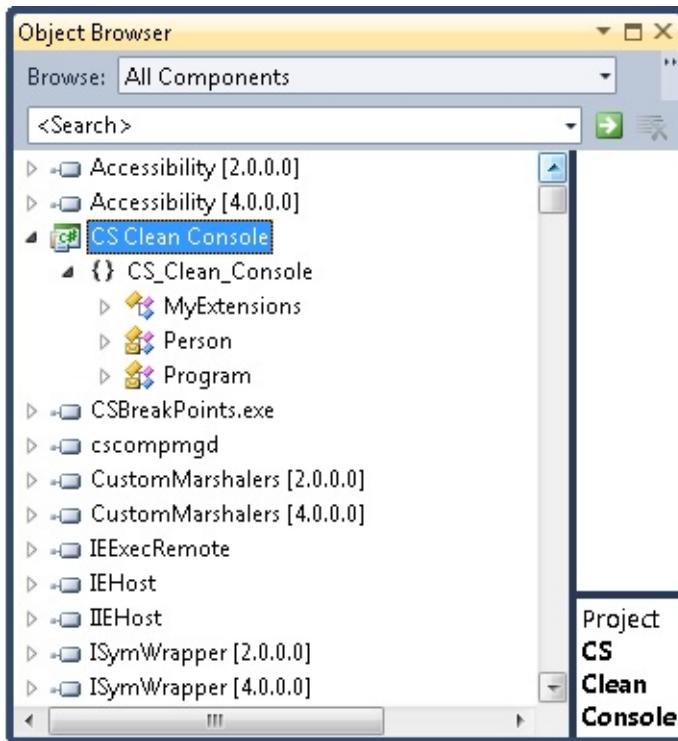


## AX.124 The Object Browser: Objects Pane

<b>DEFAULT</b>	Ctrl+Alt+J
<b>VISUAL BASIC 6</b>	Ctrl+Alt+J; F2
<b>VISUAL C# 2005</b>	Ctrl+Alt+J; Ctrl+W, Ctrl+J; Ctrl+W, J
<b>VISUAL C++ 2</b>	Ctrl+Alt+J; Shift+Alt+F1
<b>VISUAL C++ 6</b>	Ctrl+Alt+J

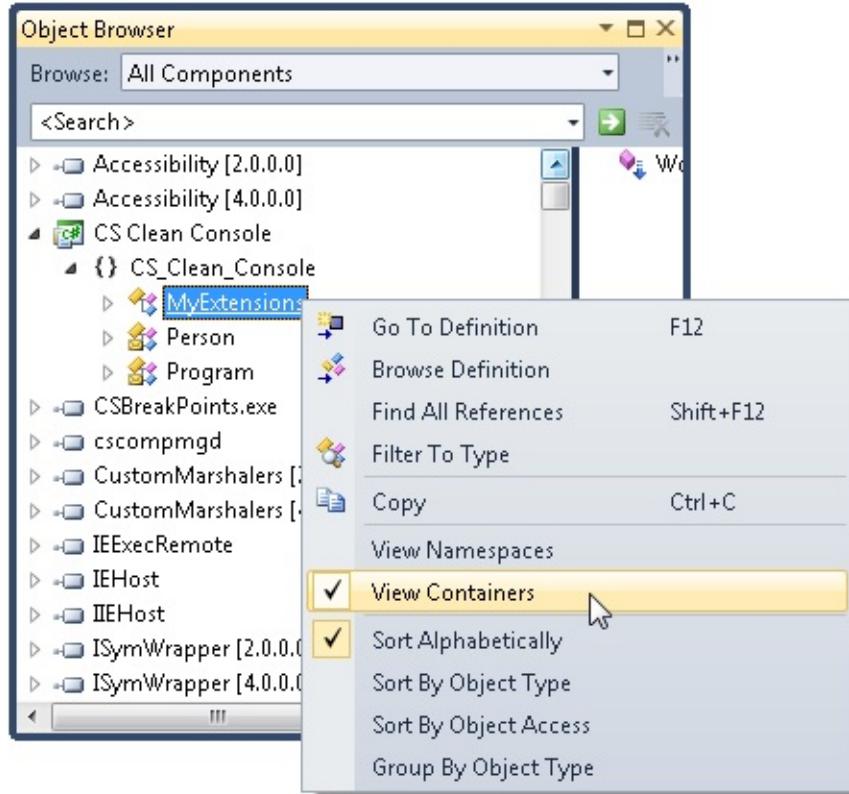
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+B; F2
<b>WINDOWS</b>	Alt,V, J
<b>MENU</b>	View   Object Browser
<b>COMMAND</b>	View.ObjectBrowser
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0082

When working with the Object Browser, you will inevitably find yourself in the Objects pane located just below the Search area:



The Objects pane displays an expandable list of symbols whose top-level nodes represent components or namespaces (based on your choice in the settings) available in the current browsing scope. These top-level nodes typically contain symbols that contain other symbols. To expand or collapse a node selected in the list, click its arrow sign or press Enter.

When you right-click in the Objects pane, you can see a list of options. What you see depends on the item chosen, but it generally looks something like the following:



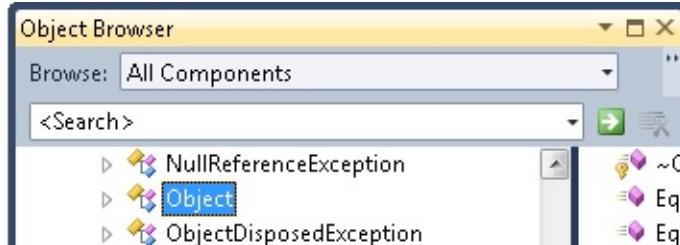
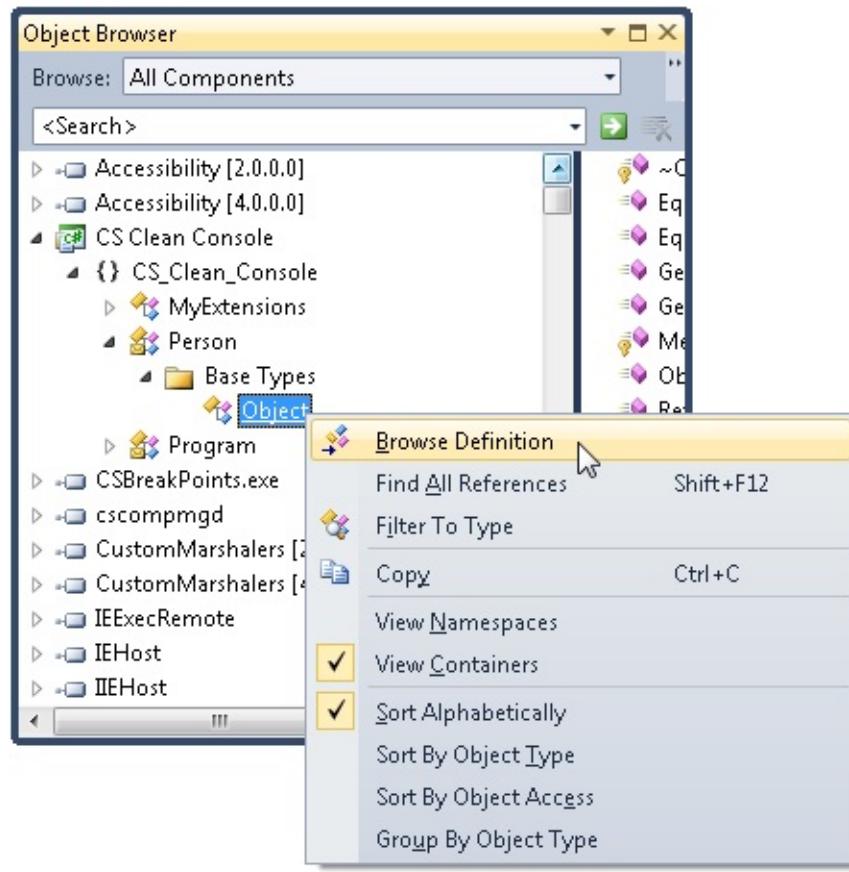
Following are descriptions of the possible options and what they do.

## Go To Definition

Takes you to the line of code where the item is defined:

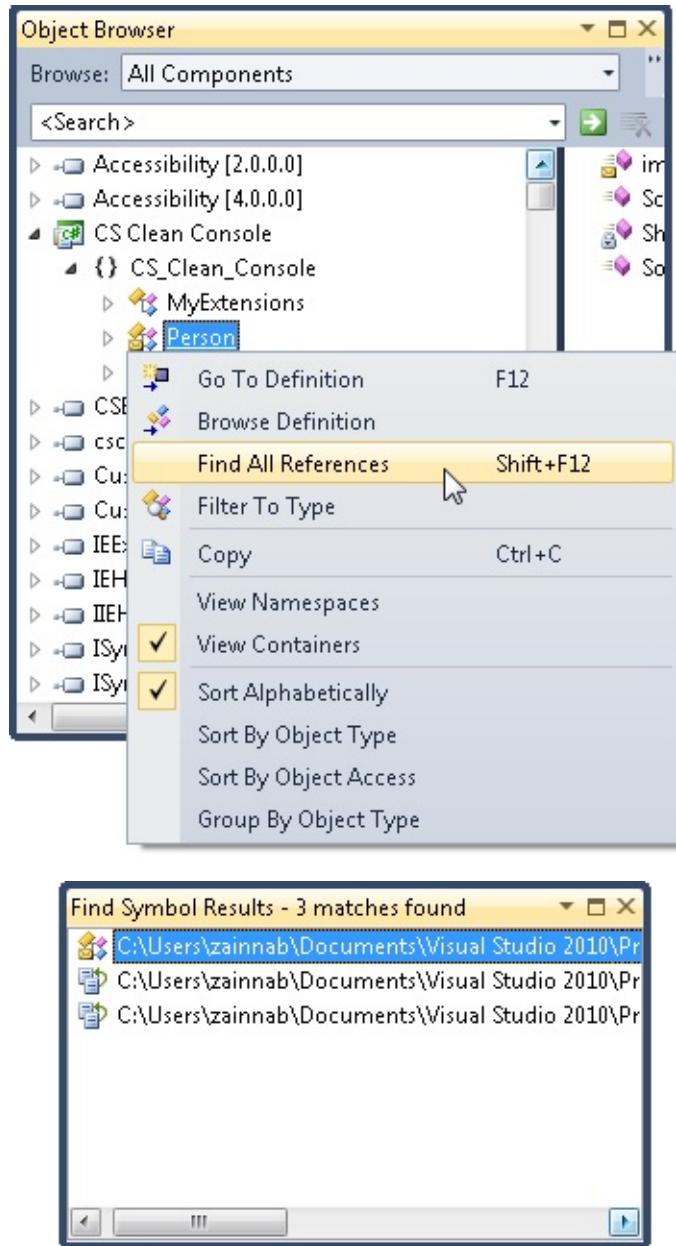
## Browse Definition

Takes you to the primary node (typically top level) for the selected symbol in the Object Browser:



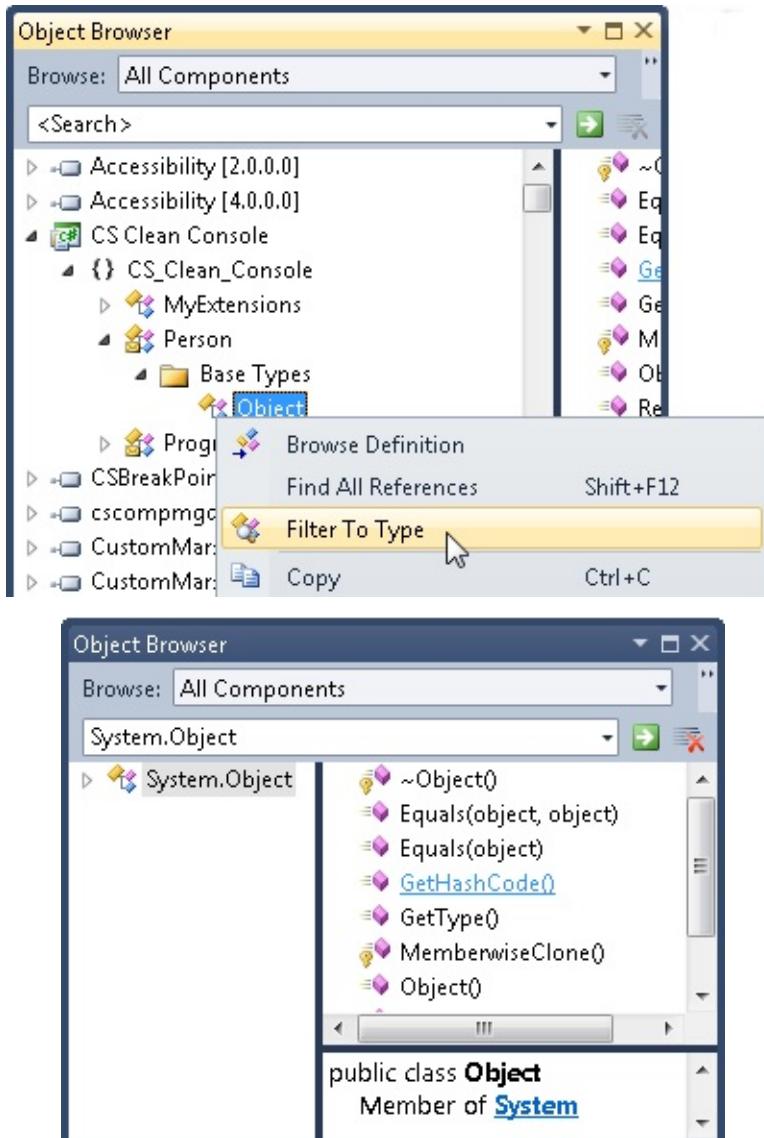
## Find All References

Performs a search on the currently selected object symbol by using the current browsing scope with results shown in the Find Symbol Results dialog box:



## Filter To Type

Shows only the selected type in the Objects pane. Essentially, it makes the selected item the top-level item in the pane. It is particularly useful for focusing on a single namespace or component:



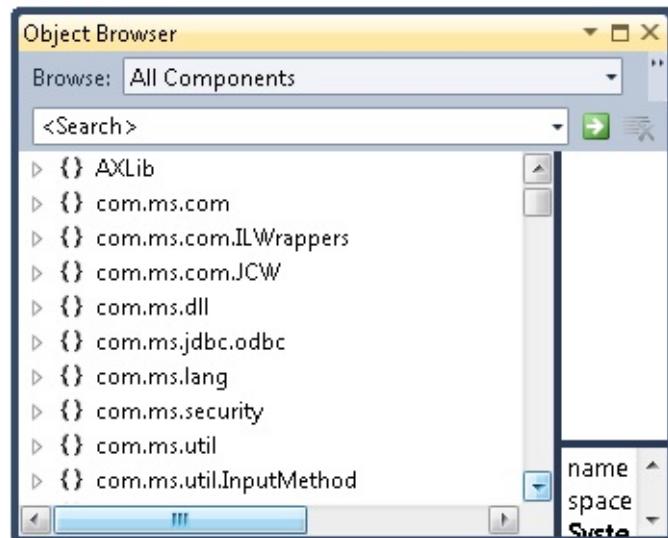
Filter To Type searches only on the item selected, and you can take the filter off by clicking the Clear Search button to the right of the Search box:

## **Copy**

Copies a symbol reference that can be pasted into a designer and also copies the full path and name of the selected item to the clipboard.

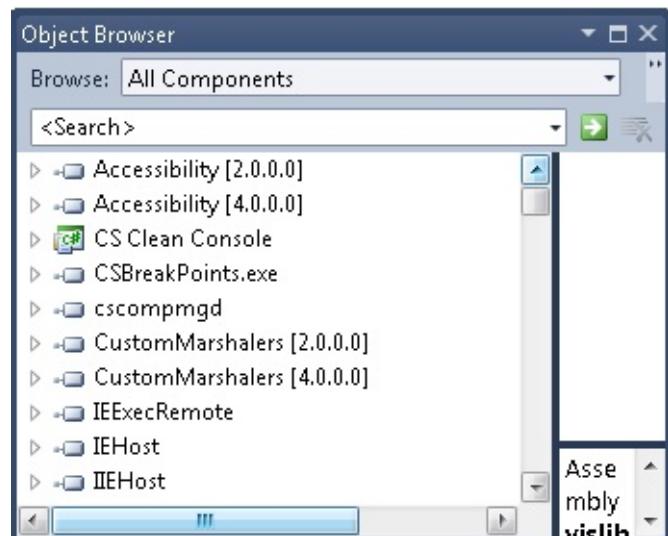
## **View Namespaces**

Sets the highest-level items in the Objects pane to logical namespaces. Namespaces stored in multiple physical containers are merged:



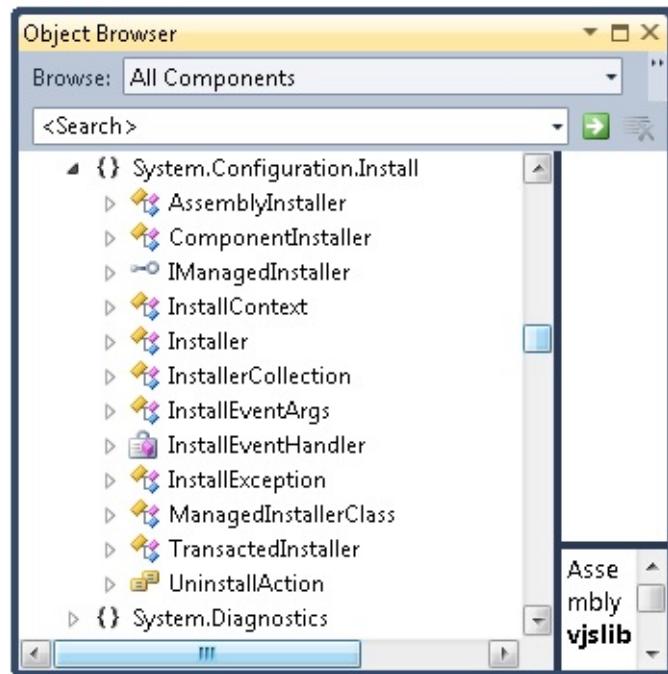
## View Containers

Sets the highest-level items in the Objects pane to physical containers, such as projects, components, assemblies, source browser (.bsc) files, and output type libraries (.tlb). These can be expanded to show the namespaces they contain:



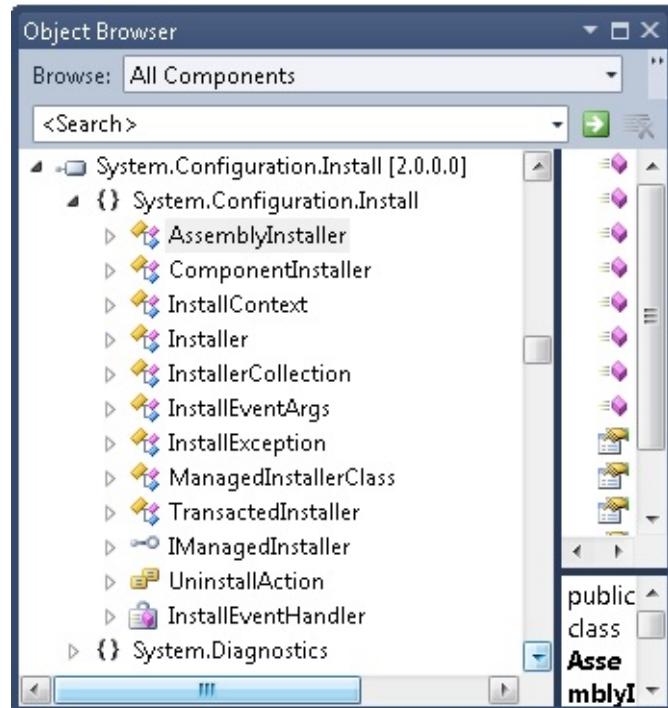
## Sort Alphabetically

Lists items alphabetically by their names in ascending order:



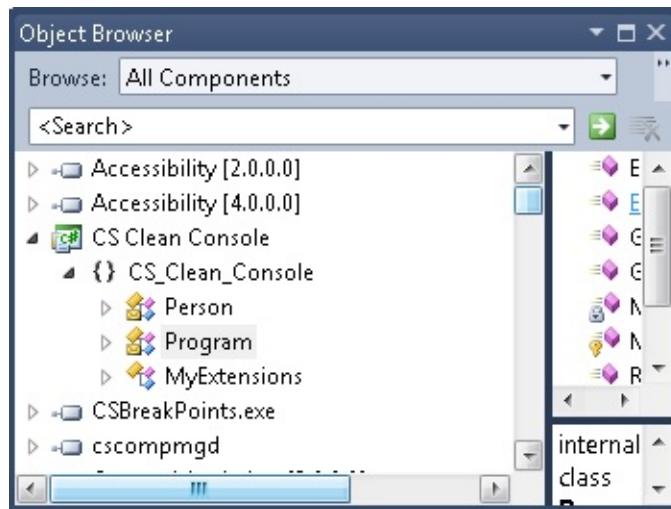
## Sort By Object Type

Lists items in order of their type, such as base classes, followed by derived classes, interfaces, methods, and so forth:



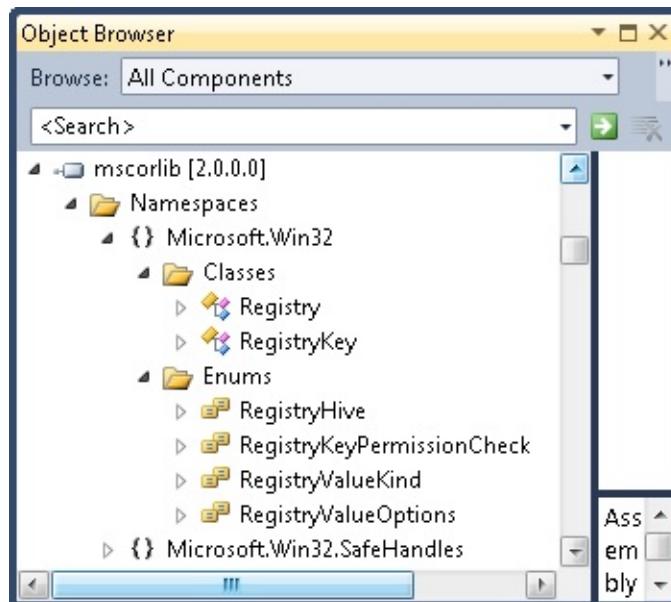
## Sort By Object Access

Lists items in order of their access type, such as public or private:



## Group By Object Type

Sorts items into groups by type, such as classes, interfaces, properties, methods, and so on. This is a great organizational feature:



## Go To Declaration

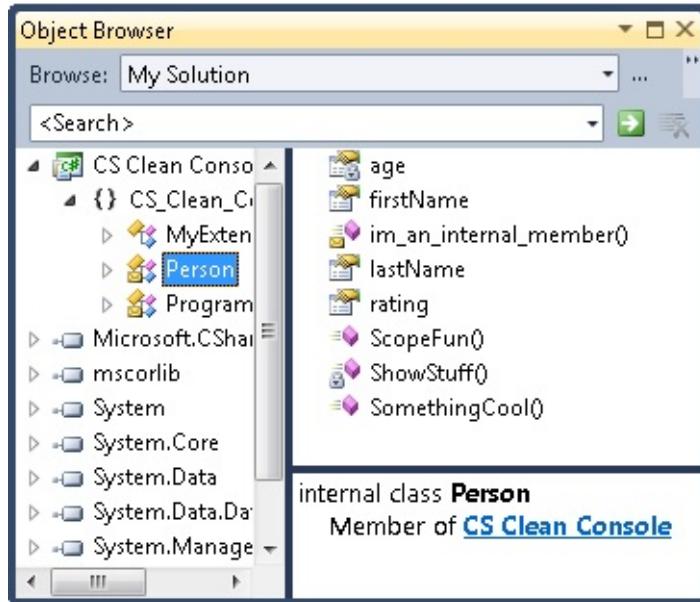
Takes you to the declaration of the symbol in the code. This is available only in Visual C++ projects.

## AX.125 The Object Browser: Members Pane

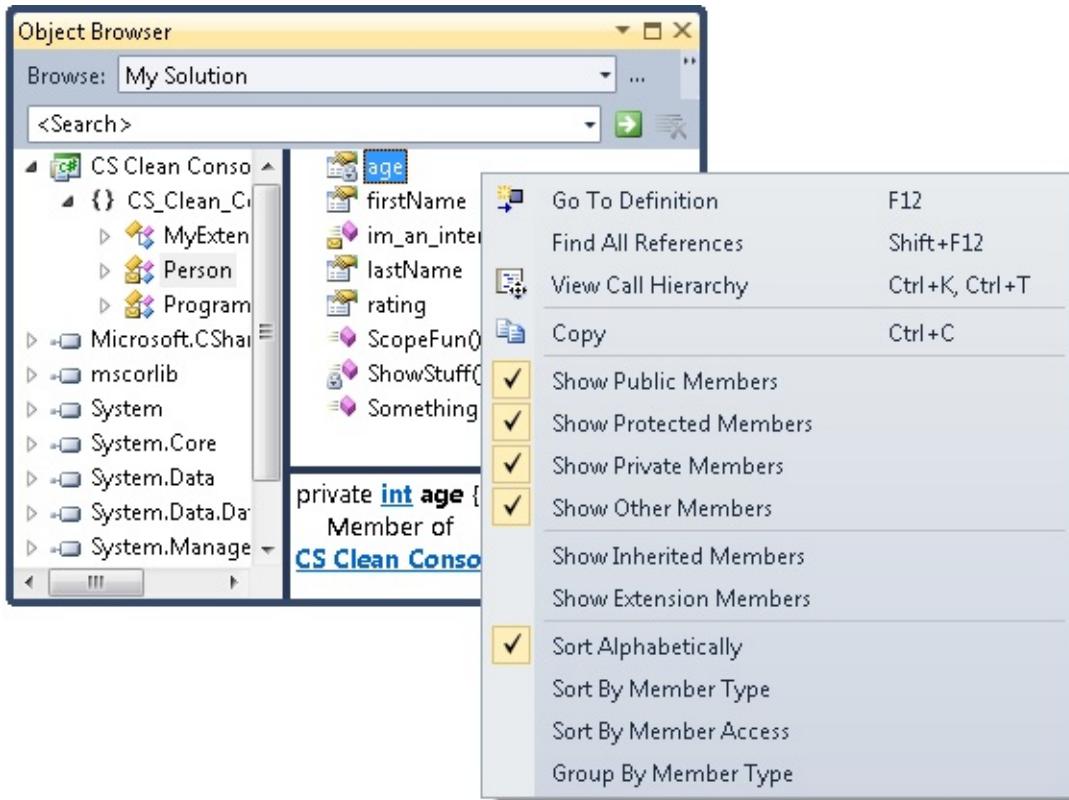
DEFAULT	Ctrl+Alt+J
---------	------------

<b>VISUAL BASIC 6</b>	Ctrl+Alt+J; F2
<b>VISUAL C# 2005</b>	Ctrl+Alt+J; Ctrl+W, Ctrl+J; Ctrl+W, J
<b>VISUAL C++ 2</b>	Ctrl+Alt+J; Shift+Alt+F1
<b>VISUAL C++ 6</b>	Ctrl+Alt+J
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+B; F2
<b>WINDOWS</b>	Alt, V, J
<b>MENU</b>	View   Object Browser
<b>COMMAND</b>	View.ObjectBrowser
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0083

Each object can contain members such as properties, methods, events, constants, variables, and enum values. Selecting an object in the Objects pane (left) displays its members in the Members pane (right):



While in the Members pane, you can do several things. Many of these activities are duplicates of actions you can take in other areas of the Object Browser, so I will reference those areas to avoid duplication:



## Go To Definition, Find All References, and Copy

These are the same as in the Objects pane (vstipTool0082, [AX.124 The Object Browser: Objects Pane](#), page A195).

## View Call Hierarchy

This command is unique to the Members pane and opens up the Call Hierarchy window (vstipTool0005, [07.18 Using the Call Hierarchy](#), page 310).

## Show \*

These options are the same as the ones available in the Object Browser settings (vstipTool0080, [AX.122 The Object Browser: Settings](#), page A186).

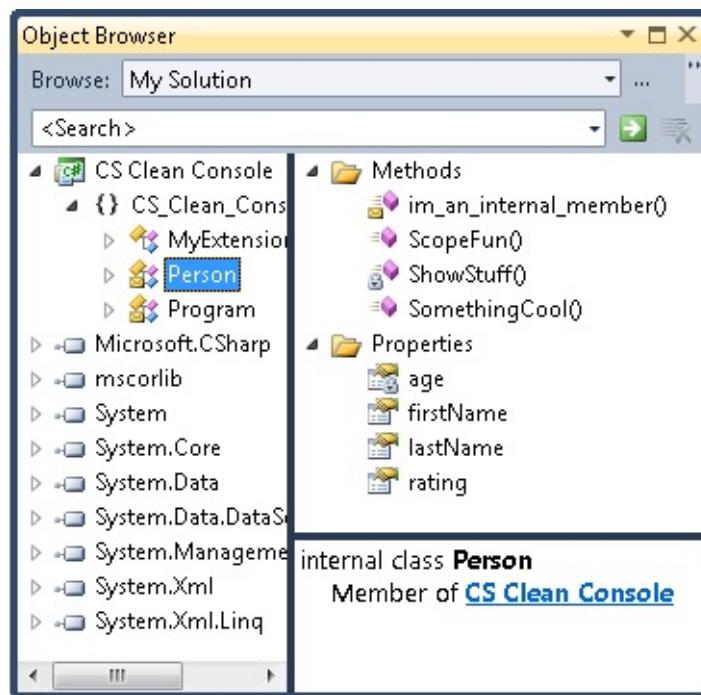
## Sort \*

These options are the same as the ones available in the Objects pane (vstipTool0082, [AX.124 The Object Browser: Objects Pane](#), page 124).

## Group By Member Type

This is similar to the same option in the Objects pane but different enough to warrant a quick look. When you use this option, members are grouped into their

respective types:

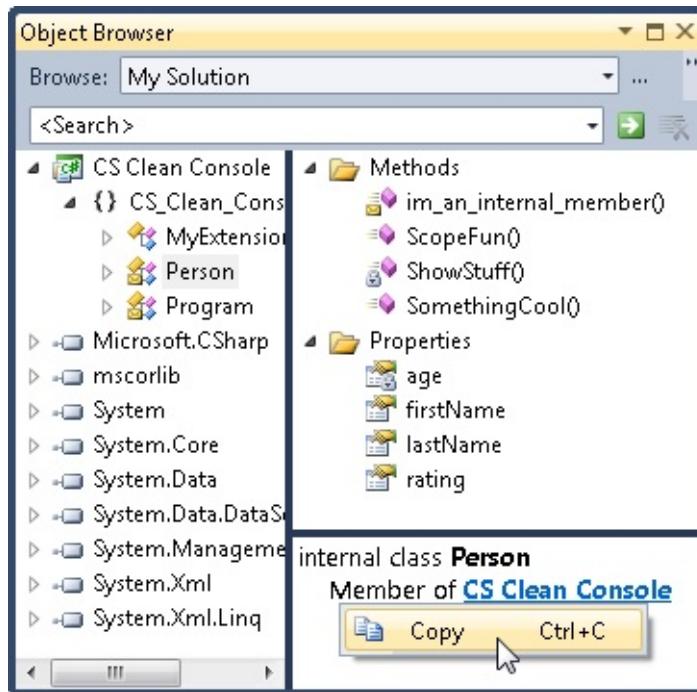


## AX.126 The Object Browser: Description Pane

<b>DEFAULT</b>	Ctrl+Alt+J
<b>VISUAL BASIC 6</b>	Ctrl+Alt+J; F2
<b>VISUAL C# 2005</b>	Ctrl+Alt+J; Ctrl+W, Ctrl+J; Ctrl+W, J
<b>VISUAL C++ 2</b>	Ctrl+Alt+J; Shift+Alt+F1
<b>VISUAL C++ 6</b>	Ctrl+Alt+J
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+B; F2
<b>WINDOWS</b>	Alt, V, J
<b>MENU</b>	View   Object Browser
<b>COMMAND</b>	View.ObjectBrowser
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0084

The Description pane (bottom right) displays detailed information about the currently selected item (Objects pane) or member (Members pane). You can copy

(right-click anywhere in the pane) data from the Description pane to the clipboard and then paste it into the code editor:



The information displayed depends on the selection and can include the following:

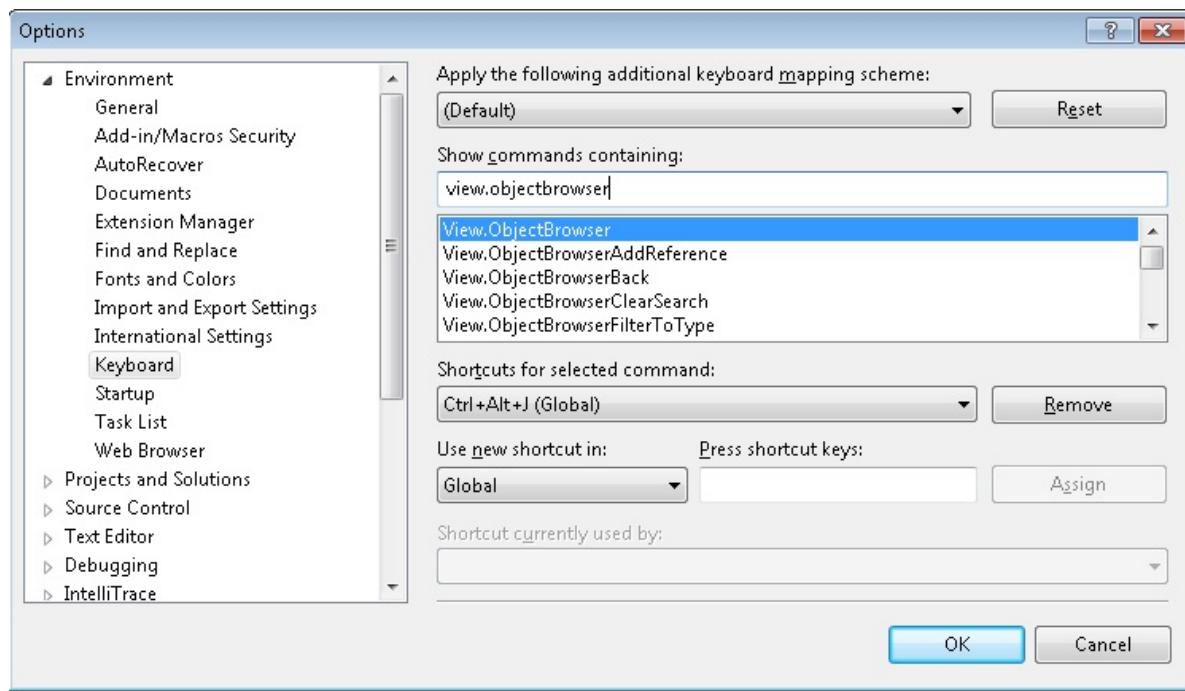
- Name and parent object
- Properties and attributes
- Syntax in the programming language of the active project
- Links to related objects and members
- Descriptions, comments, and Help text
- Version of the .NET Framework in which the object or member is included

## AX.127 The Object Browser: Creating a Keyboard Shortcut for Add To References

<b>COMMAND</b>	View.ObjectBrowserAddReference
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0085

A command is available for almost anything you can do in the Object Browser. You can see this by going to Tools | Options | Keyboard and typing in

**view.objectbrowser** in the Show Commands Containing area:

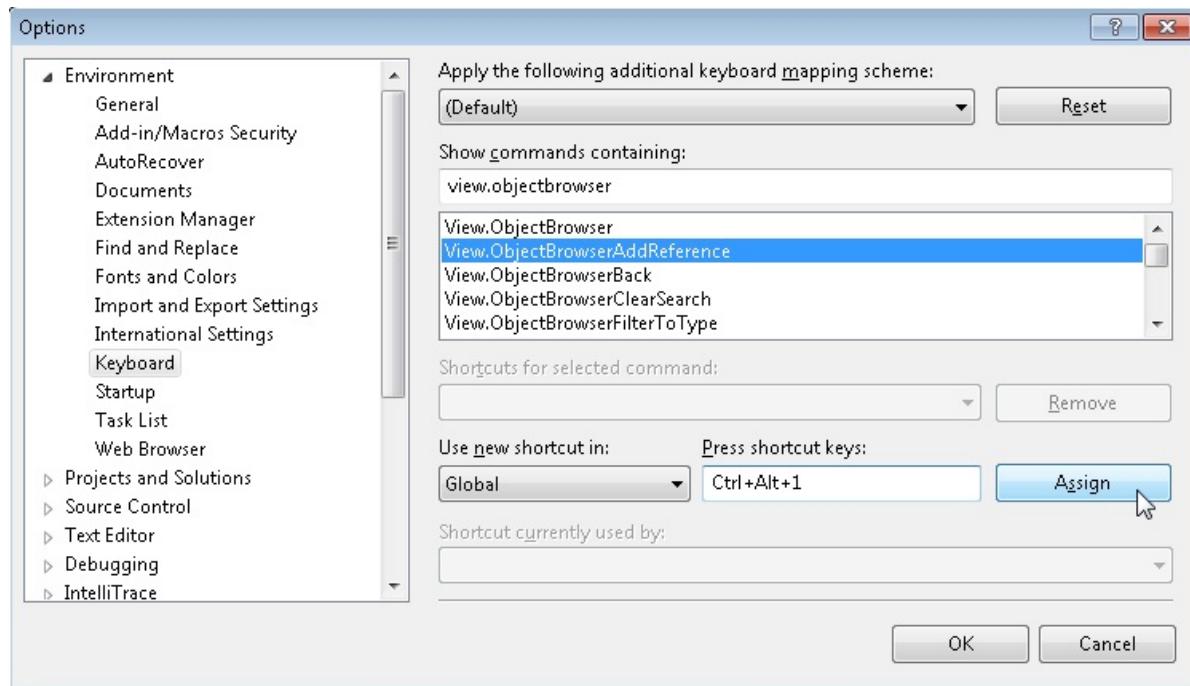


This means that you can create a shortcut key for all kinds of activities. (See vstipTool0063, [03.26 Keyboard Shortcuts: Creating New Shortcuts](#), page 127.) I'll provide a short example here.

So let's say you want to make it easy to get the Add To References In Selected Project In Solution Explorer functionality available on the toolbar in a keyboard shortcut:



Just go to Tools | Options | Keyboard, type **view.objectbrowserraddreference** in the Show Commands Containing area, enter the shortcut you want to use, and click Assign:

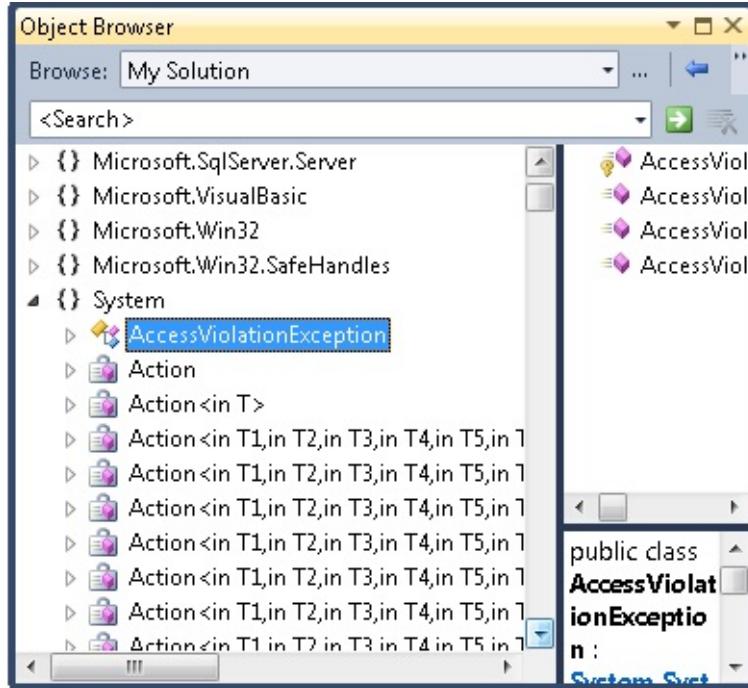


You now have a shortcut key you can use anytime you want instead of having to use the toolbar.

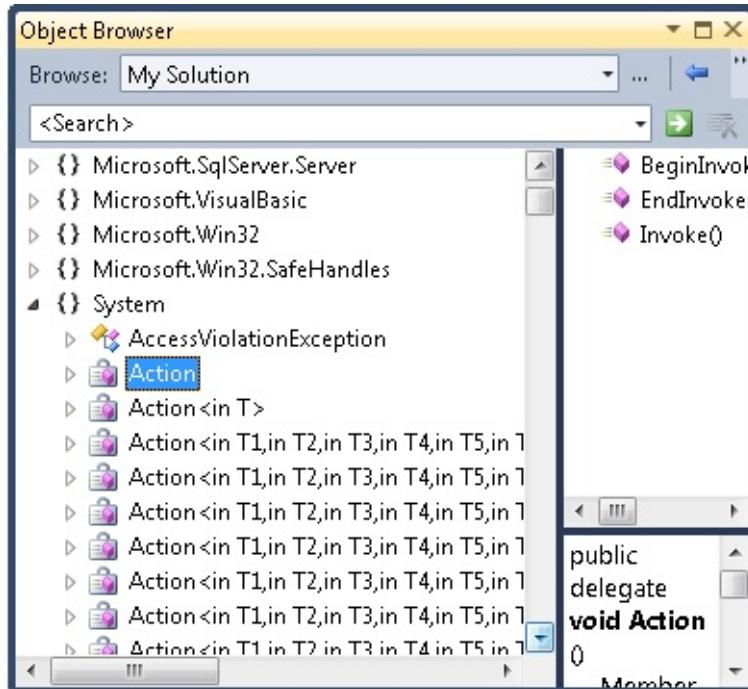
## AX.128 The Object Browser: Type-Ahead Selection

<b>DEFAULT</b>	Ctrl+Alt+J
<b>VISUAL BASIC 6</b>	Ctrl+Alt+J; F2
<b>VISUAL C# 2005</b>	Ctrl+Alt+J; Ctrl+W, Ctrl+J; Ctrl+W, J
<b>VISUAL C++ 2</b>	Ctrl+Alt+J; Shift+Alt+F1
<b>VISUAL C++ 6</b>	Ctrl+Alt+J
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+B; F2
<b>WINDOWS</b>	Alt,V, J
<b>MENU</b>	View   Object Browser
<b>COMMAND</b>	View.ObjectBrowser
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0086

Type-ahead support is available in the Object Browser lists. For example, I have a list of items in the Objects pane:



I type **act**, and I get the following result:

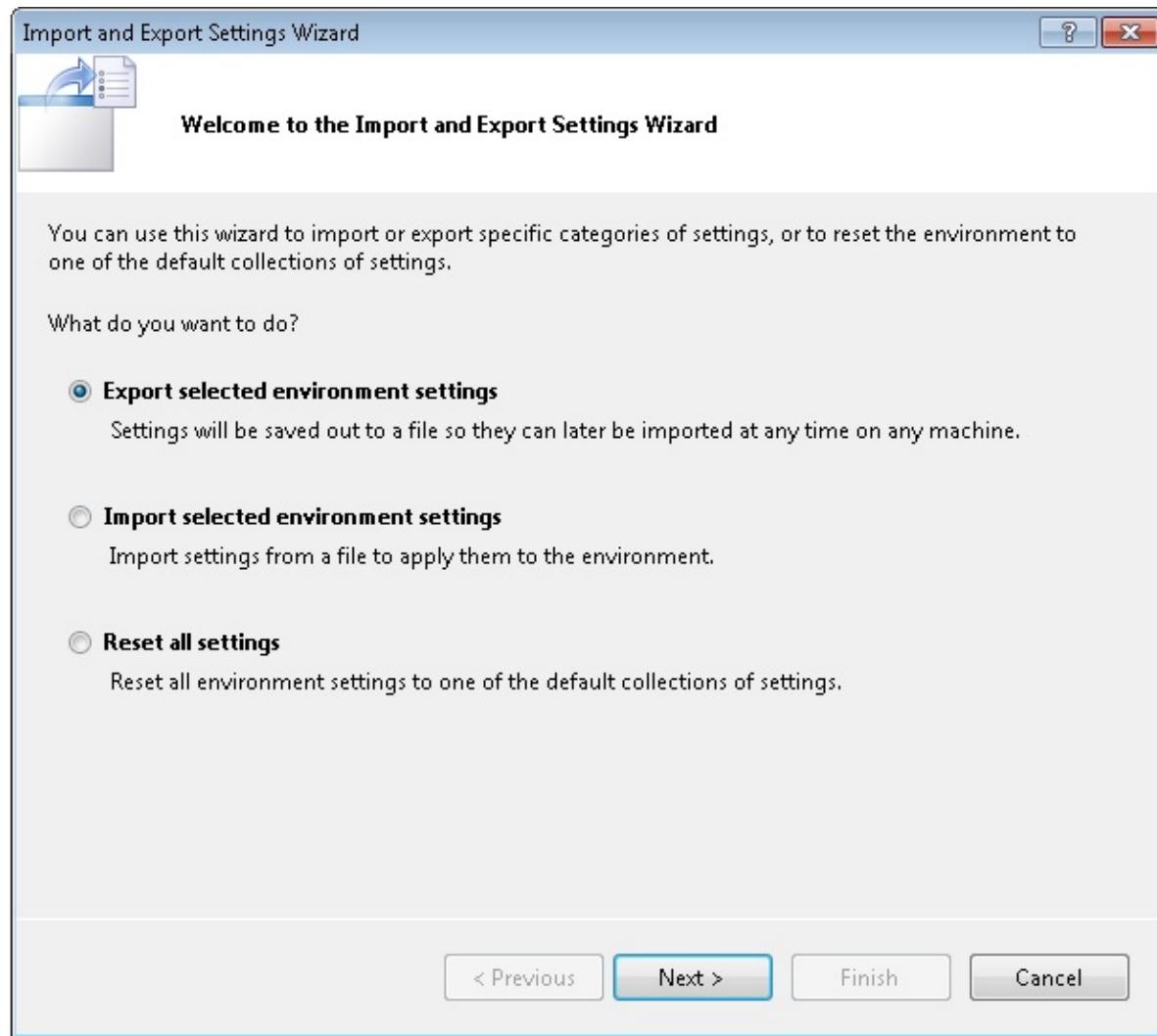


As you can see, it takes me to the first item that begins with *act*. I can continue typing until I find exactly what I want, or I can browse from there.

## AX.129 The Object Browser: Exporting Your Settings

<b>WINDOWS</b>	Alt,T, I
<b>MENU</b>	Tools   Import and Export Settings
<b>COMMAND</b>	Tools.ImportandExportSettings
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0087

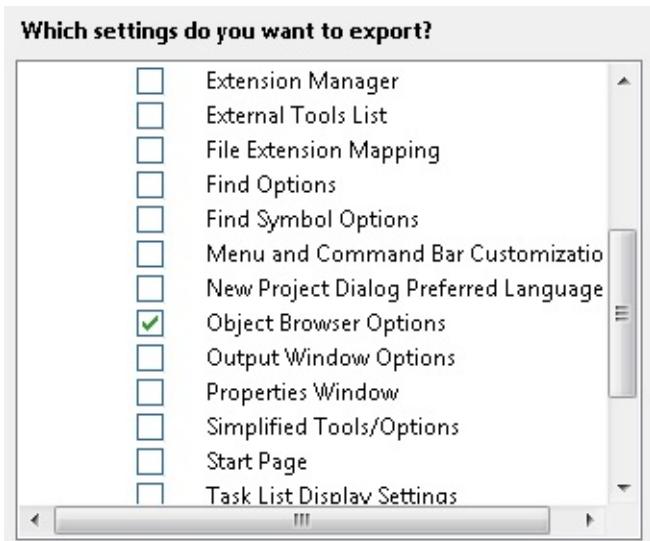
After you have the Object Browser configured the way you want it, you probably want to export the settings. In fact, you might have several different configurations you use, depending on the circumstances. For more information about exporting, see vstipEnv0021 ([01.03 Exporting Your Environment Settings](#), page 6). For now, let's just use a quick example to get your Object Browser settings exported. First, go to Tools | Import And Export Settings and choose Export Selected Environment Settings:



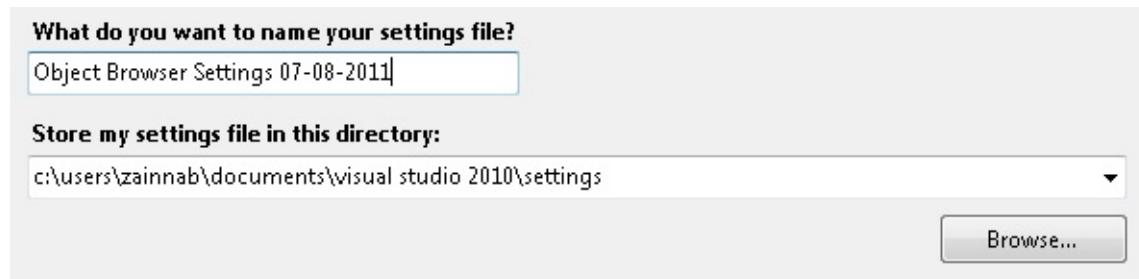
Click Next, and then clear the All Settings box to clear out all the currently selected items:



Now select Object Browser Options under General Settings:



Click Next, and then give the .vssettings file a name and the path to store it in:



Click Finish, click Close, and you are all set to go. Anytime you need these settings again, you can import them.

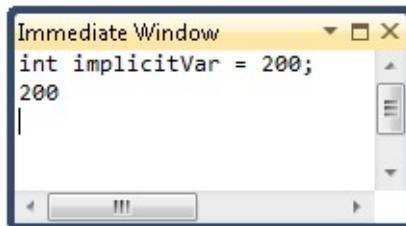
## AX.130 The Immediate Window: Implicit Variables

<b>DEFAULT</b>	Ctrl+Alt+I
<b>VISUAL BASIC 6</b>	Ctrl+Alt+I; Ctrl+G
<b>VISUAL C# 2005</b>	Ctrl+Alt+I; Ctrl+D, Ctrl+I; Ctrl+D, I
<b>VISUAL C++ 2</b>	Ctrl+Alt+I
<b>VISUAL C++ 6</b>	Ctrl+Alt+I
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+I
<b>WINDOWS</b>	Alt,D, W, I
<b>MENU</b>	Debug   Windows   Immediate
<b>COMMAND</b>	Debug.Immediate
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipTool0100

In the Immediate Window, you can create implicit variables for use in your debugging efforts. These implicit variables never go out of scope and can be treated as any other variable. They are approached differently in VB and C#.

### C#

To create an implicit variable in C#, just declare any variable in the Immediate window:

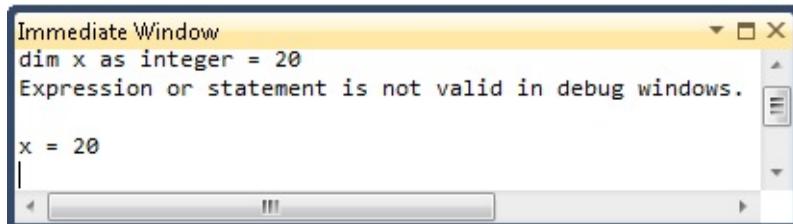


If you are in debug mode, you can actually see the variable in your Locals window. Implicit variables show up with a "\$" character in front of them:

Locals		
Name	Value	Type
args	{string[0] string}	
+ myPerson	[CS_Clea CS_CI]	
a	0	int
\$implicitVar	200	int

## VB

In VB, you cannot declare implicit variables in the Immediate Window. But if you use an undeclared variable, an implicit variable is created automatically. Unfortunately, VB implicit variables are not listed in the Locals window:



```
Immediate Window
dim x as integer = 20
Expression or statement is not valid in debug windows.

x = 20
|
```

## AX.131 Show External Code

VERSIONS	2005, 2008, 2010
CODE	vstipDebug0031

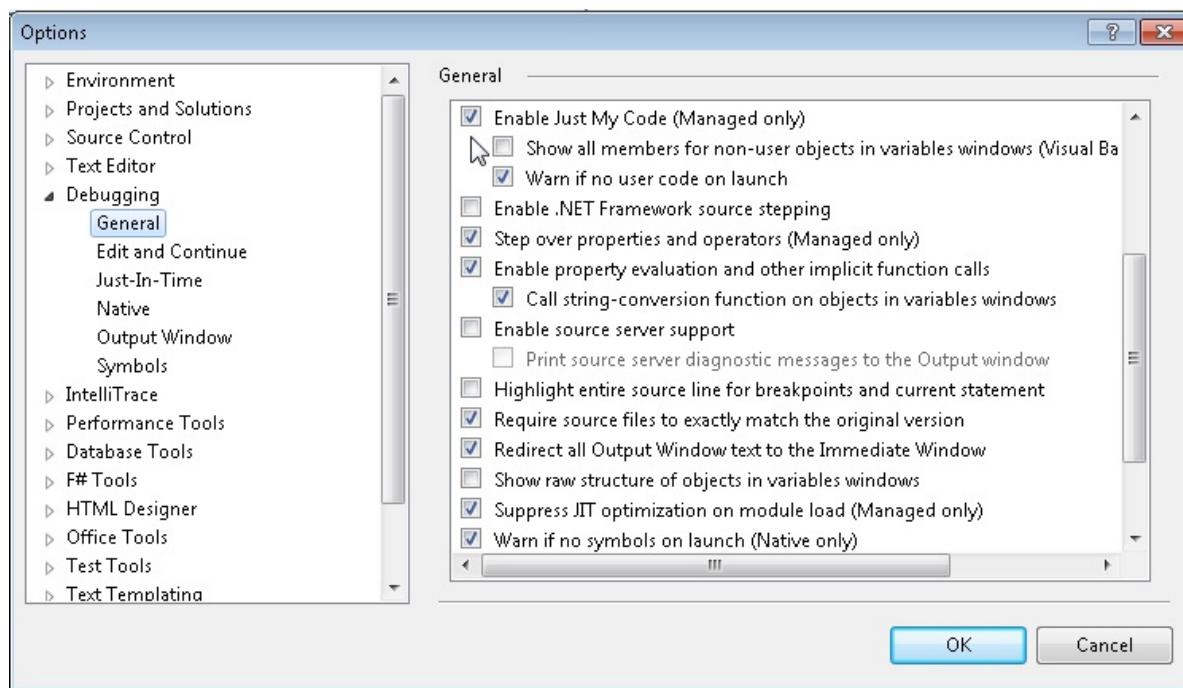
The Call Stack window provides an option to show external code. Let's start with the basics. When you are in break mode and you look at a "normal" call stack, you see the following:

### NOTE

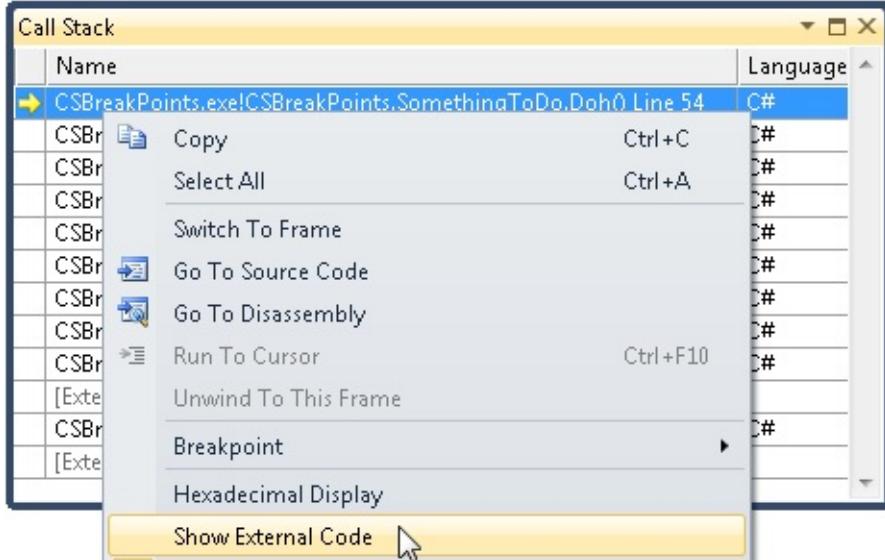
The Call Stack window (Ctrl+Alt+C) is only available while debugging.

Name	Language
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Doh() Line 54	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Ti() Line 49 + 0xa	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.La() Line 44 + 0x:	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.So() Line 39 + 0x:	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Fa() Line 34 + 0x:	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Mi(string s) Line	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Re(double d) Lin	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Do(int iParam, ir	C#
CSBreakPoints.exe!CSBreakPoints.MainForm.buttonDeepStack_Click	C#
[External Code]	
CSBreakPoints.exe!CSBreakPoints.Program.Main() Line 17 + 0x28 by	C#
[External Code]	

Let's define what "normal" is in this case. Essentially, what you see here is determined by the Enable Just My Code (Managed Only) setting in Tools | Options | Debugging | General:



This setting is on by default, and it is the reason you see the "[External Code]" sections in your Call Stack. Selecting Enable Just My Code (Managed Only) means that you want to see your code without any information getting in the way. If you want to see the details of "[External Code]," just right-click anywhere in the Call Stack and choose Show External Code:



Now you should be able to see the external calls:

Name	Language
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Doh() Line 54	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Ti() Line 49 + 0xa	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.La() Line 44 + 0x	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.So() Line 39 + 0x	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Fa() Line 34 + 0x	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Mi(string s) Line	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Re(double d) Lin	C#
CSBreakPoints.exe!CSBreakPoints.SomethingToDo.Do(int iParam, ir	C#
CSBreakPoints.exe!CSBreakPoints.MainForm.buttonDeepStack_Click	C#
System.Windows.Forms.dll!System.Windows.Forms.Control.OnClick	
System.Windows.Forms.dll!System.Windows.Forms.Button.OnMou	
System.Windows.Forms.dll!System.Windows.Forms.Control.WmMc	
System.Windows.Forms.dll!System.Windows.Forms.Control.WndPr	
System.Windows.Forms.dll!System.Windows.Forms.ButtonBase.Wn	
System.Windows.Forms.dll!System.Windows.Forms.Button.WndPro	
System.Windows.Forms.dll!System.Windows.Forms.Control.Contro	
System.Windows.Forms.dll!System.Windows.Forms.NativeWindow.	
[Native to Managed Transition]	
[Managed to Native Transition]	
System.Windows.Forms.dll!System.Windows.Forms.Application.Co	
System.Windows.Forms.dll!System.Windows.Forms.Application.Th	
System.Windows.Forms.dll!System.Windows.Forms.Application.Th	
CSBreakPoints.exe!CSBreakPoints.Program.Main() Line 17 + 0x28 by	C#
[Native to Managed Transition]	
[Managed to Native Transition]	
Microsoft.VisualStudio.HostingProcess.Utilities.dll!Microsoft.VisualS	
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threa	
mscorlib.dll!System.Threading.ThreadHelper.ThreadStart() + 0x4d b	

The grey part is where “[External Code]” used to be. Let’s zoom in on a couple of the entries:

System.Windows.Forms.dll!System.Windows.Forms.Control.OnClick
System.Windows.Forms.dll!System.Windows.Forms.Button.OnMouse
System.Windows.Forms.dll!System.Windows.Forms.Control.WmMc
System.Windows.Forms.dll!System.Windows.Forms.Control.WndPr
System.Windows.Forms.dll!System.Windows.Forms.ButtonBase.Wn

Notice that now you are looking into the details of what is happening. It remains this way until you turn it off again.

By the way, if you ever need to turn this feature off, just right-click the Call Stack again and select Show External Code. It turns off this feature, and you are back to the original view.

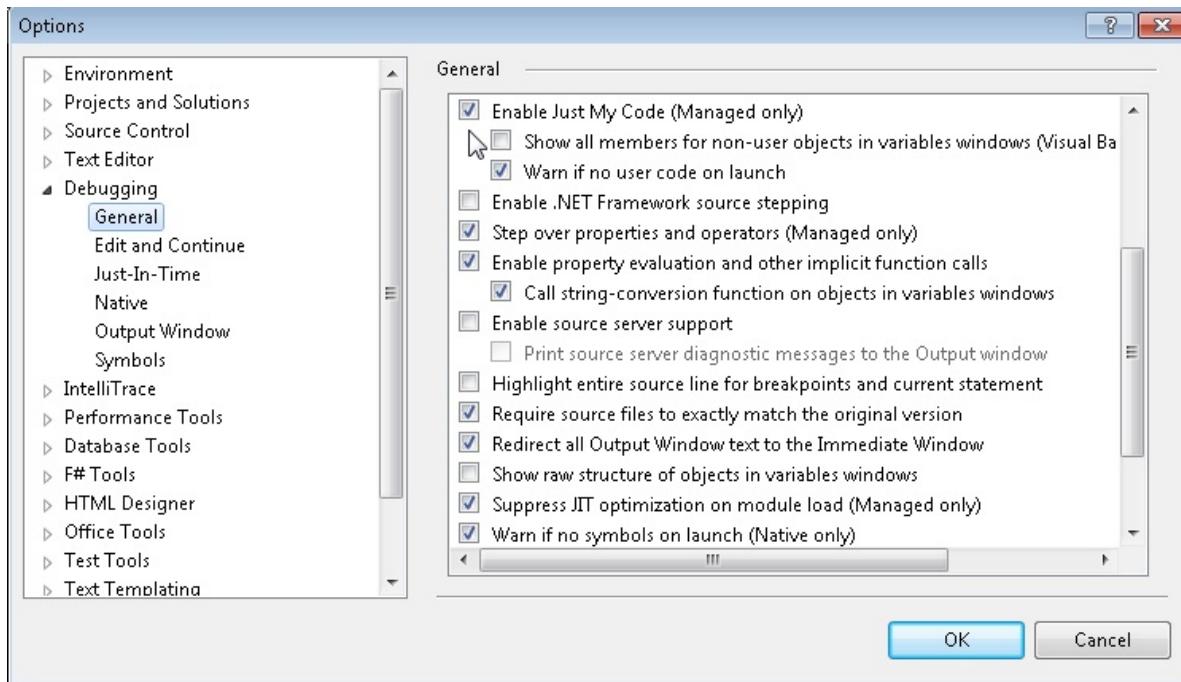
There is no option to turn this feature on and off, but just for reference, this setting is stored in the registry at

HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\10.0\Debugger under ShowExternalCode.

## AX.132 Understanding Just My Code

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>LANGUAGES</b>	C++ (managed only), C#, VB
<b>CODE</b>	vstipDebug0032

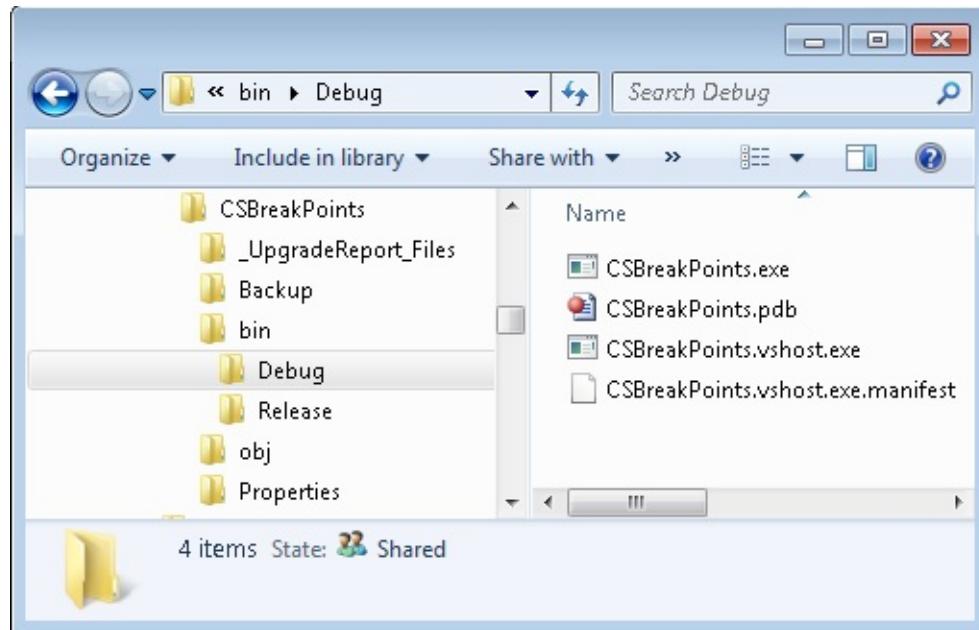
You often want to debug just the code you have written and exclude any Framework or external component code that you are using. If you go to Tools | Options | Debugging | General, you can find an option to Enable Just My Code (Managed Only):



Lots of people wonder what “Just My Code” really means. So let’s start with what the documentation says (<http://msdn.microsoft.com/en-us/library/h5e30exc.aspx>):

*“To distinguish user code from non-user code, Just My Code looks at three things: DBG Files, PDB files, and optimization.”*

## DBG and PDB files



One way to figure out what is “your code” is to look and see whether it has DBG and/or PDB files. In case you didn’t know, DBG files have been superseded by

the PDB format.

The PDB extension stands for “program database.” It holds the debugging information that was introduced in Visual C++ version 1.0. You can find out more about PDB files at <http://support.microsoft.com/kb/121366/en-us>. This is typically deep-level information about the source, so if you have these files, either it’s your code or someone trusted you enough to give them to you. In other words, it is basically “yours.”

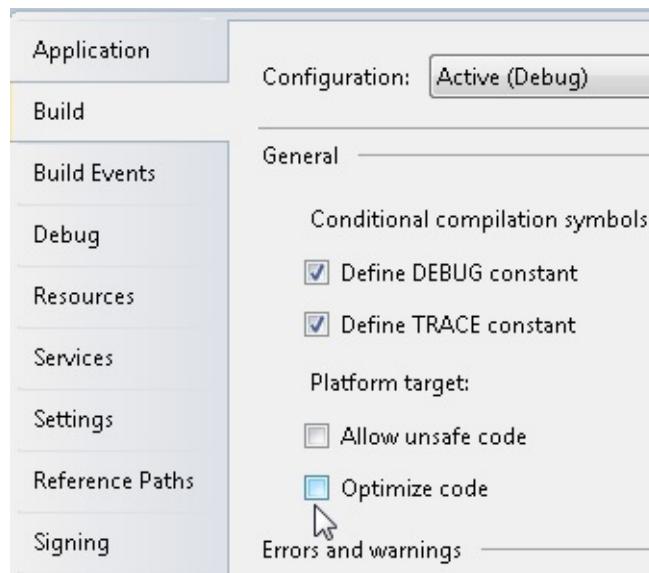
## Optimization

When optimization is turned off (the default setting for Debug builds), it factors into the code being considered “yours” as well. As stated in the documentation mentioned earlier, the optimization “option enables or disables optimizations performed by the compiler to make your output file smaller, faster, and more efficient.”

Many optimizations are available, such as optimizing for application speed or the size of your program. You can get see the full list of features at <http://msdn.microsoft.com/en-us/library/k1ack8f1.aspx>. Basically, optimization does many things that are great for a shipping application but not for one that you are currently debugging.

## C#

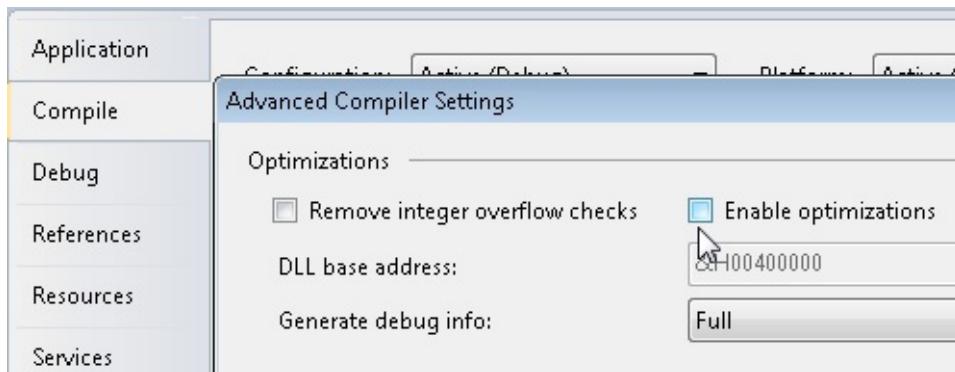
In C#, this is found in the Project properties on the Build tab:



## VB

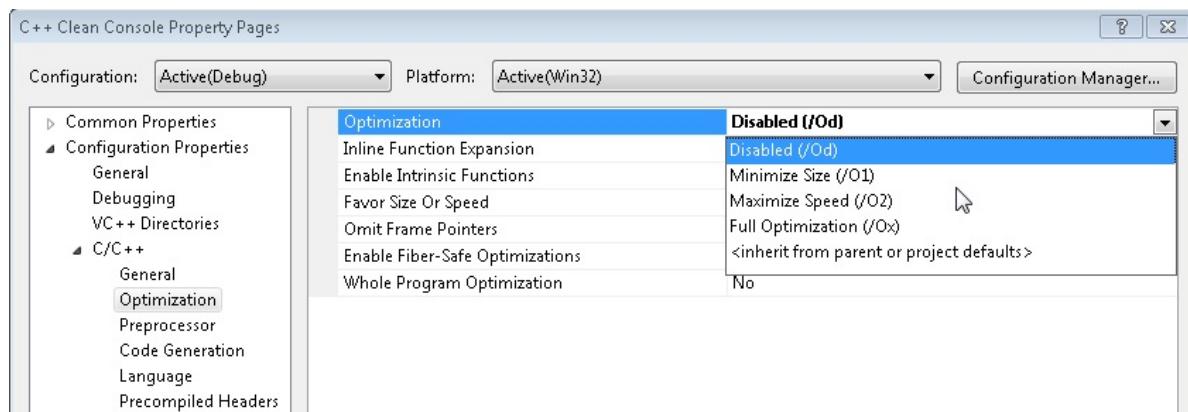
In VB, it is in the Project properties on the Compile tab, but you have to click the

Advanced Compiler Settings button:



## C++

In C++, you can find optimization options under the Project properties, under C/C++, Optimization:



## Finally

So if the PDB information is there and optimization is *not* turned on, the code is considered “yours” as far as Visual Studio is concerned.

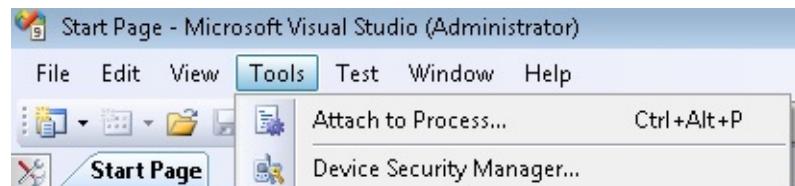
## AX.133 Attach To Process (Tools vs. Debug Menu)

<b>DEFAULT</b>	Ctrl+Alt+P
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+P; Ctrl+Shift+R
<b>VISUAL C# 2005</b>	Ctrl+Alt+P
<b>VISUAL C++ 2</b>	Ctrl+Alt+P
<b>VISUAL C++ 6</b>	Ctrl+Alt+P

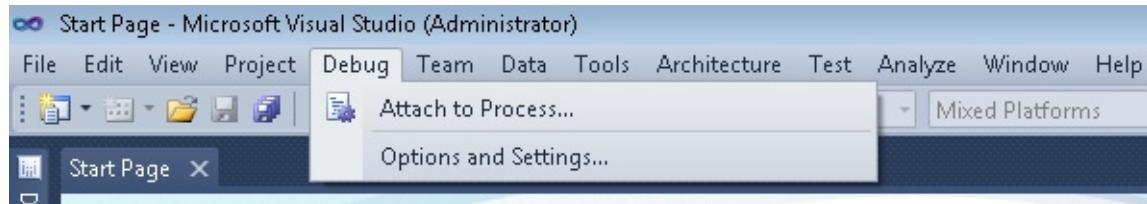
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+P; Ctrl+Shift+R
<b>WINDOWS</b>	Alt, T, P; Alt, D, P, Enter
<b>MENU</b>	Tools   Attach to Process; Debug   Attach to Process
<b>COMMAND</b>	Tools.AttachtoProcess; Debug.AttachtoProcess
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0033

I decided to figure out what the difference is between Debug | Attach To Process and Tools | Attach To Process. I'll spare you the suspense: They are the same.

OK so why are they there? The answer is simple: Prior to Visual Studio 2010, when you didn't have a project open in Visual Studio, it would not show the Debug menu. So the only way you could use Attach To Process was to use the Tools menu:



Beginning in Visual Studio 2010, the Debug menu is available even when a project isn't open:



Essentially, the redundancy is unnecessary in Visual Studio 2010, but it actually served a purpose in prior versions.

## AX.134 The Immediate Window: Running WinDbg and SOS (Son of Strike) Commands

<b>DEFAULT</b>	Ctrl+Alt+I
<b>VISUAL BASIC 6</b>	Ctrl+Alt+I; Ctrl+G
<b>VISUAL C# 2005</b>	Ctrl+Alt+I; Ctrl+D, Ctrl+I; Ctrl+D, I

<b>VISUAL C++ 2</b>	Ctrl+Alt+I
<b>VISUAL C++ 6</b>	Ctrl+Alt+I
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+I
<b>WINDOWS</b>	Alt,D, W, I
<b>MENU</b>	Debug   Windows   Immediate
<b>COMMAND</b>	Debug.Immediate
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0097

It would take a long time to go into detail about WinDbg and SOS (Son of Strike), so I will avoid that here. I want to give you a quick view into how SOS works in the Immediate Window. If you want to get hard-core with debugging, the absolute best places to learn are these two blogs:

- John Robbins, at Wintellect:  
<http://www.wintellect.com/CS/blogs/jrobbins/default.aspx>
- Tess Ferrandez, an ASP.NET Escalation Engineer at Microsoft:  
<http://blogs.msdn.com/b/tess>

## Loading SOS in the Immediate Window

With that said, let's take a look at what it takes to get SOS going in the Immediate Window when using 32-bit and 64-bit architectures. As we go along, I also want to show you the messages you commonly encounter when trying to set this up. First, open the Immediate Window (Ctrl+Alt+I), type in **.load sos**, and press Enter.

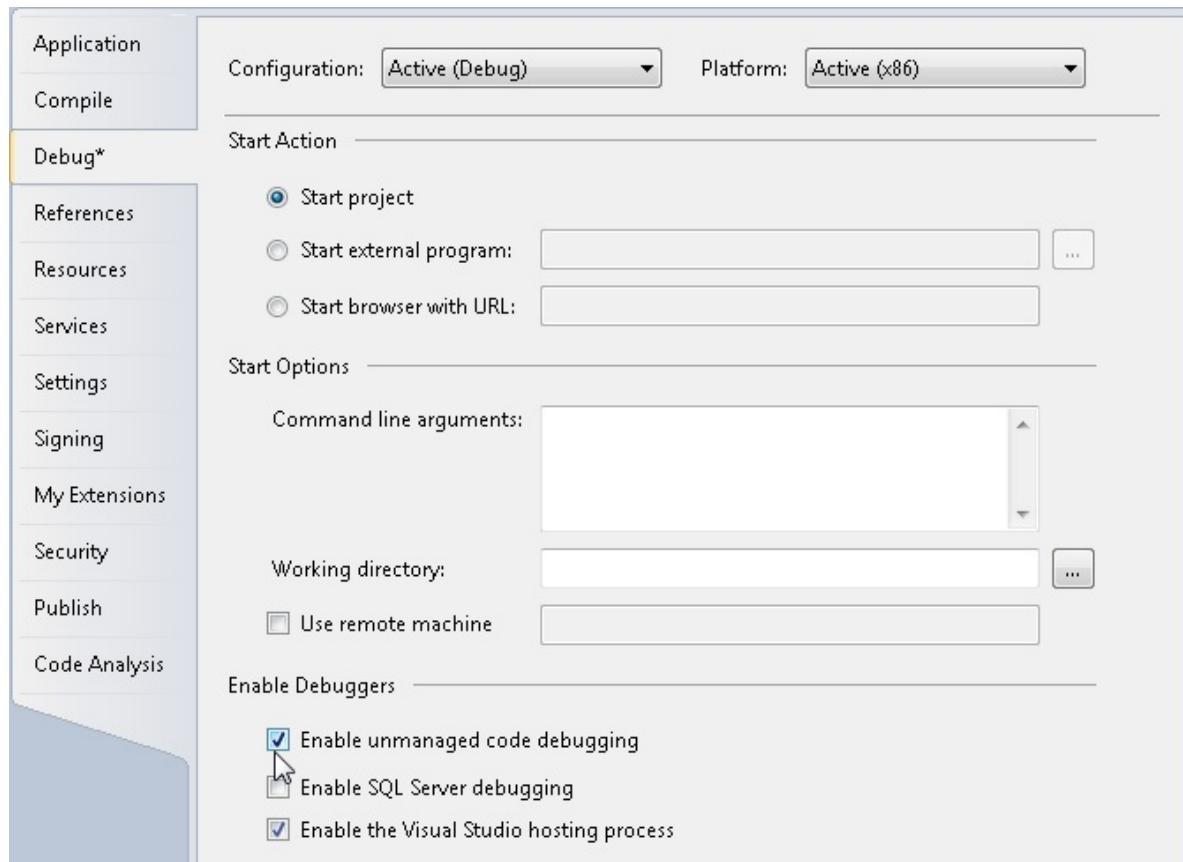
### 32-bit

You most likely get the following message:

“SOS not available while Managed only debugging. To load SOS, enable unmanaged debugging in your project properties.”

To fix this, go to your project properties page and click the Debug tab.

Then select Enable Unmanaged Code Debugging:



Now go back to the Immediate Window, and type **.load sos** again. It might take a few seconds, but eventually you should see the following message:

#### NOTE

Your version might be different based on the CLR being used.

```
Immediate Window
.load sos
extension C:\Windows\Microsoft.NET\Framework\v4.0.30319\sos.dll loaded
```

The screenshot shows the 'Immediate Window' in Visual Studio. The window title is 'Immediate Window'. Inside, the command '.load sos' is typed, followed by the output 'extension C:\Windows\Microsoft.NET\Framework\v4.0.30319\sos.dll loaded'. There is a vertical scroll bar on the right side of the window.

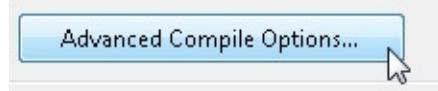
#### 64-bit

You might see the following message:

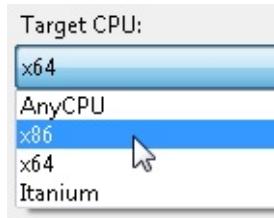
"Error during command: extension  
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\sos.dll could not load  
(error 193)"

This means that you are attempting to debug an x64 (64-bit) application. Visual Studio currently offers no support for interop (managed/unmanaged) debugging on

x64. You can fix this in VB by going to the project properties (Compile tab) and clicking the Advanced Compile Options button:



In C#, you would go to the Build tab of your project properties instead. Then, under Target CPU, choose x86:



## Using SOS

Once you have Son of Strike loaded, there are a variety of WinDbg commands you can leverage. I'll cover some of the more interesting ones in this section.

### Threads and symbols

Type **!threads** to see threading info:

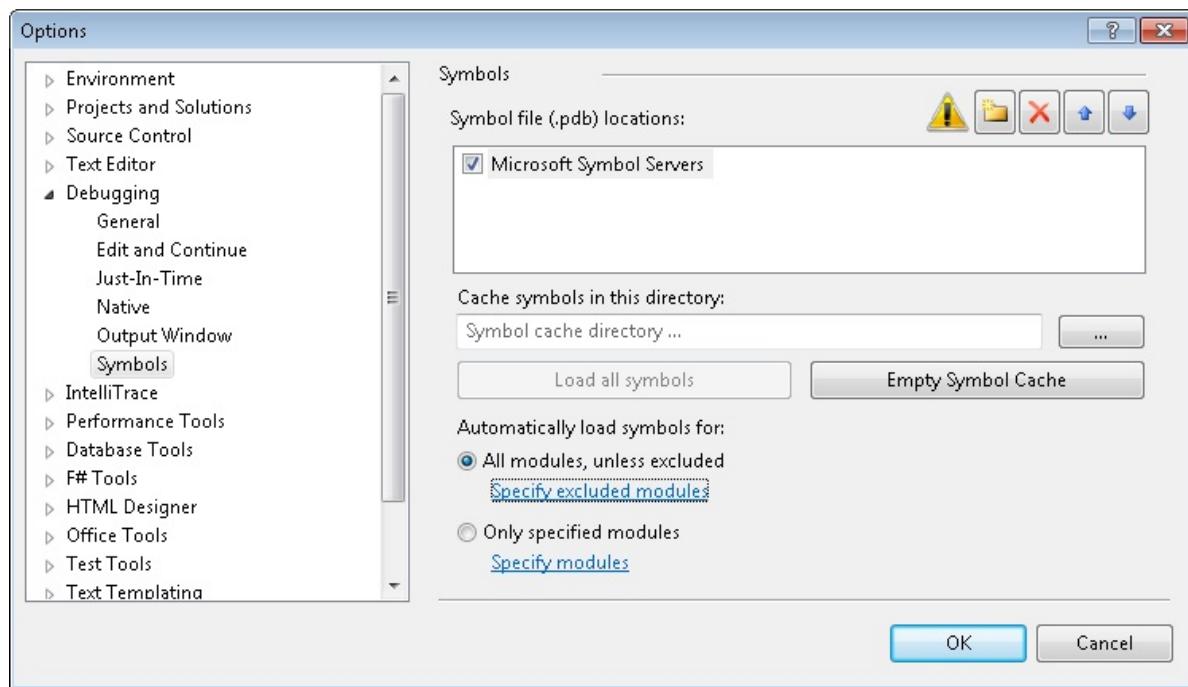
A screenshot of the Immediate Window in WinDbg. The window title is "Immediate Window". The command "!threads" was entered, resulting in the following output:

```
!threads
PDB symbol for clr.dll not loaded
ThreadCount: 2
UnstartedThread: 0
BackgroundThread: 1
PendingThread: 0
DeadThread: 0
Hosted Runtime: no
          PreEmptive   GC Alloc
ID  OSID ThreadOBJ    State GC      Context
10212  1 27e4 006c36d8  86028 Enabled  0260be84:0260c004
8288  2 2060 0068cdc0  b228 Enabled  00000000:00000000
```

Do you see the error that says “PDB symbol for clr.dll not loaded”? This is a common error that is trying to tell you that you need to get symbols. The easiest way to do this is to go to Tools | Options | Debugging | Symbols and select the Microsoft Symbol Servers check box in the Symbol File (.pdb) Locations area:

#### WARNING

There is definitely a performance hit when loading symbols, so be prepared to have some delays as you debug.



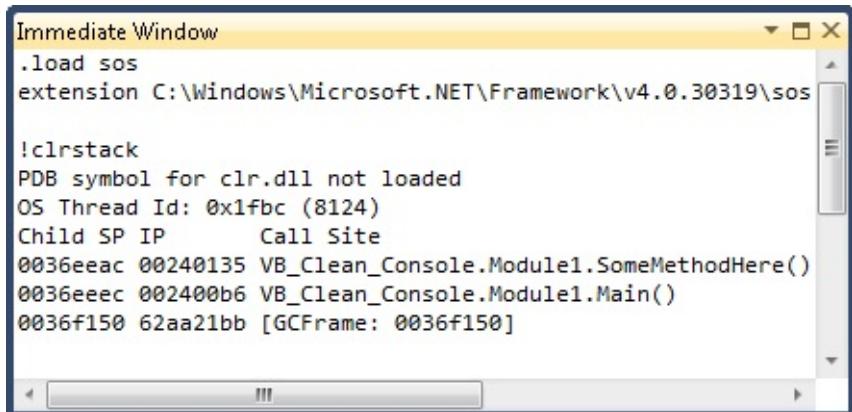
## Dump the managed heap

You can dump the managed heap by typing **!dumpheap**. Just watch out—this is pretty verbose output by default:

Address	MT	Size
02601000	0068c5f8	12 Free
0260100c	0068c5f8	12 Free
02601018	0068c5f8	12 Free
02601024	579ffb8c	84
02601078	579ffcd0	84
026010cc	579ffd1c	84
02601120	579ffd68	84
02601174	579ffdb4	84
026011c8	579ffdb4	84
0260121c	579ff568	12
02601228	579ff92c	16
02601238	57a01a98	28

## Current thread call stack

If you want to display the call stack for the current thread, you can use **!clrstack**. The following illustration shows a sample of the output:



The screenshot shows the Immediate Window in Visual Studio. The window title is "Immediate Window". The content displays the following commands and output:

```
.load sos
extension C:\Windows\Microsoft.NET\Framework\v4.0.30319\sos

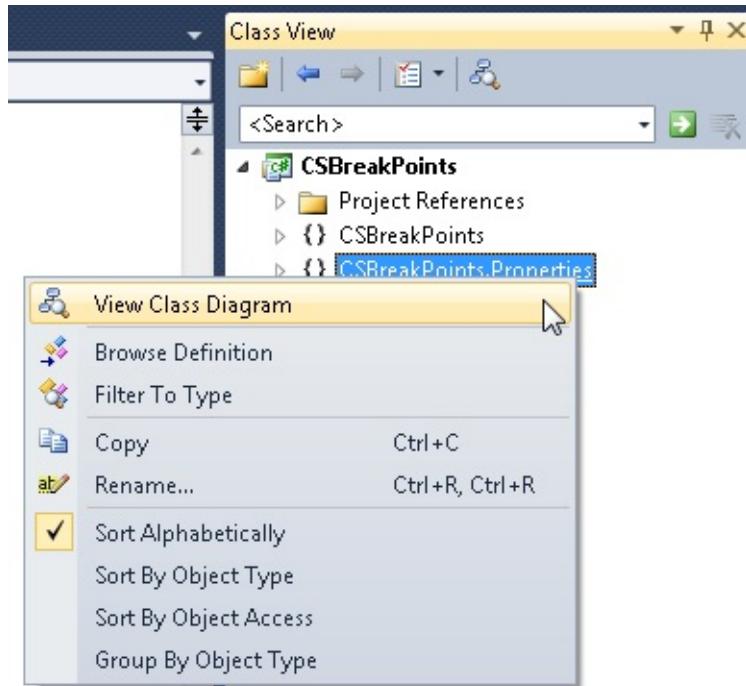
!clrstack
PDB symbol for clr.dll not loaded
OS Thread Id: 0x1fbc (8124)
Child SP IP      Call Site
0036eeac 00240135 VB_Clean_Console.Module1.SomeMethodHere()
0036eeeec 002400b6 VB_Clean_Console.Module1.Main()
0036f150 62aa21bb [GCFrame: 0036f150]
```

These are just a few of the commands you can use. You can get as deep or as shallow into this as you want, but the moral of this story is that you can run WinDbg and SOS commands from the Immediate Window and do some serious debugging from within the IDE.

## AX.135 Creating a Class Diagram from Class View

MENU	[Context Menu]   View Class Diagram
COMMAND	ClassViewContextMenus.ClassViewItem. ViewClassDiagram
VERSIONS	2005, 2008, 2010
LANGUAGES	C#, VB
CODE	vstipTool0112

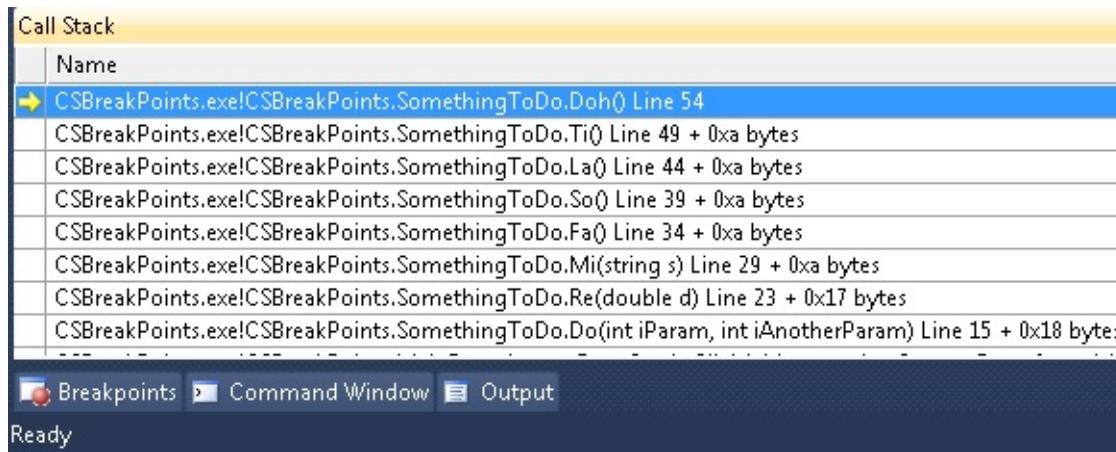
If you like using class diagrams, you can easily create them by using the Class View (Ctrl+Shift+C) window. Just right-click a namespace or class, for example, and choose View Class Diagram:



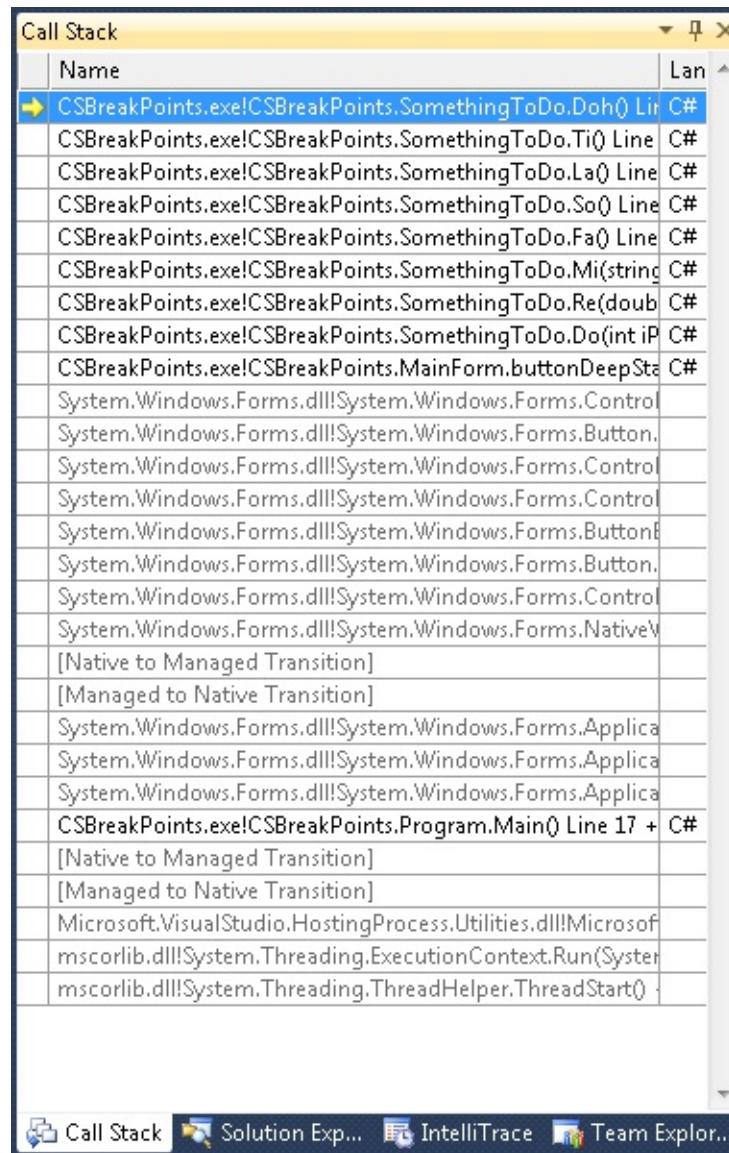
## AX.136 Placing the Call Stack and Call Hierarchy Windows

VERSIONS	2005, 2008, 2010
CODE	vstipTool0115

When you are working with the Call Stack or Call Hierarchy windows, they can sometimes get a little lengthy. Usually you see them docked at the bottom. This is a great feature, but not much fun if you want to look at, say, 20 lines in the stack. You might find it useful to dock the window with Solution Explorer (which by default is docked to the right of your screen). Just drag the tab toward Solution Explorer, and place it with the other tabs:



Now you are all set, and you can see much more information while you work:



Also, recall that this has no impact on your Design Mode experience because the window layouts are different. (See vstipEnv0052, [03.09 Window Layouts: Design, Debug, and Full Screen](#), page 91)

## AX.137 Delete All Breakpoints

<b>DEFAULT</b>	Ctrl+Shift+F9
<b>VISUAL BASIC 6</b>	Ctrl+Shift+F9
<b>VISUAL C# 2005</b>	Ctrl+Shift+F9
<b>VISUAL C++ 2</b>	[no shortcut]
<b>VISUAL C++ 6</b>	Ctrl+Shift+F9
<b>VISUAL STUDIO 6</b>	Ctrl+Shift+F9
<b>WINDOWS</b>	Alt,D, D
<b>MENU</b>	Debug   Delete All Breakpoints
<b>COMMAND</b>	Debug.DeleteAllBreakpoints
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0025

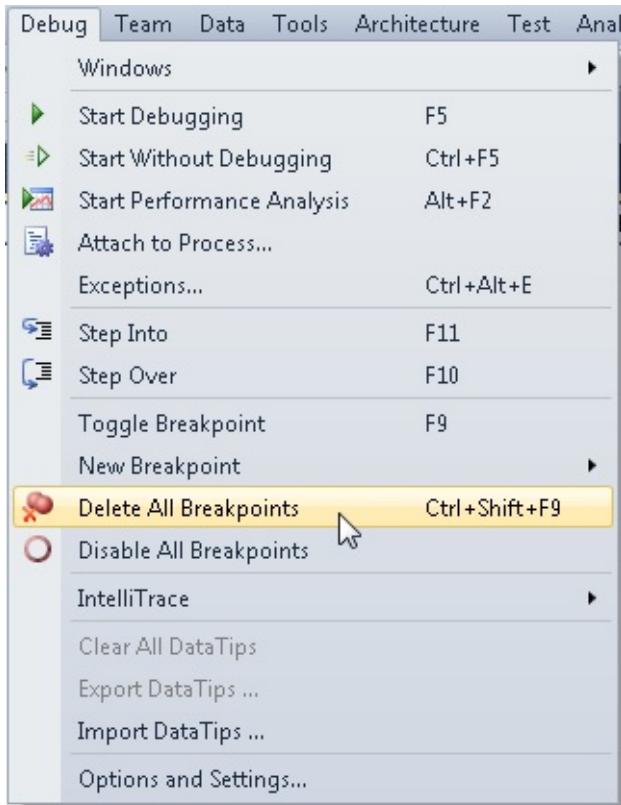
You can delete all your breakpoints at once. You have a couple of options for doing this.

### WARNING

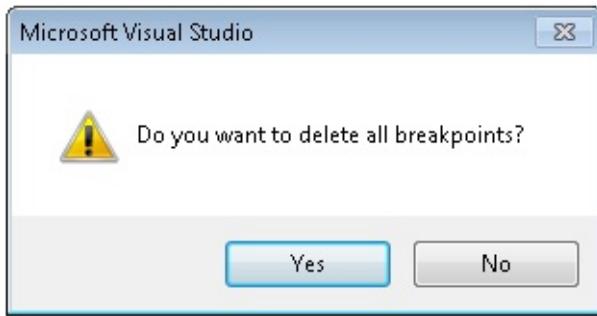
If you want them back, make sure that you export your breakpoints before you use either of these options. See vstipDebug0003, [07.26 Import and Export Breakpoints](#) on page 329, for information about how to back up your breakpoints.

## Debug | Delete All Breakpoints; Ctrl+Shift+F9

If you use the Debug menu or Ctrl+Shift+F9, the behavior hasn't changed from previous versions: All breakpoints get deleted:

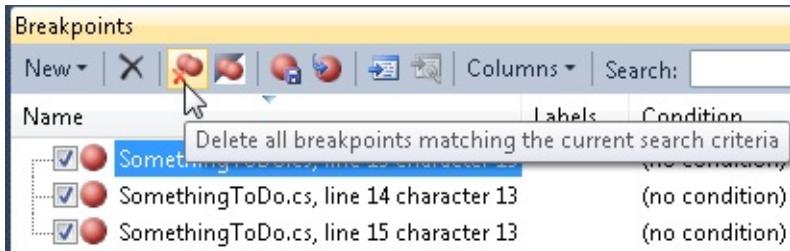


You will get the following dialog box to verify that you want to delete *all* the breakpoints:

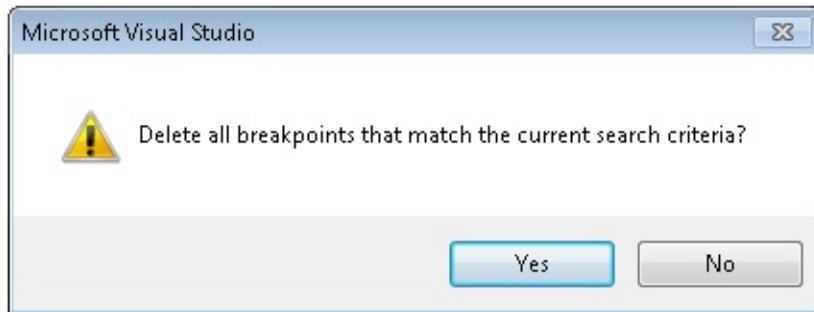


## Breakpoints Window

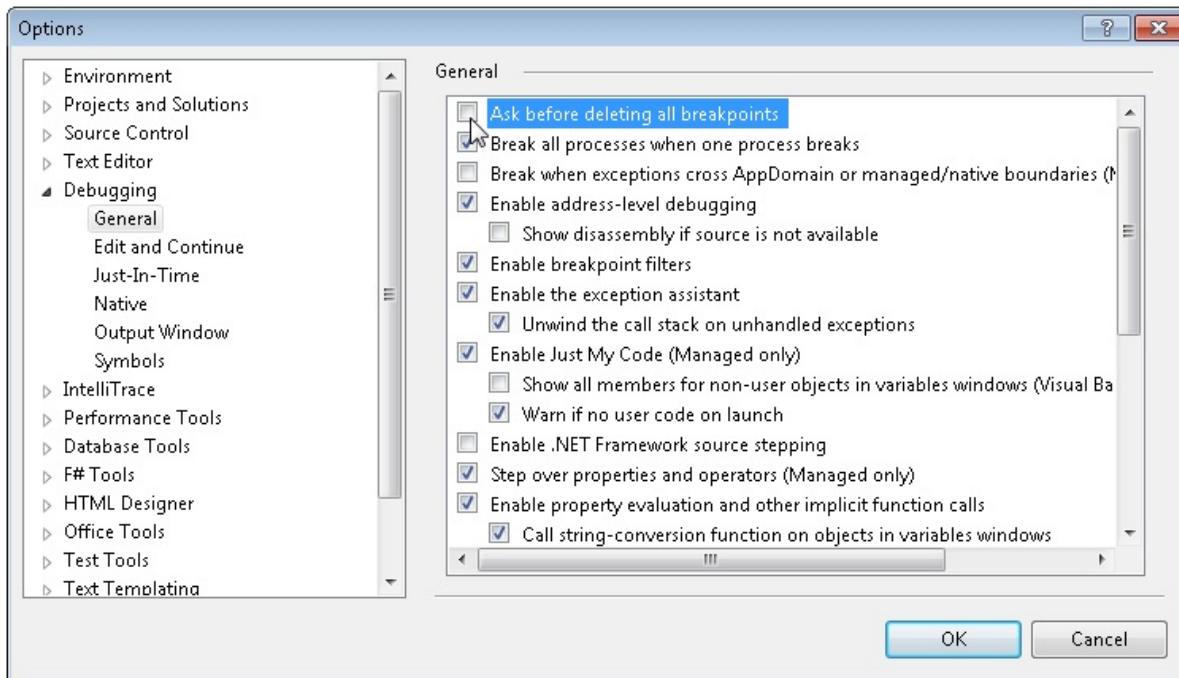
In Visual Studio 2010, you can delete all breakpoints that match the current search criteria. Only those breakpoints that are currently visible in the Breakpoints window are deleted. This is a powerful concept that allows you to remove unwanted breakpoints in bulk that are no longer needed, but still keep the ones you will use later:



You get the following dialog box to verify that you want to delete the breakpoints. Notice the additional text concerning the current search criteria:



You will always get a dialog box to verify that you want to delete all the breakpoints. You can turn this off by going to Tools | Options | Debugging | General and clearing the Ask Before Deleting All Breakpoints check box:



### WARNING

Just because you can turn it off doesn't mean you should turn it off. I don't recommend it.

## AX.138 Make Object ID

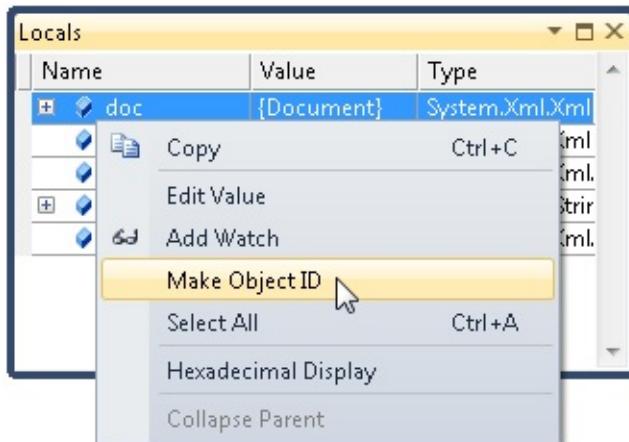
<b>COMMAND</b>	DebuggerContextMenu.AutosWindow.MakeObjectID
<b>VERSIONS</b>	2008, 2010
<b>LANGUAGES</b>	C#, VB
<b>CODE</b>	vstipDebug0015

### NOTE

The idea for this tip came originally from John Robbins at Wintellect (<http://www.wintellect.com>).

Ever want to track an object even if it is out of scope? How about seeing whether an object has been garbage collected? Well, you can do it with Object IDs. Just follow these steps:

1. Set a breakpoint in your code where you can get to an object variable that is in scope.
2. Run your code and let it stop at the breakpoint.
3. In your Locals or Autos Window, right-click the object variable and choose Make Object ID from the context menu:



You should now see something new in the Value column:

Locals			
Name	Value	Type	
+ doc	{Document}{1#}	System.Xml.	
+ myElement	null	System.Xml.	
+ attrColl	null	System.Xml.	
+ sb	null	System.Text.	
+ attr	null	System.Xml.	

That new value is the Object ID that was generated. Let's see how it works.

To experiment for this experiment, type both the object variable and the new Object ID in your Watch window:

Watch 1			
Name	Value	Type	
+ doc	{Document}{1#}	System.Xml.Xml	
+ 1#	{Document}{1#}	object [System	

Now if you go to another method where the object variable (in this case "doc") is out of scope, you get the following result:

Watch 1			
Name	Value	Type	
! doc	The name 'doc' does not exist.		
+ 1#	{Document}{1#}	object [System	

Notice that the variable name "doc" is out of scope, but I can still track the Object ID that shows the location in memory that object variable pointed to. This is very handy for looking at objects for full lifecycle.

Also, you can do some interesting things. For example, you can see which generation the memory space is currently in for purposes of garbage collection:

Watch 1			
Name	Value	Type	
! doc	The name 'doc' does not exist.		
+ 1#	{Document}{1#}	object [System	
GC.GetGeneration(1#)	0	int	

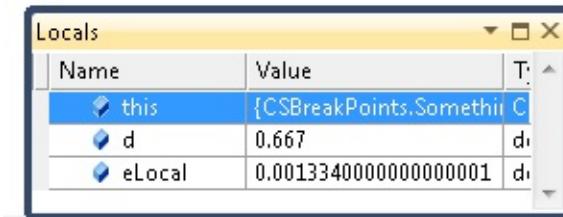
If you want to learn more about Object IDs, see the excellent blog post at <http://blogs.msdn.com/b/jimgries/archive/2005/11/16/493431.aspx>.

## AX.139 Change Values from the Locals Window

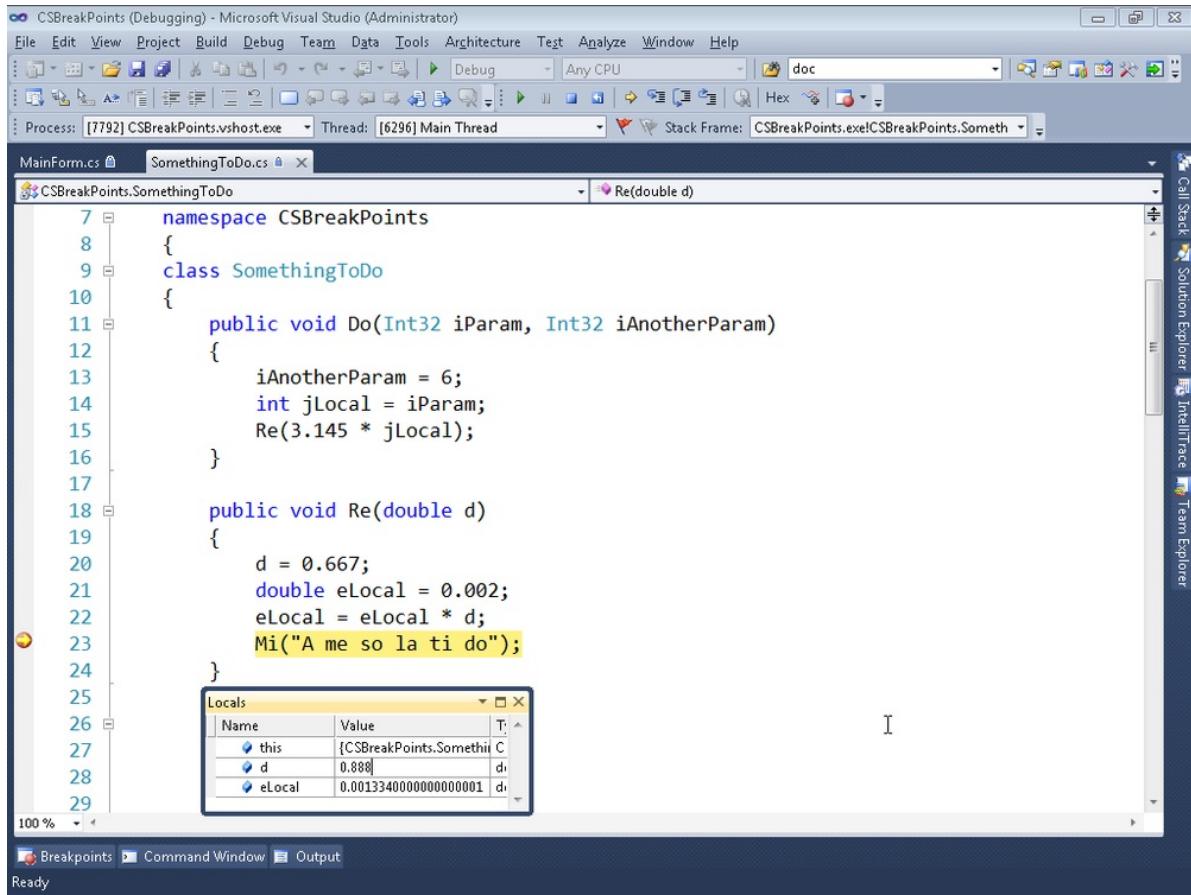
<b>DEFAULT</b>	Ctrl+Alt+V, L
<b>VISUAL BASIC 6</b>	Ctrl+Alt+V, L
<b>VISUAL C# 2005</b>	Ctrl+Alt+V, L; Ctrl+D, Ctrl+L; Ctrl+D, L
<b>VISUAL C++ 2</b>	Ctrl+Alt+V, L; Alt+3
<b>VISUAL C++ 6</b>	Ctrl+Alt+V, L; Alt+4
<b>VISUAL STUDIO 6</b>	Ctrl+Alt+V, L; Ctrl+Alt+L
<b>WINDOWS</b>	Alt,D, W, L
<b>MENU</b>	Debug   Windows   Locals
<b>COMMAND</b>	Debug.Locals
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipTool0102

You can use the Locals window to change values while debugging in break mode. Just find any variable you want to change in your Locals window:

```
public void Re(double d)
{
    d = 0.667;
    double eLocal = 0.002;
    eLocal = eLocal * d;
    Mi("A me so la ti do");
}
```



In this case, let's change "d" to another value:



The changed value turns red afterward to indicate that it has been modified:

Locals	
Name	Value
this	{CSBreakPoints.SomethingToDo}
d	0.888
eLocal	0.001334000000000001

## AX.140 Debug Executable Without Using Attach to Process

<b>DEFAULT</b>	Ctrl+Shift+O
<b>VISUAL BASIC 6</b>	Ctrl+Shift+O; Ctrl+O
<b>VISUAL C# 2005</b>	Ctrl+Shift+O
<b>VISUAL C++ 2</b>	Ctrl+Shift+O
<b>VISUAL C++ 6</b>	Ctrl+Shift+O

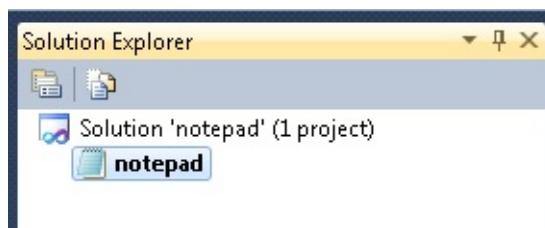
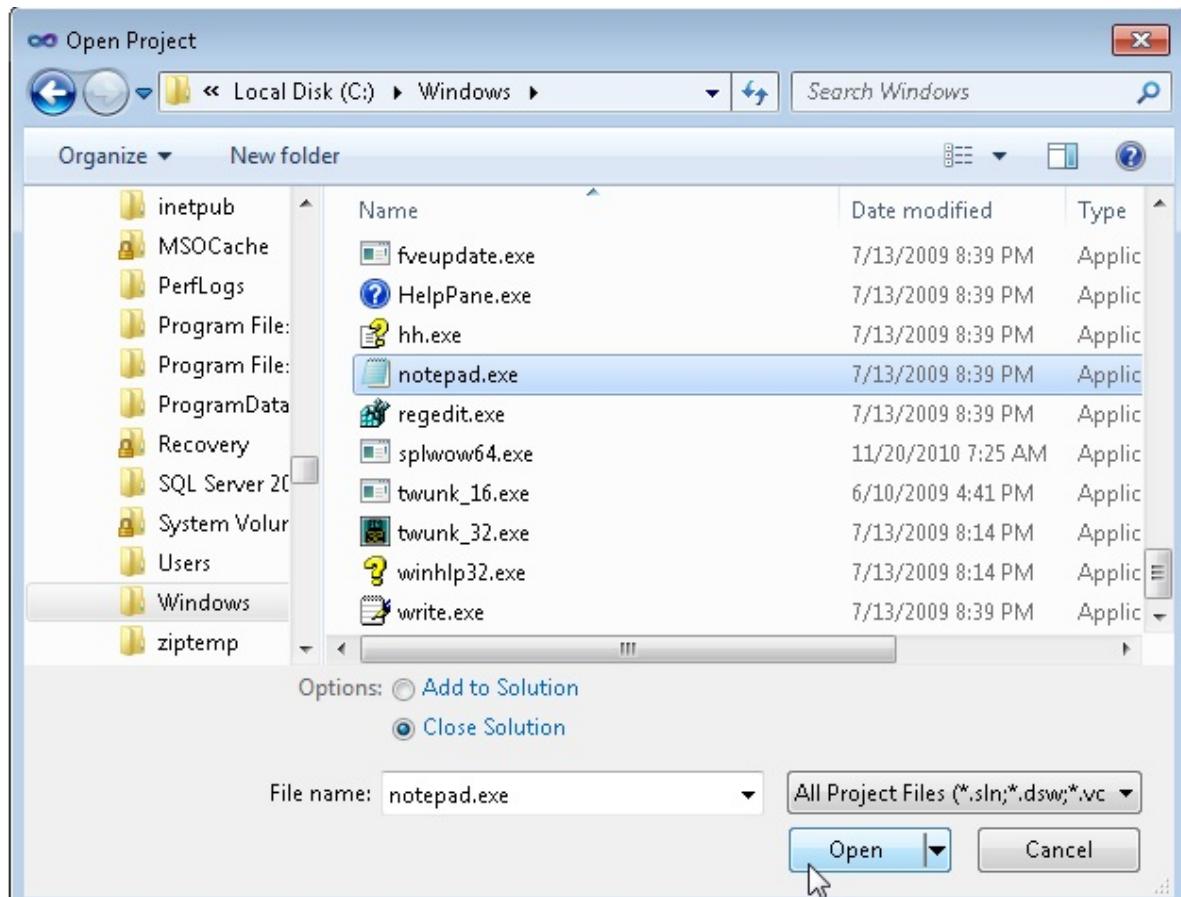
<b>VISUAL STUDIO 6</b>	Ctrl+O
<b>WINDOWS</b>	Alt,F, O, P; Alt,F, D, N
<b>MENU</b>	File   Open Project/Solution; File   Add   New Project
<b>COMMAND</b>	File.OpenProject; File.AddNewProject
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0034

#### **NOTE**

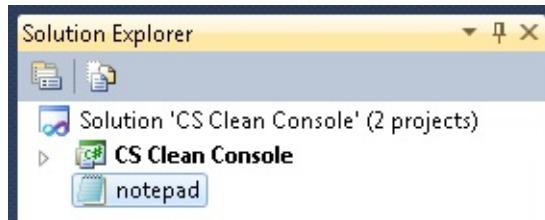
You might need to start Visual Studio with administrator rights before you can use this tip. You will get a warning message that allows you to elevate privileges if this is the case.

You probably already know about the Attach To Process menu items on the Debug and Tools menus, but what if, for example, the process fails before you can attach to it? Maybe it fails on startup, or it runs too fast for you to catch it. Did you know you can create a solution for executables?

It's easy to do. Just find the executable you want to create a solution for by going to File | Open Project/Solution:



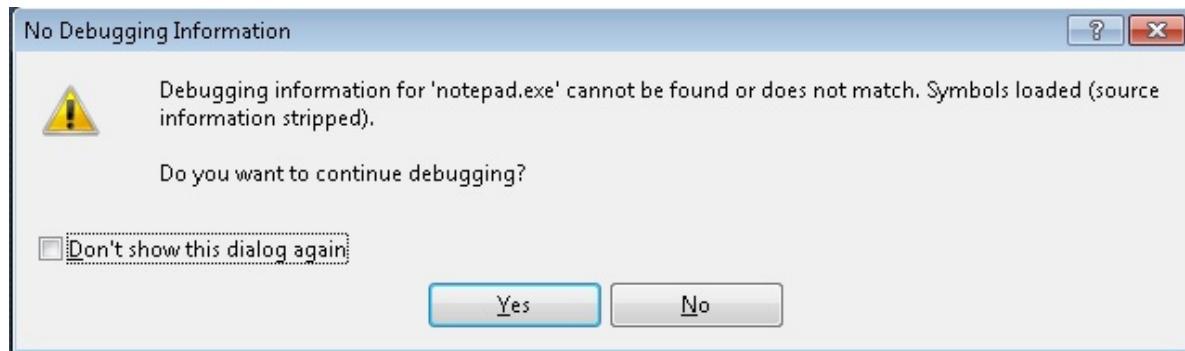
Or, if you have a solution open already, go to File | Add | Existing Project:



Now you can run the executable just like any other project by pressing F5. If you have multiple projects, make sure to set it as the startup.

When you are debugging an executable without the source code, the available debugging features are limited, whether you attach to a running executable or add the executable to a Visual Studio solution.

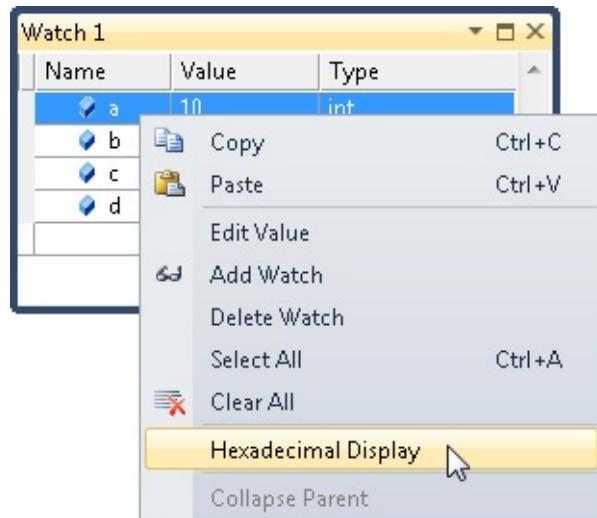
If the executable was built without debug information in a compatible format, available features are further limited. If you have the source code, the best approach is to import the source code into Visual Studio and create a debug build of the executable in Visual Studio:



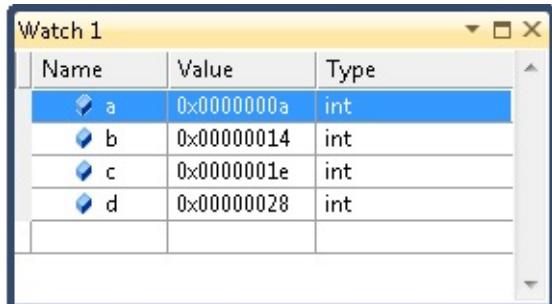
## AX.141 The Watch Window: Hexadecimal Display

MENU	[Context Menu]   Hexadecimal Display
COMMAND	Debug.HexadecimalDisplay
VERSIONS	2005, 2008, 2010
CODE	vstipTool0110

All the variable windows (Locals, Autos, QuickWatch, and Watch) support showing the hexadecimal display for values. These values are particularly useful when you are dealing with data that requires any hex input for values. In any variable window, you can right-click anywhere and choose Hexadecimal Display:



You now have hex values displayed for the values in all the variable windows:

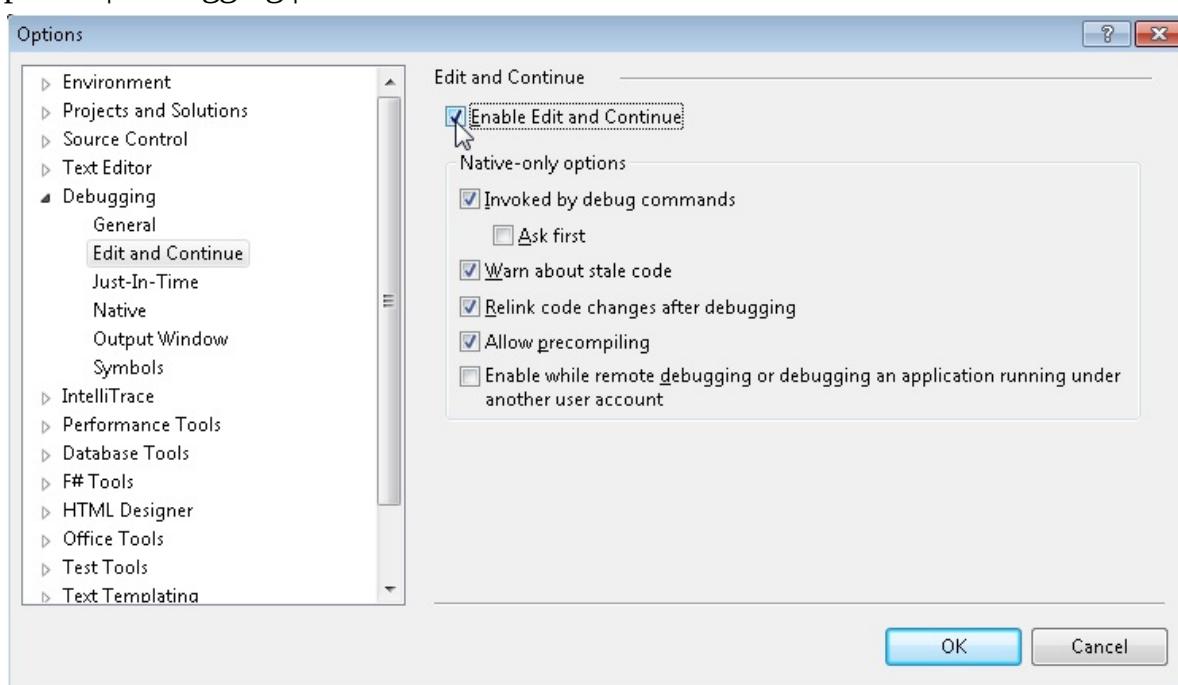


You can repeat this action to turn the hex display off.

## AX.142 Edit And Continue

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipDebug0038

Did you know you can edit your code while you are debugging? You can do it with the Edit And Continue (EnC) feature. First, you can find this option under Tools | Options | Debugging | Edit And Continue:



However, in many language-specific scenarios, you can't use this feature. For more detailed information, see the topic "Edit and Continue" at [http://msdn.microsoft.com/en-us/library/bcew296c\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/bcew296c(v=VS.100).aspx).

The preceding caveat notwithstanding, this is an interesting feature that allows you to edit code while you are debugging and to continue execution without having to do a full recompile of the code.

## Disclaimer

I would be remiss if I didn't mention that there are some people who don't think using Edit And Continue is a good idea. John Robbins, whose opinion I respect a great deal, is one of those people, and he makes some compelling arguments why you might not want to use this feature all the time. You can find John's post at <http://www.wintellect.com/CS/blogs/jrobbins/archive/2004/10/17/c-edit-and-continue-announced.aspx>.

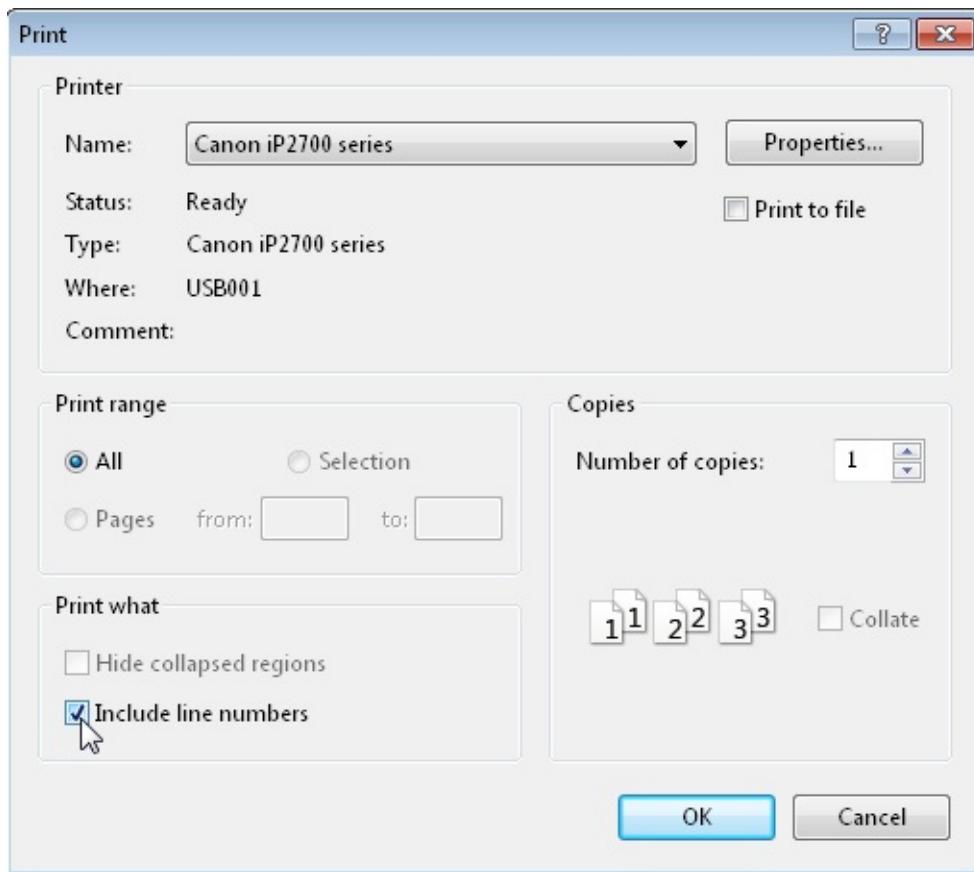
Interestingly, you can find Jeff Atwood's rebuttal to John's argument at <http://www.coding-horror.com/blog/2006/02/revisiting-edit-and-continue.html>.

The bottom line: Make your own informed decision about using Edit And Continue, but even if you do use it, be fully aware of the implications of changes you make when using it.

## AX.143 Print with Line Numbers

<b>DEFAULT</b>	Ctrl+P
<b>VISUAL BASIC 6</b>	Ctrl+P
<b>VISUAL C# 2005</b>	Ctrl+P
<b>VISUAL C++ 2</b>	Ctrl+P; Ctrl+Shift+F12; Ctrl+Shift+Alt+F2
<b>VISUAL C++ 6</b>	Ctrl+P
<b>VISUAL STUDIO 6</b>	Ctrl+P
<b>WINDOWS</b>	Alt,F, P
<b>MENU</b>	File   Print
<b>COMMAND</b>	File.Print
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0006

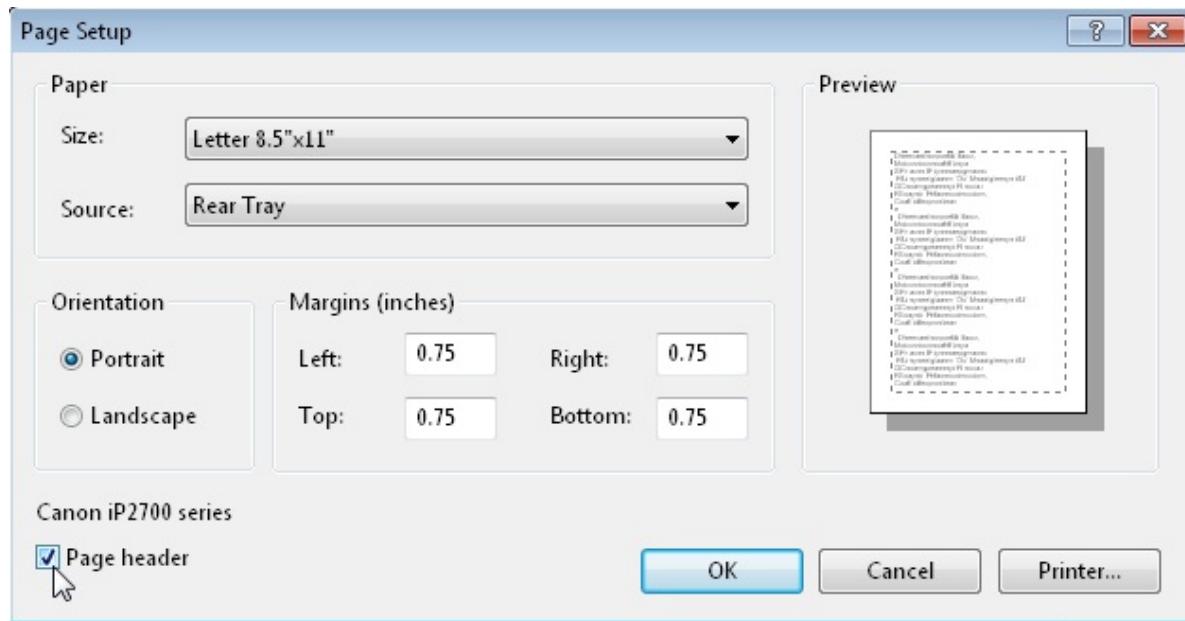
Want to print your line numbers with your code? Don't worry! You can do it by just checking the Include Line Numbers option in the Print dialog box:



## AX.144 Printing the File Path in the Page Header

<b>WINDOWS</b>	Alt,F, U
<b>MENU</b>	File   Page Setup
<b>COMMAND</b>	File.PageSetup
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0008

This feature is on by default in Visual Studio 2010, but just in case you accidentally turn it off or maybe you don't want the file path in the page header, just go to File | Page Setup and notice the Page Header check box:

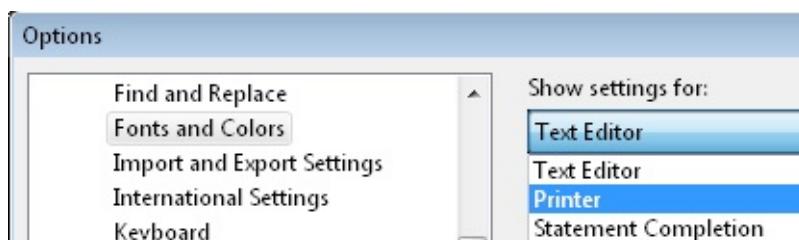


The Page Header setting toggles the printing of the file path in the page header.

## AX.145 Printing in Different Fonts and Colors

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008
<b>CODE</b>	vstipEnv0007

Ever change your fonts or colors in the editor only to be frustrated because the fonts and colors do not print correctly or as expected? Just go to Tools | Options | Environment | Fonts And Colors on your menu bar, and then in the Show Settings For drop-down box, select Printer. Now you can change how your printed output looks:



To use your editor colors when you print, just click Use and select Text Editor

Settings:



The color and font settings from the text editor will now be applied to your printed output.

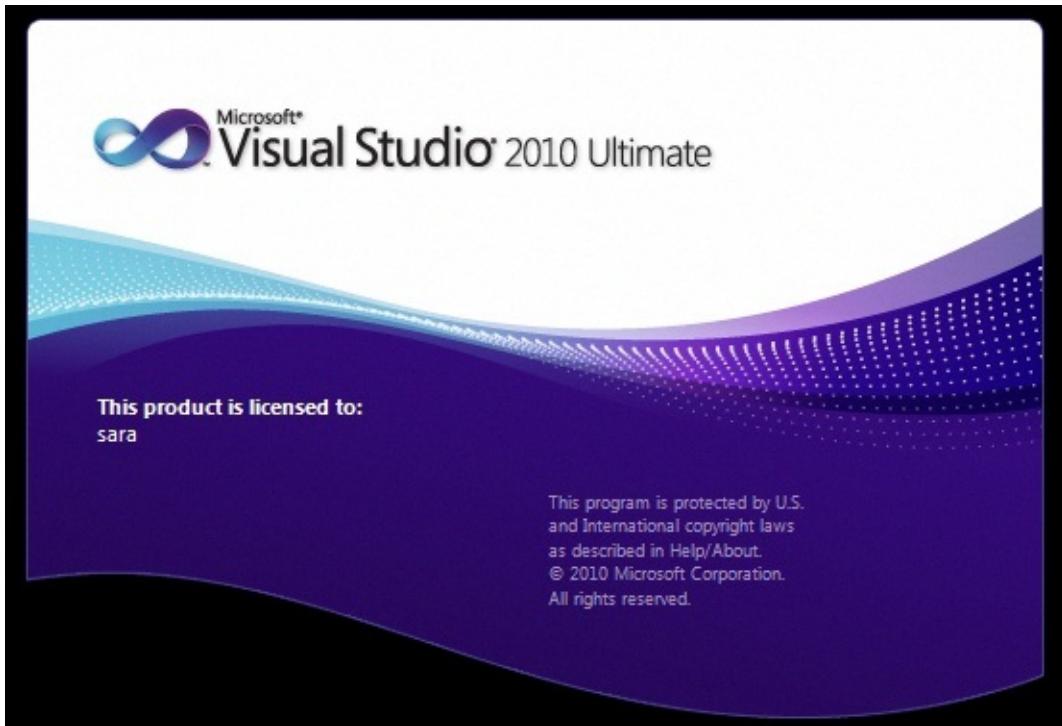
**NOTE**

This is a copy operation, so if you change the text editor settings, you must redo this step to copy the new settings over.

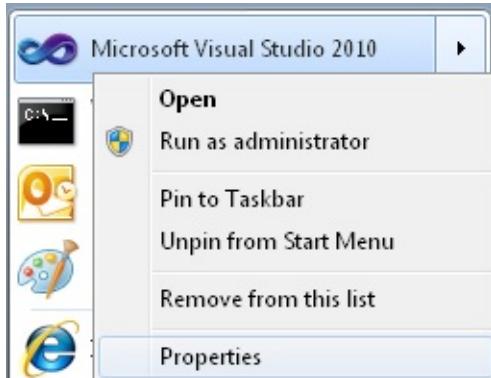
## AX.146 Get Rid of the Splash Screen

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0046

When you start Visual Studio, the splash screen is often the first thing you see:



Did you know that you can suppress it? Just go to the properties of your Visual Studio program icon:



Now click the Shortcut tab:



Add **/nosplash** to the end of the Target area:

Target: `udio 10.0\Common7\IDE\devenv.exe" /nosplash`

Now when you run Visual Studio, you no longer see the splash screen.

## AX.147 Understanding Check Accessibility

<b>WINDOWS</b>	Alt,T, B
<b>MENU</b>	Tools   Check Accessibility; [Context Menu]   Check Accessibility
<b>COMMAND</b>	Tools.CheckAccessibility
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipProj0028

According to the documentation in “Accessibility in Visual Studio and ASP.NET,” at [http://msdn.microsoft.com/en-us/library/ms228004\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/ms228004(v=VS.100).aspx), accessibility is an important consideration in your development work:

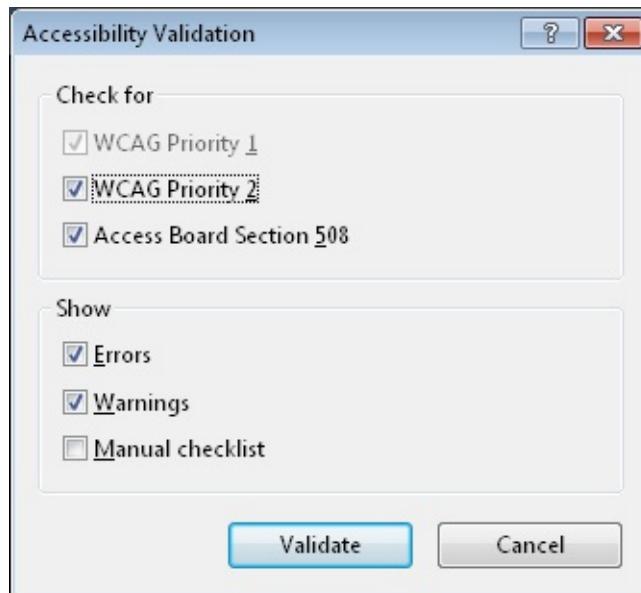
*“Accessibility standards enable you to build Web pages that can be used by people who have disabilities. [You can] configure ASP.NET Web server controls to make sure that they generate accessible HTML.”*

It is always better to make your websites accessible to people with disabilities,

and in some cases, it's required. Did you know that you can easily determine whether a page meets accessibility requirements by right-clicking on any page and choosing Check Accessibility?



This opens the Accessibility Validation dialog box:



Following are descriptions of each option on the Accessibility Validation dialog box.

## Check For

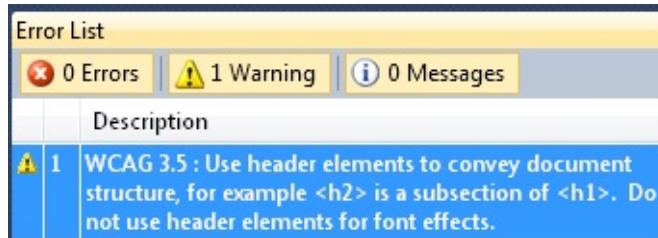
*WCAG Priority 1 & 2*—checks for compliance with Web Content Accessibility Guidelines (<http://www.w3.org/WAI/intro/wcag.php>).

*Access Board Section 508*—checks accessibility by using the standards that were defined by the United States government in Section 508 of the Rehabilitation Act,

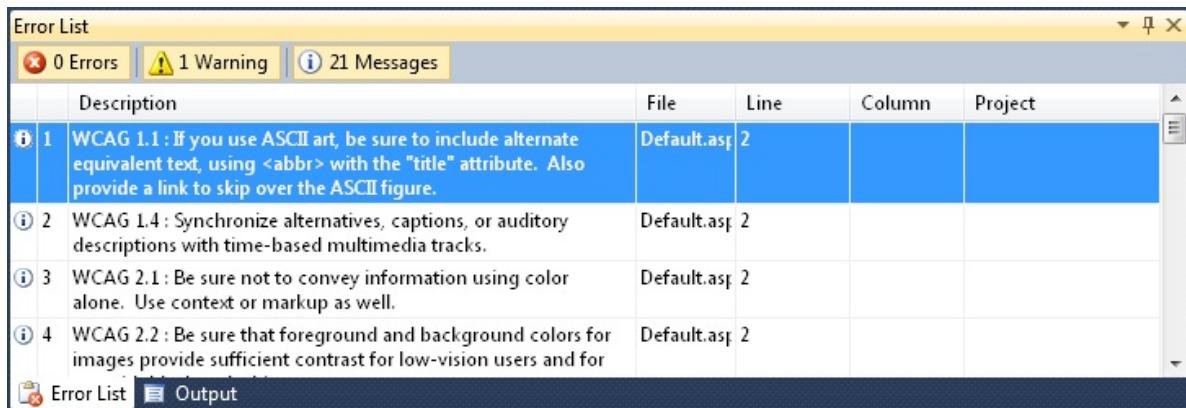
which are based on the WCAG (<http://www.section508.gov>).

## Show

- **Errors and Warnings** Shows relevant items that violate the rules selected in the Check For section:



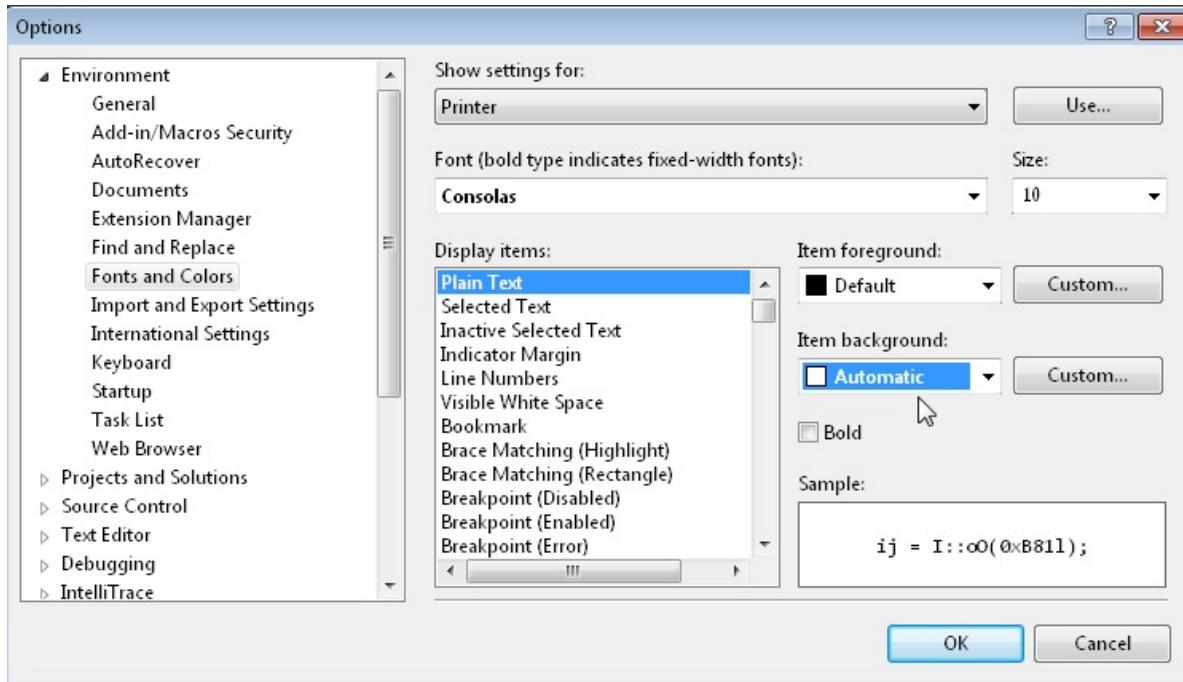
- **Manual Checklist** Generates informational messages that can be used as guides while the errors and warnings are addressed:



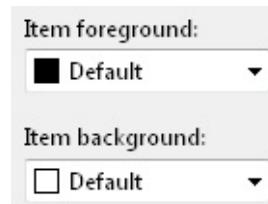
## AX.148 Automatic vs. Default in Fonts and Colors

<b>WINDOWS</b>	Alt,T,O
<b>MENU</b>	Tools   Options
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEnv0009

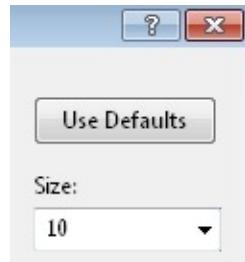
If you've ever wondered what the difference is between the Default and Automatic options in the fonts and colors found at Tools | Options | Environment | Fonts And Colors, see the next illustration:



## Default



The Default setting is pretty straightforward. It uses the default colors set up by Visual Studio for the item selected. The colors are what Visual Studio defines as the default. You might call these the “normal” colors, for want of a better term. The good news is that you can always restore these settings by just clicking Use Defaults in the upper-right corner of the dialog box:

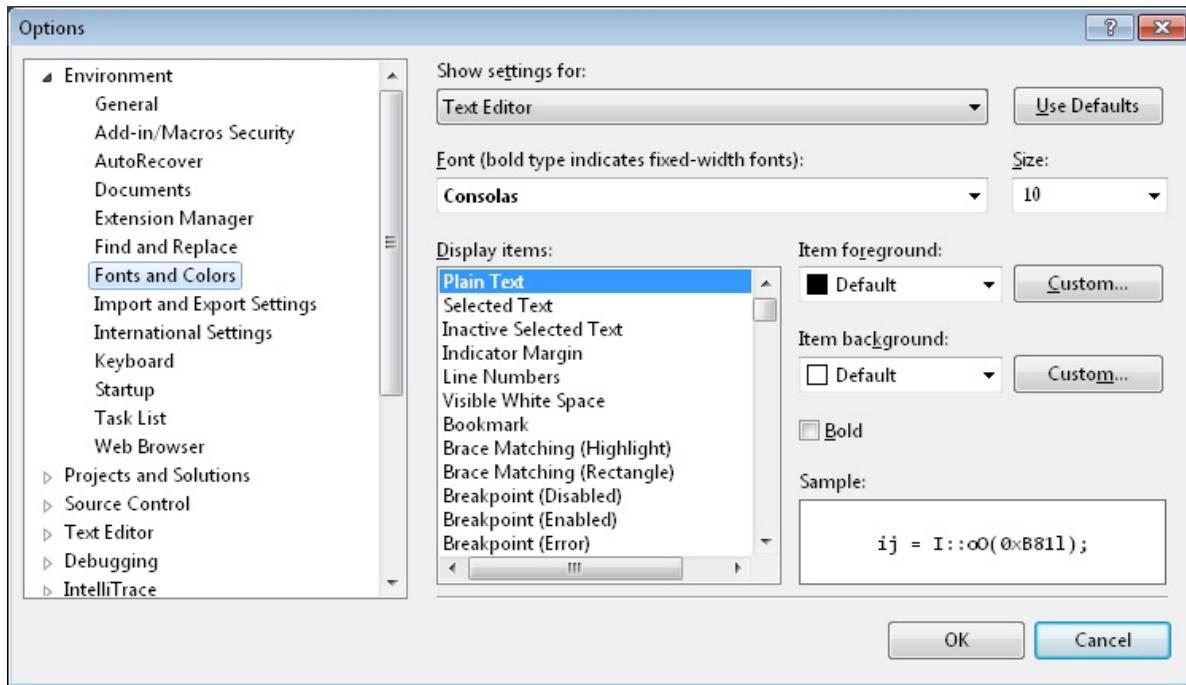


## Automatic

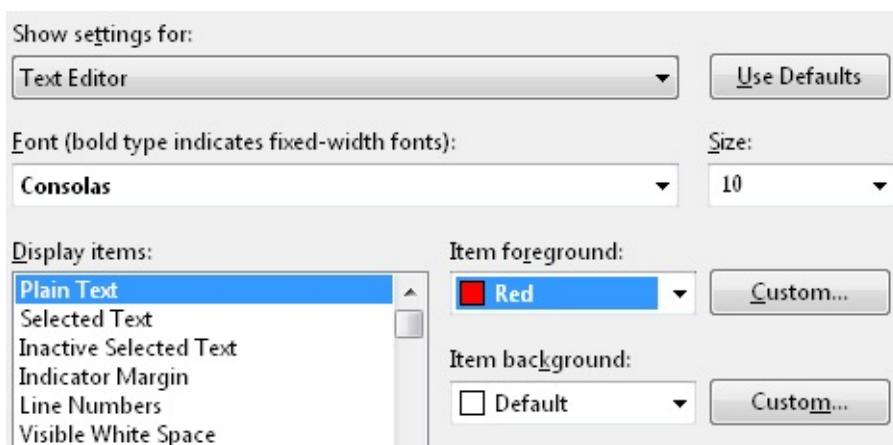
The Automatic setting is a lot more interesting. Here is the definition of Automatic from the documentation (the bold emphasis is mine):

*"Items can inherit the [...] color from other display items such as Plain Text. Using this option, when you change the color of an inherited display item, the color of the related display items also change automatically. For example, if you selected the Automatic value for Compiler Error and later changed the color of Plain Text to Red, Compiler Error would also automatically inherit the color Red."*

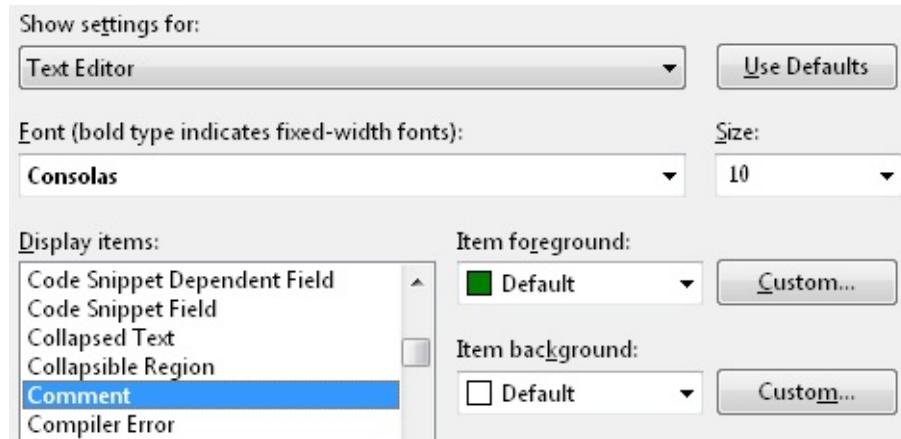
OK, so let's show you what this means. Following are the current settings for Plain Text:



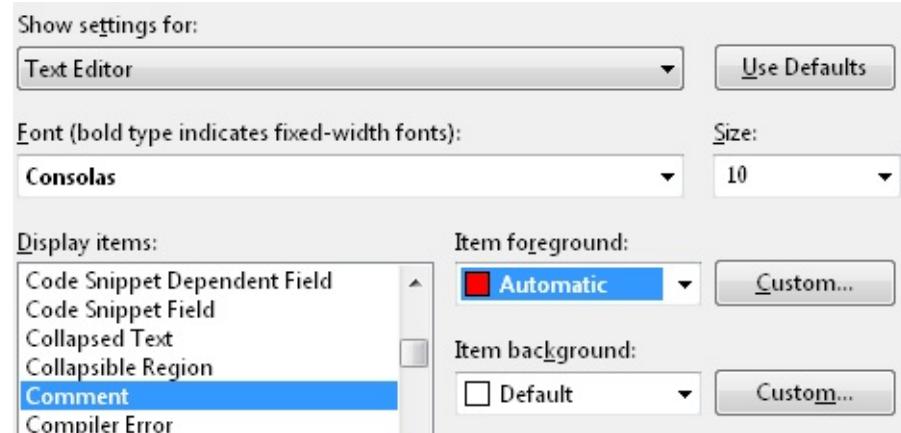
Notice, among other things, that the Item foreground is black. Now let's change the foreground color to red:



Now let's change the display item to Comment and see what the settings are there:



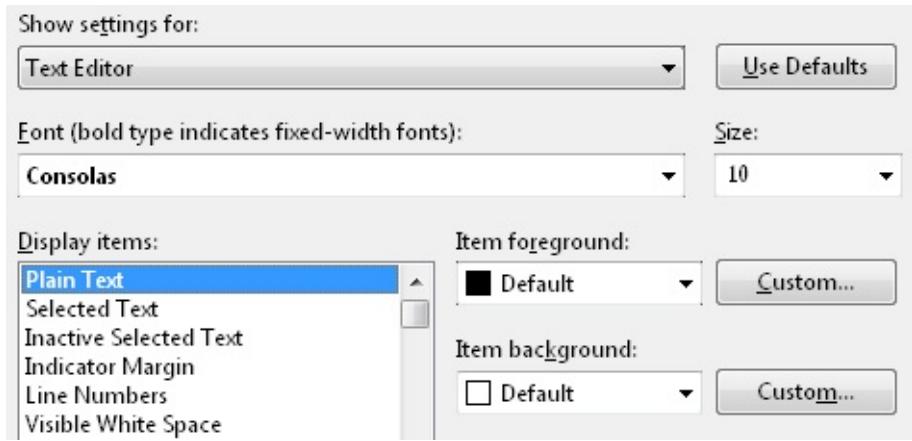
Notice that the Item foreground is green and is set to Default. If we change the Item foreground to Automatic, we get the following:



Notice that Comment inherited the Item Foreground setting from the Plain Text setting we just changed to red. So now we clearly see the relationship between the Plain Text setting and some of the other display items. You need to be aware of the following factors:

- Not all items inherit from plain text.
- Plain text inherits from the Windows System.

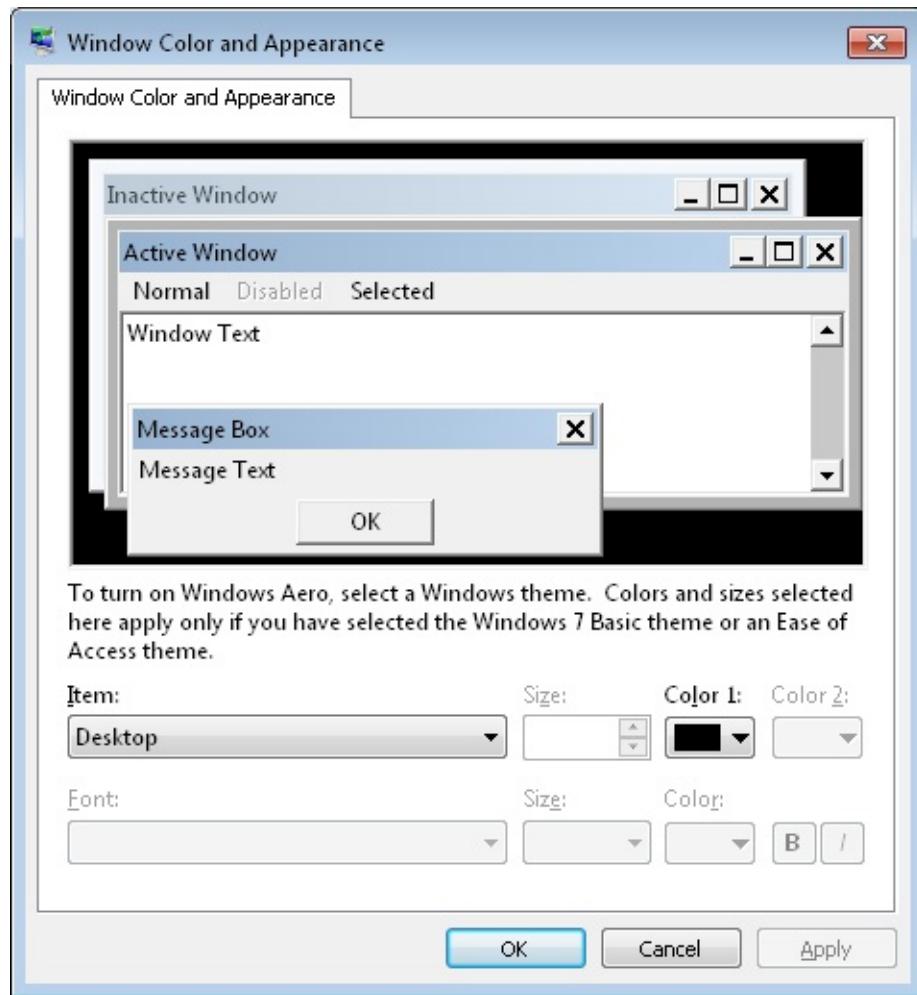
Let's address the second point. With all the colors back to the default settings, let's look at the Plain Text setting again:



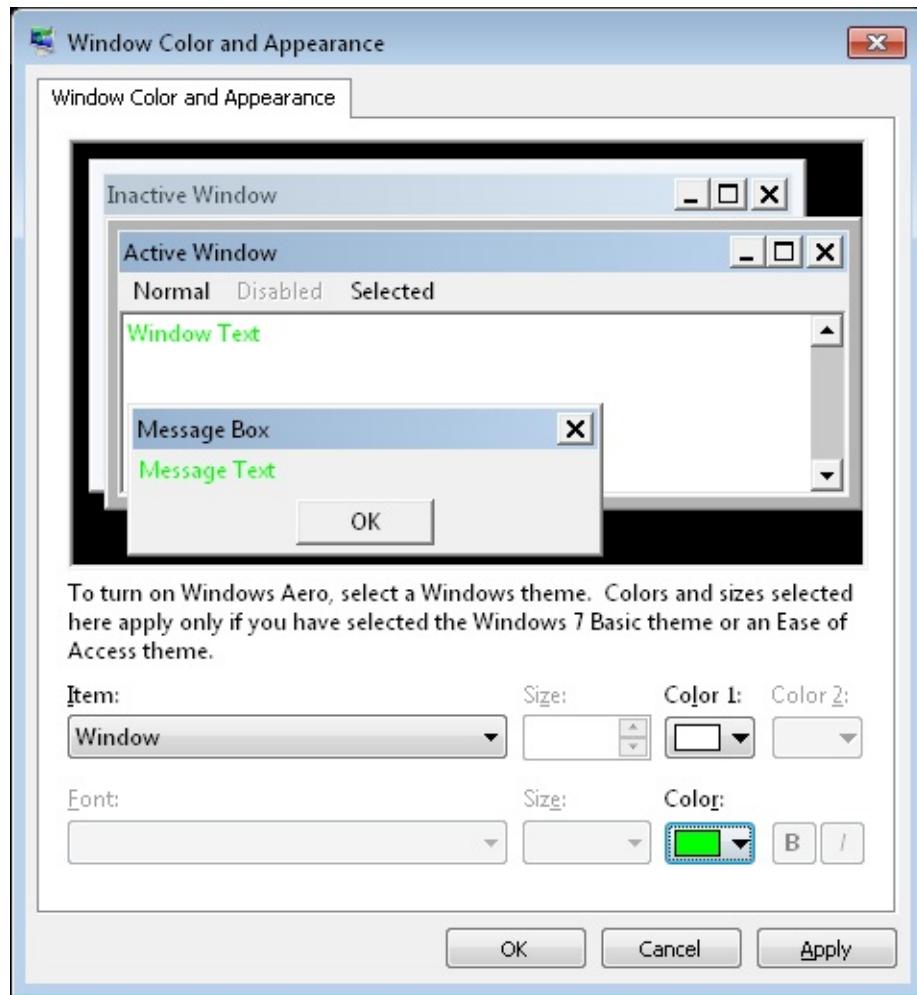
Plain Text has a default Item Foreground setting of black again. Where does the plain text default color come from? Let's see what happens if we change the Windows System colors. In this example, I'm using Windows 7, so your settings might be elsewhere, but all versions of Windows have a similar area. In the case of Windows 7, it's called Change Windows Colors And Metrics, as you can see in the following illustration:



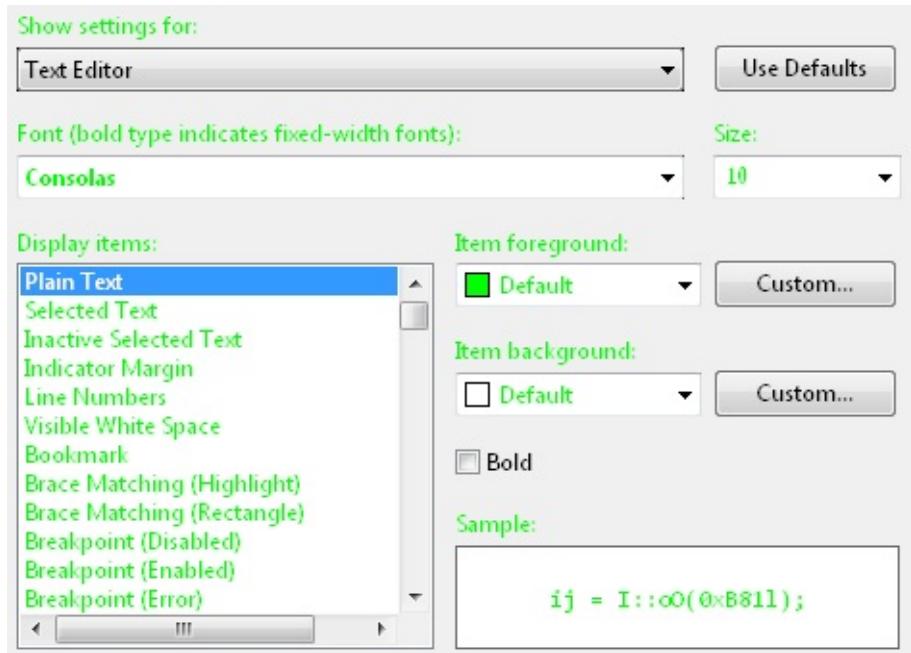
When I click this option, it brings me to the Window Color And Appearance dialog box:



Now I change the Item to Window and the font color to lime green:



I click Apply and then switch back to Visual Studio and notice that, among other things, the Plain Text setting now has a default color of lime green:



So the plain text default color clearly inherits from the Windows System, and now various display items inside Visual Studio can inherit from the Plain Text setting by setting their color to Automatic. Now we finally have a clear picture of the difference between Default and Automatic in the Fonts And Colors dialog box.

At this point, you might want to change all your colors back to what they were before we started this adventure.

## AX.149 Visual Studio Permissions Needed on Windows Vista or Later

VERSIONS	2005, 2008, 2010
CODE	vstipEnv0056

A popular misconception is that you need to have Administrator privileges to use Visual Studio. While this is true in some cases, it isn't true in all of them. So when do you need to run Visual Studio as an administrator and when don't you? This tip offers some guidance.

### Installing Visual Studio (All Versions)

You need Administrator rights to install Visual Studio.

### Running Visual Studio 2005

To run Visual Studio 2005 on Windows Vista or later, you are prompted to run as Administrator when you start the application. This version of Visual Studio requires that you have Administrator rights to use it.

## **Specific Scenarios for Visual Studio 2008/2010**

### **Web/Internet Information Services**

Creating a new local or remote IIS website project—You cannot make changes to the Internet Information Services (IIS) metabase (for example, creating new entries) because it requires administrative privileges. This affects your ability to configure some settings in the Web.config file.

Opening a local or remote IIS website project—You cannot run your website unless you use the ASP.NET Development Server, which is the default web server for filesystem websites. Alternatively, you can set project options to open the browser and point to the website by using IIS.

Debugging a local or remote IIS website project—You cannot attach to a process that is running under the IIS worker process because it requires administrative privileges.

More information can be found here: [http://msdn.microsoft.com/en-us/library/ms178112\(v=VS.90\).aspx](http://msdn.microsoft.com/en-us/library/ms178112(v=VS.90).aspx).

### **Windows Installer deployment**

Windows Installer technology supports software installation on the Windows Vista (or later) operating system. The end user installing applications on Windows Vista should receive prompts only for each component installation that requires elevation, even when the user's computer runs under User Account Control (UAC).

More information can be found here: [http://msdn.microsoft.com/en-us/library/Bb384154\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/Bb384154(v=VS.100).aspx).

### **ClickOnce deployment**

Windows Installer deployment requires administrative permissions and allows only limited user installation; ClickOnce deployment enables non-administrative users to install and grants only those Code Access Security permissions necessary for the application.

More information can be found here: <http://msdn.microsoft.com/en-us/library/t71a733d.aspx>.

### **Code**

Some code requires Administrator access to execute. If possible, alternatives to this code should be pursued. Examples of code operations that require Administrator access are as follows:

- Writing to protected areas of the filesystem, such as the Windows or Program Files directories
- Writing to protected areas of the registry, such as HKEY\_LOCAL\_MACHINE
- Installing assemblies in the Global Assembly Cache (GAC)

Generally, these actions should be limited to application installation programs. This allows users to use administrator status only temporarily.

More information can be found here: [http://msdn.microsoft.com/en-us/library/ms173360\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/ms173360(v=VS.100).aspx).

## Debugging

According to the documentation, “you can debug any applications that you launch within Visual Studio (native and unmanaged) as a non-administrator by becoming part of the Debugging Group. This includes the ability to attach to a running application using the Attach to Process command. However, it is necessary to be part of the Administrator Group in order to debug native or managed applications that were launched by a different user.”

More information can be found here: [http://msdn.microsoft.com/en-us/library/ms173360\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/ms173360(v=VS.100).aspx).

## COM/COM Interop

### Classic COM

When you add a classic COM control, such as an .ocx control, to the toolbox, Visual Studio tries to register the control. You must have administrator credentials to register the control.

Add-ins written by using classic COM must be registered to work in Visual Studio. You must have administrator credentials to register the control.

### COM Interop

When you build managed components and you have selected Register For COM Interop, the managed assemblies must be registered. You must have administrator credentials to register the assemblies.

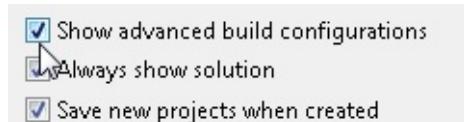
More information can be found here: [http://msdn.microsoft.com/en-us/library/ms165100\(v=VS.90\).aspx](http://msdn.microsoft.com/en-us/library/ms165100(v=VS.90).aspx).

## AX.150 Show Advanced Build Configurations

WINDOWS	Alt,T, O
MENU	Tools   Options   Projects and Solutions   General
COMMAND	Tools.Options
VERSIONS	2005, 2008, 2010
CODE	vstipProj0016

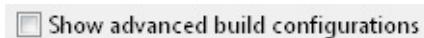
This one is interesting, and the main reason I'm showing it to you is so that you know how to turn this back on if it ever gets turned off.

If you go to Tools | Options | Projects And Solutions | General, you can see many available options. Locate the Show Advanced Build Configurations option:



## Simplified Build Configurations

So what does this option do? Well, when it is turned off, it uses simplified build configurations, which involve several changes:



## Configuration Manager

In simplified builds, the Configuration Manager is not available. It's just disabled (or removed completely) from all areas:



The practical implications of this are that you can't do any custom build configurations, which makes sense with a simplified build configuration option. So when you don't see the Configuration Manager anymore or it's disabled, that is usually a clear sign that you have turned off Show Advanced Build Configurations.

## Debug Build

In a simplified build scenario, each time you press F5 or go to Debug | Start Debugging, a debug build of your application is created. This is the expected

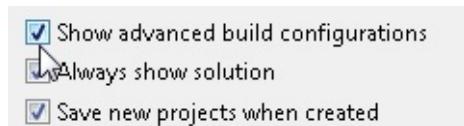
behavior.

## Release Build

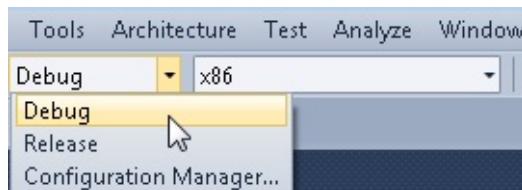
The real shocker is that you have no obvious way to create a release build. With simplified builds, you create a release build by going to Build | Build Solution (Ctrl+Shift+B).

## Advanced Build Configurations

I could do an entire series on advanced builds, but I'll just keep it to the basics for now. Essentially, you first select Show Advanced Build Configurations:



This option gives you access to the Configuration Manager, so you can actively switch between build types without having to remember some arcane steps to do it:



As an added bonus, it gives you the ability to make custom builds with your own special configuration options specified for the build.

In short, you have many good reasons to show the advanced build configurations options and few good reasons to turn it off.

## AX.151 Emacs Emulation

<b>WINDOWS</b>	Alt,T, O
<b>MENU</b>	Tools   Options   Environment   Keyboard
<b>COMMAND</b>	Tools.Options
<b>VERSIONS</b>	2005, 2008, 2010
<b>CODE</b>	vstipEdit0079

For those not familiar with the term, according to Wikipedia (<http://en.wikipedia.org/wiki/Emacs>), Emacs can be defined as follows:

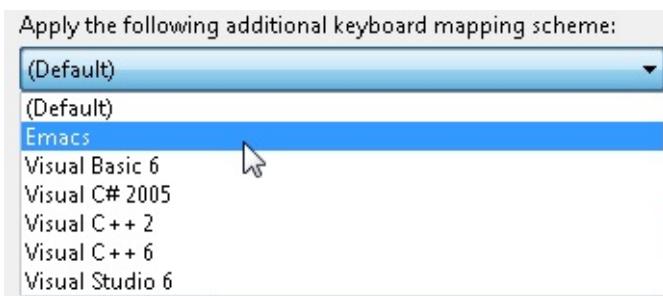
*[...] a class of feature-rich text editors, usually characterized by their extensibility. Emacs has, perhaps, more editing commands than other editors, numbering over 1,000 commands. It also allows the user to combine these commands into macros to automate work.*

*“Development began in the mid-1970s and continues actively as of 2010. Emacs text editors are most popular with technically proficient computer users and computer programmers. The most popular version of Emacs is GNU Emacs, a part of the GNU project, which is commonly referred to simply as ‘Emacs’.”*

Did you know that Visual Studio supports Emacs emulation? If you are using Visual Studio 2010, you need to install the Emacs extension that can be found here:

<http://visualstudiogallery.msdn.microsoft.com/en-us/09dc58c4-6f47-413a-9176-742be7463f92>

Then, for Visual Studio 2008, 2005, and 2010, go to Tools | Options | Environment | Keyboard and choose Emacs from the Apply The Following Additional Keyboard Mapping Scheme drop-down list:



## AX.152 ViM Emulation

VERSIONS	2005, 2008, 2010
CODE	vstipEdit0080

Many people like to use ViM—there is no denying it. Unfortunately, Visual Studio does not support ViM emulation out of the box. However, as of the time of this writing, you have two solutions in the Visual Studio Gallery, depending on your version.

### Visual Studio 2010

VsVim by Jared Parsons is very well reviewed, plus it is free. You can find it at <http://visualstudiogallery.msdn.microsoft.com/en-us/59ca71b3-a4a3-46ca-8fe1-0e90e3f79329>.

### Visual Studio 2008 and Prior

ViEmu by NGEDIT Software is available for Visual Studio versions prior to Visual Studio 2010. It has good reviews, and the trial version is available from the gallery. It is available at <http://visualstudiogallery.msdn.microsoft.com/en-us/C9055830-39AB-4B39-A19E-4D60F195E7FC>.

So, if you need ViM emulation, you have a couple options. These might not be the only options, but they are readily available in the Visual Studio Gallery.

## About the Authors

Zain Naboulsi is a Senior Developer Evangelist at Microsoft® and frequently lectures on Visual Studio® topics.

Sara Ford is the Senior Product and Community Manager at Black Duck Software for Ohloh.net, the largest public destination for finding and evaluating open source software. Prior to Black Duck, she worked for nine years at Microsoft Corp., where she was responsible for CodePlex.com, the open source project hosting forge for Microsoft. She started her career as a software tester on Visual Studio, a software development tool, where she drove the effort to make it possible for developers who are blind or have low-vision be able to write software applications. She is the author of Visual Studio Tips, published by Microsoft Press, where she donated her author royalties to start a scholarship fund ([http://www.mgcccc.edu/news/book\\_raises\\_scholarship\\_money\\_for\\_MGCCCC\\_students.php](http://www.mgcccc.edu/news/book_raises_scholarship_money_for_MGCCCC_students.php)) designed for residents of her hometown Waveland, Miss. to attend the Mississippi Gulf Coast Community College.

# Coding Faster: Getting More Productive with Microsoft® Visual Studio®

Zain Naboulsi

Sara Ford

Editor  
Russell Jones

Copyright © 2011 Zain Naboulsi and Sara Ford

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at [mspinput@microsoft.com](mailto:mspinput@microsoft.com). Please tell us what you think of this book at <http://www.microsoft.com/learning/booksurvey>.

Microsoft and the trademarks listed at <http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, O'Reilly Media, Inc., Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Microsoft Press  
A Division of Microsoft Corporation  
One Microsoft Way  
Redmond, Washington 98052-6399

2014-02-06T20:59:05-08:00