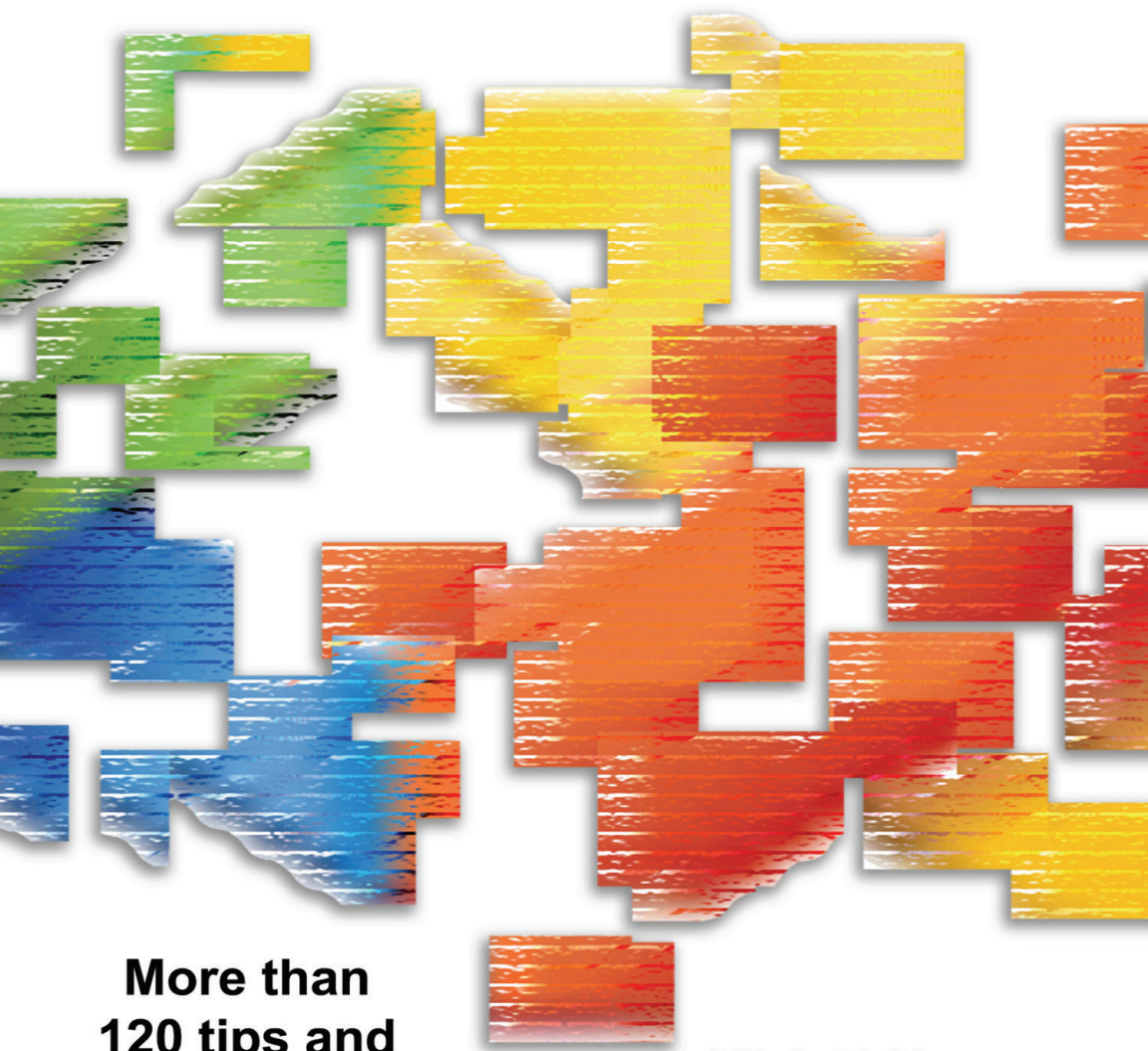


Visual Studio .NET Tips and Tricks

Covers versions 2002, 2003, and 2005



**More than
120 tips and
tricks!**

Minh T. Nguyen

FREE ONLINE EDITION

(non-printable free online version)

If you like the book, please support
the author and InfoQ by
purchasing the printed book:

<http://www.lulu.com/content/78560>

(only \$13.95)

Brought to you
Courtesy of
Minh T. Nguyen &



This book is distributed for free on InfoQ.com, if you
have received this book from any other source then
please support the author and the publisher by
registering on InfoQ.com

Visit the homepage for this book at:

<http://infoq.com/books/vsnettt>

http://www.enderminh.com/minh/vsnet_tt.aspx

Visual Studio .NET Tips and Tricks

Minh T. Nguyen

Visual Studio .NET Tips and Tricks
by Minh T. Nguyen
www.enderminh.com

First Edition: September, 2004

ISBN: 1-4116-1396-1

Copyright © 2004

All rights reserved by the author. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publisher and/or author.

*To my parents
who 25 years ago sacrificed everything
so that I could live the American Dream*

and

*to Minh Thu
for sharing this dream with me*

Contents

Chapter 1: Editing Code	1
Adding XML Commenting	2
Commenting Web Pages	3
Using IntelliSense Across Projects	3
Inserting Comment Tokens	4
Commenting Code Blocks	5
Creating Regions	6
Hiding the Current Selection	7
Selecting a Single Word	8
Selecting an Entire String Literal	8
Switching to the Front and End of a Selection	8
Placing Code into the Toolbox	8
Using the Clipboard Ring	9
Transposing a Single Character or Word	9
Cutting, Copying, Deleting, and Transposing a Single Line	10
Editing XML in Tabular Mode	10
Pasting Text as HTML	12
Adding Class Field Members	12
Formatting Entire Blocks	12
Toggling Word-Wrapping	14
Forcing IntelliSense for Field Members	15
Forcing IntelliSense for Parameter Information	15
Completing a Word	16
Implementing Methods of an Interface	16
Overriding Methods	18
Creating GUIDs	19
Creating Rectangular Selections	19
Switching Between Views	20
Going to a Line Number	20
Searching for a Word	21
Finding and Highlighting Matching Tokens	21
Performing an Incremental Search	21
Searching or Replacing with Regular Expressions or Wildcards	22
Performing a Global Search or Replace	23
Using Bookmarks	25
Going to the Definition of a Method	27
Using Browser-Like Navigation	27
Inserting External Text File	28
Inserting JavaScript Tags	28
Outlining HTML/Form Hierarchy	29

Chapter 2: Exploring the IDE _____ 31

Opening with Default Action _____	32
Showing Extra Files _____	32
Setting Project Dependencies _____	33
Embedding Files As Resources _____	34
Changing the Font Size of IDE Windows for Demos _____	34
Dragging Files to Obtain a Full Path _____	35
Moving Any Window Around _____	36
Creating Split Screens in the Same File _____	37
Customizing the VS.NET Menu and Toolbars _____	38
Adding External Programs to the VS.NET Menu _____	39
Dragging Files from Windows Explorer into VS.NET _____	40
Accessing the Command Window _____	41
Aliasing Your Favorite Commands _____	42
Switching to Immediate Mode from the Command Window _____	42
Using the Command Window in Find a Drop-Down List _____	42
Using the Built-in Web Browser _____	43
Using Full-Screen Mode _____	44
Copying the Fully Qualified Name of a Class _____	45
Recording and Replaying a Temporary Macro _____	45
Saving, Editing, and Debugging Macros _____	46
Assigning Shortcuts and Menu Items to Macros _____	46
Changing Properties of Several Controls _____	48
Locking Controls _____	49
Toggling the Description in the Properties Window _____	49
Change Drop-Down List Values in the Properties Window _____	50
Adding and Removing Event Handlers Through the IDE _____	50
Selecting Control Through a Drop-Down List _____	51
Adding an Installer Through the Designer for Windows Services _____	52

Chapter 3: Compiling, Debugging, and Deploying _____ 53

Linking Files Instead of Copying Them into a Project _____	54
Setting the Default Namespace and Assembly Name _____	54
Generating Compiler Warnings and Errors _____	55
Generating Compiler Warnings Through the Obsolete Attribute _____	56
Setting Pre/Post Compile Build Steps _____	56
Setting the Assembly Output Path _____	57
Setting the .NET Framework Version for Your Assembly _____	57
Deploying ASP.NET Web Applications _____	59
Moving the Next Statement During Debugging _____	60
Changing Variable Values in the Watch Window _____	61
Executing SQL Procedures Through the Server Explorer _____	61
Using the Immediate Window to Display Variables and Execute Methods _____	62
Customizing the Call Stack _____	63
Placing Program Command-Line Arguments into Project Properties _____	64
Attaching VS.NET to an Already Running Process _____	65
Debugging Several Projects Inside the Solution _____	66
Breaking Only for Certain Exception Types _____	67

Breaking Only When Certain Conditions Apply	69
Debugging ASP.NET Web Application Through Trace.axd	71
Saving Any Output Window	73

Chapter 4: Using VS.NET 2005 **75**

Refactoring Code	76
Generating Method Stubs	77
Using Error Correction Suggestions	79
Using Predefined Code Snippets	79
Aligning UI Elements Automatically	82
Adding a Standard Menu Strip	82
Editing UI Element Properties with the Property Editing View	83
Controlling C# Code Formatting Precisely	84
Setting the Tab Order of Controls	85
Performing a Class, Instance, and Method Search	85
Viewing Code Definitions	85
Editing Web Controls in the HTML View	86
Validating HTML Code for Accessibility	87
Working with Different .NET Languages	88
Opening Web Projects Through FTP	88
Importing and Exporting IDE Settings	89
Closing All Other Windows	91
Showing Shortcuts for All Buttons	91
Building Selected Subset of Projects	92
Using Edit-and-Continue in VB.NET	92
Expanding Variable Members While Debugging	93
Using Data Visualization	94

Chapter 5: Other .NET Tips and Tricks **97**

Obfuscating Your .NET Assemblies	98
No Rebuilding After HTML Changes	98
Iterating Over Strings on a Char per Char Basis	98
Using Inline Strings as Object Instances	99
Adding App.config to Your Application	99
Using Intermediate Language Disassembler to Inspect a .NET Assembly	100
Using Windows Class Viewer to Reference a Class	101
Running aspnet_regiis to Fix an IIS Installation	102
Precompiling Your ASP.NET Web Application	103
Setting ASP.NET Versions for Web Applications	104
Clearing the Assembly Cache Manually	104
Using Unicode for Strings, Labels, and Inline .NET Code	105
Rethrowing the Same Exception	106

Index **109**

Introduction

When Microsoft introduced C# and Visual Basic .NET upon the release of the .NET Framework in 2002, they also gave the developer community a brand-new integrated development environment, Visual Studio .NET. Over the past few years, while developers argued over which .NET language was better or whether .NET will hold its promise of being “the next big thing,” the vast majority of developers have come to agree on one thing: Visual Studio .NET is one of the most scalable, capable, and flexible development tool that Microsoft has ever released.

Visual Studio .NET is filled with thousands of features and functionalities that make our lives as developers more efficient. Not only is it a code editor, compiler, and debugger, but it contains features to stress-test, analyze, and optimize your code, and to allow easy integration with code documentation, reporting, or smart-device programming, such as the Pocket PC.

The number of features that Visual Studio .NET contains is immense. I doubt that many .NET developers know all of its features, shortcuts, and functionalities. Being a fan of keyboard shortcuts and tidbits that make my life as a developer better, or simply faster, I began to collect Visual Studio .NET tips and tricks. What started out as a small list became bigger and bigger, leading to workshops I gave in Visual Studio .NET. Those workshops finally evolved into this book.

Visual Studio .NET Tips and Tricks is a compilation of my favorite, or the most popular, tips and tricks that exist in this great IDE. This is not a comprehensive list, however, because that would have resulted in a book probably twice this size. I also consciously skipped a few obvious tips and tricks that I assume all developers should know. This book features the most common, most popular, or most effective tips and tricks that developers will find very useful. Beginners will find a box of treasures, while advanced Visual Studio .NET users will be amazed by how many new features and improvements the new Visual Studio .NET 2005 brings.

The majority of these tips and tricks are not undocumented. On the contrary, most of them are accessible through the VS.NET main menu or context menus. However, given the vast amount of features with which VS.NET is equipped, I often find that developers do not know them, or do not use them, as much as I think they should.

This book is based on the Beta 1 version of Visual Studio .NET 2005. As such, the final product might vary from how I describe it.

I welcome you to dive into this book. I hope you enjoy reading about these tips and tricks as much as I enjoyed discovering them.

Minh T. Nguyen
September, 2004
Bellevue, Washington

Chapter 1: Editing Code

Programmers perform a variety of daily tasks: We attend team meetings, design and test our programs, write documentation, and perform code reviews. But writing code is one task that awaits every programmer. If you love to code, Microsoft Visual Studio .NET is perfect for you because it offers many ways to help you write and edit code. This chapter covers some of the tips and tricks you can use to write and navigate through your code much more quickly than before. From code comments and code navigation to generating code snippets and performing complex find-and-replace searches, this chapter covers all you need to know while you write code.

Adding XML Commenting

Probably the feature most talked about in VS.NET is the ability to extend IntelliSense—VS.NET’s popular feature that shows you member fields of a given object instance—by inserting code comments into your user-defined classes and methods. You do this in C# by placing three forward-slashes (“///”) in front of a class definition or member. VS.NET automatically creates the necessary XML-formatted segments so you can insert the description and parameter information (see Figure 1). After rebuilding the project, VS.NET is equipped with the information you have specified so it can display it using IntelliSense. This information includes comments for methods, method parameters, method return values, enumerations, enumeration values, and properties.

```
/// <summary>
/// Provides some helper utilities
/// </summary>
public class HelperClass
{
    /// <summary>
    /// Copies specified file into backup directory
    /// </summary>
    /// <param name="fileName">Full path to physical file</param>
    public static void BackupFile(string fileName)
    {
        // ...
    }
}
```

Figure 1 - Adding XML comments to C#

In addition to the much-seen <summary> tag (the description displayed by IntelliSense), you can insert the following tags into your comments (don’t forget the respective closing tags):

- <example>
- <exception>
- <include>
- <paramref>
- <permission>
- <remarks>
- <value>

VS.NET 2002 and 2003 do not support XML commenting for VB.NET. However, VS.NET 2005 does. All you need to do is type three apostrophes to generate the XML comment fragment for filling out (see Figure 2). Alternatively, you can right-click the construct you want to comment and choose Insert Comment from the pop-up menu.

```
''' <summary>
''' Prints a text file to the printer
''' </summary>
''' <param name="filePath">Full path of the text file to be printed</param>
''' <param name="printCover">Prints a cover page before the text file</param>
''' <remarks></remarks>
Public Sub PrintFile(ByVal filePath As String, ByVal printCover As Boolean)
```

Figure 2 - XML Commenting for VB.NET in VS.NET 2005

Certainly one of the reasons why the .NET Framework is so intuitive is the massive descriptions presented through IntelliSense. I highly recommended that you use this feature for your own projects.

Commenting Web Pages

The XML comments that you place in your code not only show up through IntelliSense but also can be used to generate quickly a web document that describes all your classes and their fields. Do this by selecting Tools > Build Comment Web Pages. Choose to output the comments of only one project or the entire solution.

The produced Code Comment Web Report lists all classes sorted by their namespace and projects, along with descriptions of class members and inheritance information.

If you do not like the layout of the predefined template, you can export the comments into an XML file and then create your own XSLT transformation to format it for your own needs. Just right-click your project in the Solutions Explorer and choose Properties > Build from the pop-up menu. Enter an XML filename in the XML Documentation File field. Upon building the project, VS.NET outputs the complete code comments into this XML file. Because this XML file is a complete list of comments, all XML comments must be present. In other words, each undocumented method, property, enumeration, and so on will produce a warning message during compilation.

The ability to generate code comments web reports no longer appears in VS.NET 2005.

Using IntelliSense Across Projects

If you use XML commenting, you might notice that code comments do not appear in IntelliSense across projects in the same solution.

You can fix this problem by giving the XML file the same name as the assembly ("MyAssemblyName.xml"). Doing this ensures that other projects in the same solution

automatically pick up this XML file whenever you add a reference to the project and display the comments through IntelliSense. This trick works only if you set the Copy Local property of the reference to True. Because VS.NET just copies the XML and assembly from the referenced project into the working project's bin directory, any changes you make to the comments in the referenced project are not reflected in IntelliSense until you update this XML file again. Unfortunately, the only way to do this quickly is to set Copy Local to False, wait a few seconds, and then set it back to True. VS.NET will actually delete the referenced copies of the XML files and then copy them back from the referenced project.

Fortunately, in VS.NET 2005 there is no need to generate XML files in order to enjoy IntelliSense support across multiple projects. Projects within the same solution simply have full IntelliSense support automatically. You don't even have to recompile when you change comments.

Inserting Comment Tokens

Do you often write a comment to remind yourself of a task you still need to do, and then forget where it is? You can insert *comment tokens* to help you. These are keywords that VS.NET recognizes and can compile in a list. For instance, place a TODO comment anywhere in your code, as shown in Figure 3.

```
// TODO: Double-check this algorithm
for(int i=0;i<100;i++)
{
    // ...
}
```

Figure 3 - Comment tokens

Without recompiling, select View > Show Tasks > All to see a list of all reminders you placed in your code (see Figure 4). In C#, VS.NET only shows you the TODOs of any currently opened documents, whereas in VB.NET it shows you all TODOs in the entire solution.

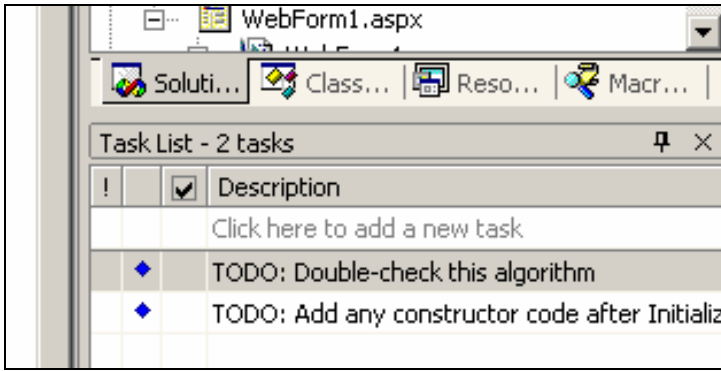


Figure 4 - TODO tokens in the task list

In VS.NET 2005, select View > Other Windows > Task List (or press Ctrl-Alt-K) and filter the tasks on “Comments” using the drop-down menu above the Task Lists.

In addition to TODO, VS.NET comes preconfigured with HACK, UNDONE and UnresolvedMergeConflict tokens. You can configure the priority of these tasks, as well as add your own tokens, by selecting Tools > Options > Environment > Task List.

Note: Comment tokens are case-sensitive.

Besides using comment tokens, you can place *task shortcuts* in your code by pressing Ctrl-K, Ctrl-H or by right-clicking your code and choosing Add Task List Shortcut from the pop-up menu. This marks the current line with a shortcut icon and places a clickable shortcut icon in the Task List. Use the same keyboard combination to remove the shortcut. These shortcuts survive IDE restarts.

Commenting Code Blocks

You can place one-line comments in code using the “//” token for C# and the “'” token for VB.NET. Additionally, C# allows you to comment entire paragraphs and segments by placing the “/*” (and corresponding “*/”) tag around the comment.

A faster way to comment entire paragraphs is to select the text and click the Comment button (see Figure 5) or press Ctrl-K, Ctrl-C. This comments an entire selection. Uncomment any selection by clicking the Uncomment button (or Ctrl-K, Ctrl-U).



Figure 5 - Comment and Uncomment buttons

Creating Regions

As your code gets longer and longer, it becomes more and more difficult to navigate. While you can select classes and their methods from the drop-down lists above the main editor, you can also group your code into logical regions. Specify regions with the `#region` keyword and a description at the beginning of your segment and a corresponding `#endregion` keyword at the end of your segment (see Figure 6).

```
#region calculation algorithm

// TODO: Double-check this algorithm
for(int i=0;i<100;i++)
{
    System.Diagnostics.Debug.WriteLine(i);
}

#endregion
```

Figure 6 - Creating regions around code

The beauty of using regions is that you can collapse any region by clicking the plus sign next to the `#region` keyword. This collapses the code into a gray line that shows the region description. As a developer, you may already be aware of this feature because automatically generated code in VS.NET usually uses this feature.

Note: In VB.NET you specify the region description in double-quotes. Also, regions are not allowed within methods. Instead, create regions around methods and classes.

You can quickly see the inside of a collapsed region by mousing over the gray description line (see Figure 7). You can even drag and drop collapsed regions inside your code. Unfortunately, once you paste a collapsed region into a different location, the pasted text is automatically expanded.

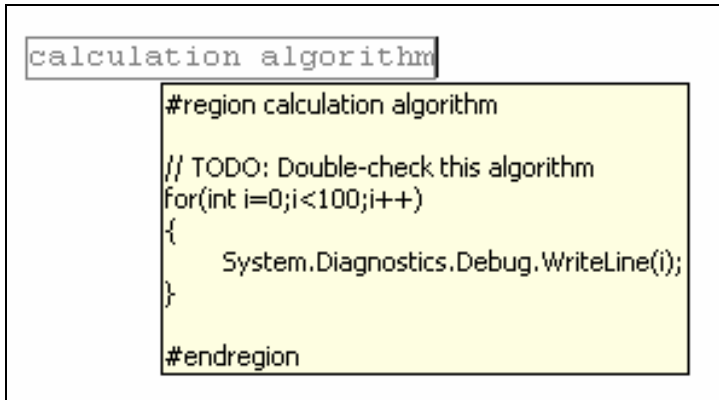


Figure 7 - Mousing over a region to see its content

Expand and collapse the current region you are in by pressing Ctrl-M, Ctrl-M.

Regions are extremely helpful for dividing code logically or even commenting it, especially because you can nest them. Expand or collapse all regions at once by right-clicking the gray bar to the left of the main editor window. The Outlining menu displays various collapse and expand options.

Hiding the Current Selection

Regions are created for methods, comments, and sections encompassed by the #region compiler directive. In VB.NET and in regular text files, you can create regions around any section without the need for #region. Just highlight the section you want to hide and press Ctrl-M, Ctrl-H. This hides the current selection in a temporary collapsed region (see Figure 8). Expand them by pressing Ctrl-M, Ctrl-M. Temporary regions are lost after you close a project.

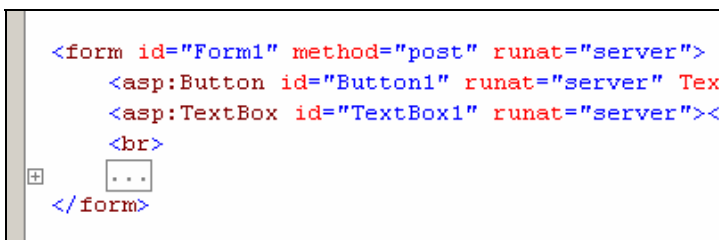


Figure 8 - Hiding portions of any text file

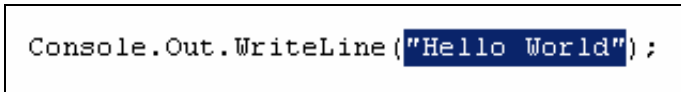
Because this trick works for any text file, it proves to be extremely useful for collapsing HTML tags. This trick works for VB.NET code using VS.NET 2002 or 2003 (but not 2005).

Selecting a Single Word

Selecting a single word is a very common task when editing code. You can do this by double-clicking anywhere in the word, or just pressing Ctrl-W.

Selecting an Entire String Literal

Another common task is selecting an entire string literal. If you want to select all the text that lies between two double-quotes, move your mouse to the left of the opening quote and double-click that area. The entire string literal (including both quotes) will be selected for you (see Figure 9). Instead of double-clicking, you can also single-click with the Ctrl key held down. This trick only works in C#.

A screenshot of a code editor window. The code is `Console.Out.WriteLine("Hello World");`. The opening double quote of the string literal is highlighted with a blue selection bar, and the entire string literal, including the closing double quote, is also highlighted. The rest of the code is not selected.

```
Console.Out.WriteLine("Hello World");
```

Figure 9 - Double-clicking the opening quote to select the entire string

Switching to the Front and End of a Selection

Whenever you want to jump quickly to the front or end of a block of selected text, you can repeatedly press Ctrl-R, Ctrl-P. This allows you to jump back and forth between these two points.

Placing Code into the Toolbox

Every project contains certain snippets of code or text that you use over and over again—perhaps a standard copyright header you place at the top of each file or a certain code snippet to perform a common task. If you find yourself pasting this snippet of text many times into files, it might be useful to place it into your Toolbox (the window that lists all windows or web controls and that you can pull up by pressing Ctrl-Alt-X). Just highlight your text snippet and drag the selected text onto the General tab in your Toolbox (see Figure 10).

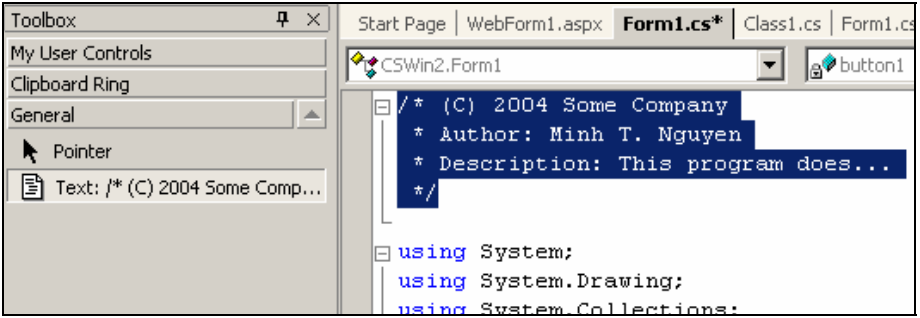


Figure 10 - Adding a text selection into the Toolbox for reuse

Rename the produced text snippet in your Toolbox by right-clicking it and choosing Rename Item from the pop-up menu. Just as with any other control in the Toolbox, you can now drag it into your code window or just double-click the entry to insert the fragment at the cursor's current location.

The General tab in the Toolbox is project- and solution-independent; therefore, it retains its content even after you restart VS.NET.

Using the Clipboard Ring

The Clipboard Ring contains the last several dozen text snippets that you placed on the Clipboard. It works very much like the Clipboard Ring you may be familiar with in Microsoft Office, and comes in handy when you accidentally overwrite the current Clipboard content or when you find yourself juggling several different items.

The Clipboard Ring is a tab in the Toolbox. You can either double-click one of the remembered Clipboard items to paste it at the cursor's current location or just drag it into the editor.

When the Clipboard Ring contains many Clipboard items, or when you cannot see the complete contents of each item in the ring, it's useful to cycle through the Clipboard Ring. Selecting Edit > Cycle Clipboard Ring (or pressing Ctrl-Shift-V) repeatedly makes VS.NET cycle through the Clipboard Ring's contents, displaying the stored Clipboard contents in the text editor at the cursor's current location. The beauty of cycling through the Clipboard Ring is that it makes it easy to paste specific content in the code editor as it becomes visible during the cycling. Continue to cycle through the Clipboard's contents until you find the item you are looking for and then you are done.

Transposing a Single Character or Word

Transposing a single character or word means swapping the current characters or words around the cursor and moving the cursor to the right. If you mistype a word or write a

sentence or code segment with words in the incorrect order, you can use transposing to fix this.

Transpose a single character by pressing Ctrl-T. This swaps the two characters surrounding the cursor and moves the cursor to the right by one character. Pressing Ctrl-T repeatedly allows you to move a single character further to the right.

Transpose a single word by pressing Ctrl-Shift-T. Note that this does more than just swap two adjacent words. VS.NET is smart enough to ignore “unimportant” single characters, such as equal signs, string quotes, white spaces, commas, etc.

Suppose you have a line of code that originally looks like this:

```
new SqlCommand("trans", stored_procedure, conn);
```

Pressing Ctrl-Shift-T repeatedly on the word “trans” would yield the following:

```
new SqlCommand("stored_procedure", trans, conn);
```

and finally this:

```
new SqlCommand("stored_procedure", conn, trans);
```

The quotation marks and commas retain their original positions throughout the process. When you reach the end of a line, pressing Ctrl-Shift-T transposes the word with the first word of the next line.

Cutting, Copying, Deleting, and Transposing a Single Line

To copy the complete, current line to the Clipboard, press Ctrl-C (or click the Copy icon) without any text selected. The same thing applies if you want to cut an entire line. Without anything selected, press Ctrl-X (or click the Cut icon) to cut the entire current line and place it in the Clipboard.

You can delete a single line by just pressing Ctrl-L. If you want to swap the current line with the one below it, press Alt-Shift-T. Doing this also moves the cursor down by one line. This allows you to press this keyboard shortcut repeatedly until you move your current line to the desired position.

Editing XML in Tabular Mode

Editing XML in Text mode can be difficult at times. While the text editor beautifully does insert closing tags, you often have to copy and paste entire XML nodes just to insert a new element (see Figure 11).

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person>
    <FirstName>John</FirstName>
    <LastName>Doe</LastName>
    <Address>
      <StreetNumber>1234</StreetNumber>
      <StreetName>Main St</StreetName>
      <City>Los Angeles</City>
      <State>CA</State>
      <ZIP>12345</ZIP>
    </Address>
  </Person>
  <Person>
    <FirstName>Jane</FirstName>
    <LastName>Doe</LastName>
    ...
  </Person>
</Persons>
```

Figure 11 - Editing XML in Text mode

A better way to edit the data part of an XML file is to switch to the Data mode (by clicking Data at the bottom of the main window). This presents you with a tabular format of the XML (see Figure 12), which allows you to edit the raw data and drill down to child elements by clicking on the plus sign next to each row. You can even insert new XML nodes by simply inserting data into the last row (the one marked with an asterisk). Once you complete the row, the corresponding XML node with its tags and contents are inserted into the actual XML file.

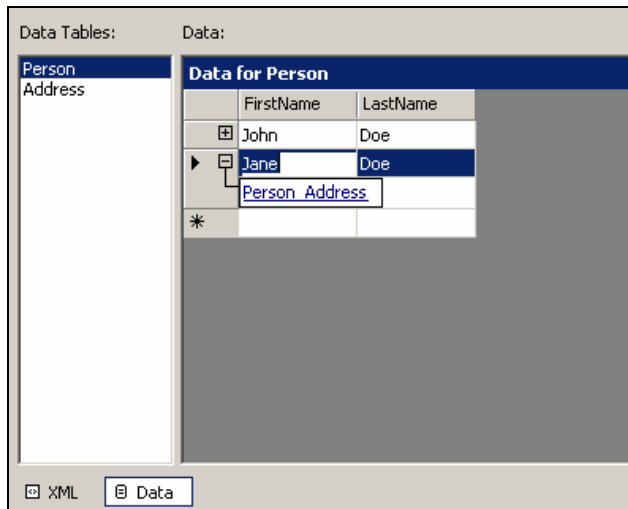


Figure 12 - Editing XML in Data mode

This tabular mode is really meant for editing only XML-serialized datasets. Editing XML files with a deeper hierarchy produces misplaced XML fragments.

In order to get to this tabular mode in VS.NET 2005, select View > Data Grid.

Pasting Text as HTML

Whenever you copy a segment of formatted text (from a website or word processor, for instance) and paste it into your editor using the usual Paste command, or keyboard shortcut Ctrl-V, you will see that VS.NET is “smart” enough to insert the text with additional HTML code so that its formatting is retained. In most cases this is not what you expect from the normal Paste behavior.

If you want to paste the copied text verbatim, without fancy formatting, use VS.NET’s Paste As HTML feature. For instance, if you copy a code snippet from a programming website, you probably want to paste the code as code, without an additional formatting. Using the Paste As HTML feature places only the text content into your editor.

Unfortunately, these command names are counterintuitive: Paste As HTML should be the normal Paste functionality and the current Paste should be named “Paste As HTML” because that’s exactly what it does.

To make things more intuitive, VS.NET 2005 has made “Paste As HTML” the default behavior for the Paste command and renamed “Paste As HTML” as the Paste Alternate command.

Adding Class Field Members

You can add class field members (properties, variables, indices, and methods) easily using an Add dialog box. Just go to the Class view, right-click your class, and navigate to the appropriate Add dialog box item (Add Method, Add Property, Add Field, or Add Indexer). Using the dialog box, you can enter field member information such as field type, name, and access modifiers. However, in some cases it might be faster just to type the code manually.

In addition, the Add dialog box items do not always place the added class member at a desired location in your code. These items are not available in VS.NET 2005 because the new refactoring methods provide the same functionality (see “Refactoring Code” in Chapter 4).

Formatting Entire Blocks

There are several useful functions you can use to modify a given selection: Upper casing, lower casing, or deleting horizontal white space are just a few examples. Select Edit > Advanced to access these features. One of the most useful features here is the Format Selection function. Accessible also by pressing Ctrl-K, Ctrl-F, it formats an entire

selection and inserts tabs where appropriate to modify the code with the correct code-specific block indentation. This is usually done automatically when someone enters code upon closing a block (such as by typing the “}” sign in C#) but Format Selection forces this automatic format in case entire blocks are already messed up (see Figures 13 and 14).

```
public static void DoAlgorithm()
{
#region calculation algorithm
    // TODO: Double-check this algorithm
    for(int i=0;i<100;i++)
    {
        Debug.WriteLine(i);
    }
#endregion
}
}
```

Figure 13 - Before formatting block

```
public static void DoAlgorithm()
{
    #region calculation algorithm
    // TODO: Double-check this algorithm
    for(int i=0;i<100;i++)
    {
        Debug.WriteLine(i);
    }
#endregion
}
```

Figure 14 - After formatting block

In VS.NET 2002 and 2003, this feature works only for C# or VB.NET code, not for XML or HTML code. In VS.NET 2005, however, you can format markup-language files as well (see Figure 15). In fact, this formatting feature is so popular that in VS.NET 2005 there is a button to format the entire document (circled in Figure 16). You can do the same by selecting Edit > Advanced > Format Document or pressing Ctrl-K, Ctrl-D.

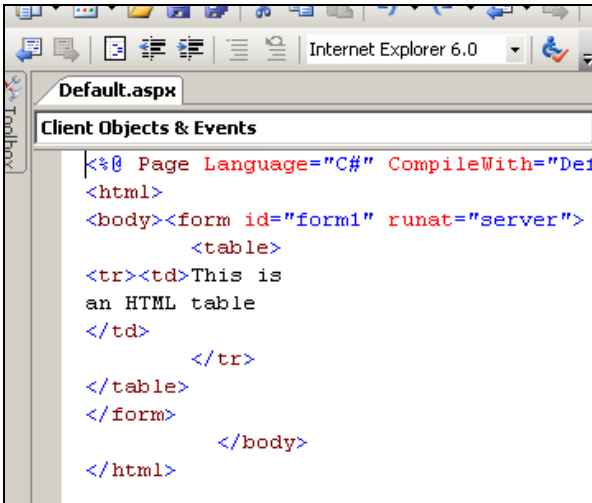


Figure 15 - Formatting markup language in VS.NET 2005

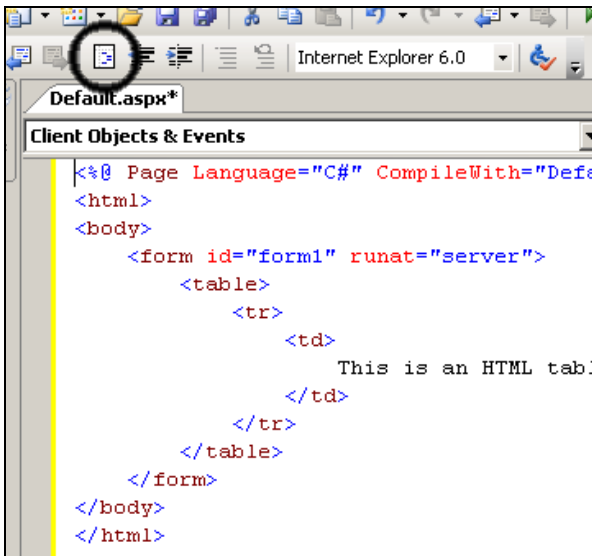


Figure 16 - Formatting whole document button in VS.NET 2005

Toggling Word-Wrapping

Select Edit > Advanced to turn word-wrapping on and off for the current view. As usual, it is easier to use the keyboard shortcut (Ctrl-R, Ctrl-R) instead.

Forcing IntelliSense for Field Members

IntelliSense is an extremely useful feature. Every VS.NET programmer literally relies on it to navigate the .NET Framework class hierarchy. While IntelliSense usually runs as soon as you type the period after a class instance, there are occasions where you need to force IntelliSense to appear. For instance, suppose you misspell a method name. You can bring IntelliSense up only by completely backspacing to the period and starting all over again.

In those situations, you can force IntelliSense to run by pressing Ctrl-J. This allows you to select the appropriate method through IntelliSense and replace the misspelled method name with the selected one. VS.NET 2005 has a default button for this operation (see Figure 17).



Figure 17 - Forcing IntelliSense (first button) in VS.NET 2005

Forcing IntelliSense for Parameter Information

Just as you can force IntelliSense for class field members, you can also force it for parameter information. I find this feature extremely useful when I need to look up information about a parameter for a given method of a class in a file that is currently checked into Microsoft SourceSafe. Currently, the only way to pull up the parameter information is to modify the actual parameter list. Unfortunately, you can do this only after you check the file out of Source Safe.

Instead of checking out the file just to view parameter information, simply place the cursor over the parameter and press Ctrl-Shift-Spacebar. VS.NET 2005 provides a button for this feature (see Figure 18).



Figure 18 – Forcing parameter info (second button) in VS.NET 2005

Note: The third button on the IntelliSense toolbar (Ctrl-K, I) simply displays the quick info of the word you are currently in. In other words, it displays what you would usually see by mousing over the current word.

Completing a Word

One of the most popular shortcuts in the entire VS.NET IDE is probably the Ctrl-Spacebar shortcut. When you start typing a word, and you have typed enough leading characters so that it uniquely identifies a given word in your current file, try pressing Ctrl-Spacebar. The usual IntelliSense drop-down list will be indexed by the specified starting characters. Simply choose the appropriate selection to complete the word quickly.

If the series of starting letters you type uniquely identifies a single word in the IntelliSense list, then the complete word will be inserted automatically. As you become used to working this way, you can type entire code segments at an extremely fast pace. VS.NET 2005 provides a button for this feature (see Figure 19).



Figure 19 – Completing a word (last button) in VS.NET 2005

Implementing Methods of an Interface

If you are writing a class and you decide that it will implement a certain interface, you can allow VS.NET to generate the stubs for all required methods of the interface. There are two ways of doing this.

As you write your class in VS.NET 2003, you should see a ToolTip right after you type the interface name that says that VS.NET can complete all interface members by pressing the Tab key. After you press Tab, VS.NET automatically inserts the stubs of the interface in a nicely sectioned-off region.

If you have already defined the interface once before and you need to add the interface implementations, then go to the Class view, right-click the class' interface under the “Bases and Interfaces” tree node, and navigate to the Add > Implement Interface pop-up menu item (see Figure 20). This feature only works for C# in VS.NET 2003.

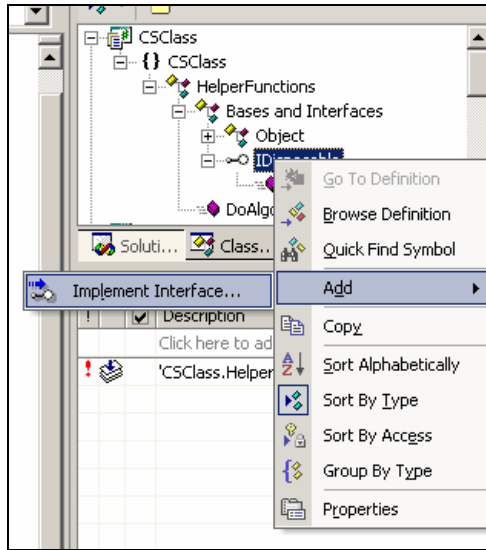


Figure 20 - Implementing interface methods

With VS.NET 2005, you don't implement interfaces through that menu item. In VB.NET all the interface methods are automatically generated (you really don't have a choice). In C#, you use the smart-tag that appears as you click your interface in your editor (see Figure 21).

```
public class Foo : IDisposable
{
    // implicit
    #region IDisposable Members

    public void Dispose()
    {
        throw new NotImplementedException();
    }

    #endregion

    // explicit
    #region IDisposable Members

    void IDisposable.Dispose()
    {
        throw new NotImplementedException();
    }

    #endregion
}
```

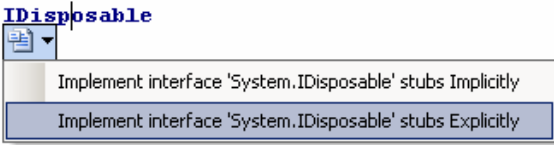


Figure 21 - Implementing interfaces with VS.NET 2005

Overriding Methods

In the same fashion as implementing the interface, you can override methods of a base class by going through the same menu. Go to the Class view, expand one of the base class nodes under the “Bases and Interfaces” tree node, right-click the method you want to override, and navigate to Add > Override.

This immediately writes out an empty method that overrides a base method. With VS.NET 2003, you can also find out which methods are capable of being overridden by simply starting out the method declaration. Typing **public override**, for example, brings up IntelliSense showing all methods that you can override. By choosing a method from this list, you will create the method with the correct signature and a call to its base method as the method body.

Creating GUIDs

As you develop new classes and components, you often need to create so-called Global Unique Identifiers (GUIDs) which are 128-bit values often represented by 32 hexadecimals. In the beginning, component developers used GUIDs to assign their components with unique names because the likelihood of two components sharing the same GUID was extremely small. These days, developers use GUIDs for virtually anything that requires a unique identifier. Manually creating GUIDs by randomly selecting 32 hexadecimals is rather tedious.

Fortunately, VS.NET is preshipped with a utility that creates GUIDs for you whenever you need one. Select Tools > Create GUID to open the Create GUID dialog box (see Figure 22). Here you can generate identifiers in various formats, including common code snippets often used in COM development.

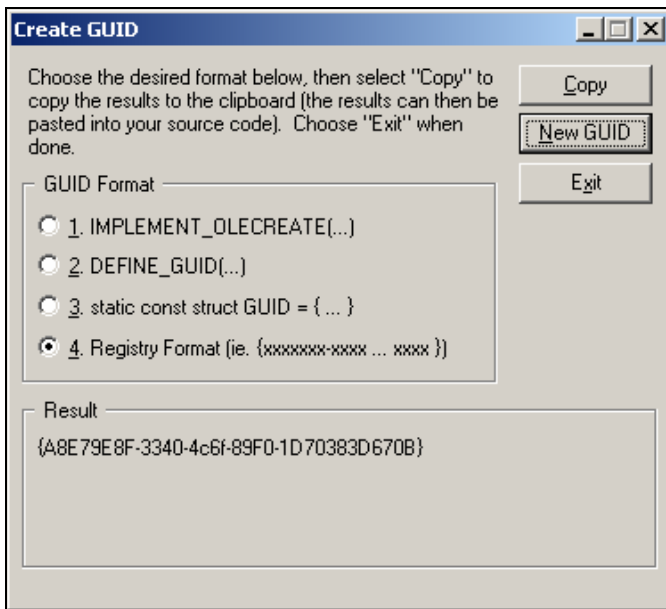


Figure 22 - Create GUIDs

Unfortunately, this utility does not offer a bare-basic format in which the 32 hexadecimals are simply provided in a sequence. This would be very useful as well.

Creating Rectangular Selections

Did you know that you can make rectangular selections in VS.NET by pressing the Alt key while dragging the mouse over a rectangular area? This technique allows you to

create a rectangular selection that does not include the intervening lines (see Figure 23). Copying, cutting, and pasting rectangular blocks can be done very easily this way.

```
/// Summary description for WebForm1.
/// </summary>
public class WebForm1 : System.Web.UI.Page
{
    protected System.Web.UI.WebControls.Button Butt
    protected System.Web.UI.WebControls.DataGrid Da
    protected System.Web.UI.WebControls.Button Butt
    protected System.Web.UI.WebControls.LinkButton
    protected System.Web.UI.WebControls.Repeater Re
    protected System.Web.UI.WebControls.Calendar Ca

    private void Page_Load(object sender, System.Ev
    {
```

Figure 23 – Press and hold the Alt key to create rectangular selections

You might wonder why anyone would ever need this esoteric feature. In fact, I have used it often to make targeted query-replaces in certain parts of a document that otherwise prevented me from doing so because of normal line-wrapping selections. I assure you that you will find this technique very handy.

Switching Between Views

When you develop ASP.NET applications, you often need to switch between the Designer view and the HTML view. Instead of moving the mouse to the appropriate view button, you can switch between them much faster by pressing the Ctrl-PgUp and Ctrl-PgDn keyboard shortcuts. (For web forms, it really doesn't matter which shortcut you use because either one will switch to the other view.) This also works for switching between the XML view and the Data view when reading XML documents.

For Windows forms, press F7 to switch to the Code view and press Shift-F7 to switch to the Designer view.

In VS.NET 2005, Ctrl-PgUp and Ctrl-PgDn work for web forms but do not work for XML documents. For Windows forms, F7 toggles the designer between both views. (Shift-F7 is reserved for another feature entirely in VS.NET 2005.)

Going to a Line Number

Runtime exceptions usually provide you with line numbers. To go quickly to a line number in your text file, press Ctrl-G or just double-click the line number status bar at the bottom. In the small dialog box that appears, enter a line number to jump to.

If you type a line number that is out of the range of possible line numbers, the cursor jumps to the beginning or end of the file, respectively. A too-high number jumps to the end of the file, while a too-low number jumps to the beginning of the file.

Searching for a Word

Usually when you search for a word inside a file, you select Edit > Find and enter a term in the Find dialog box. But there are other ways to find a word. For instance, there is a combo box in the main toolbar right next to the configuration drop-down list. Enter or paste a word into this list and press Enter to immediately invoke the search function. Repeatedly pressing Enter in that drop-down list finds the next match. You can navigate to that combo box quickly by pressing Ctrl-D.

There is an even faster way to search for a word. Select the entire word, or place the cursor somewhere inside the word, and press Ctrl-F3. This invokes the same search function I described just previously. Repeatedly pressing Ctrl-F3 iterates over all matches.

Using either the combo box or the Ctrl-F3 shortcut applies the same search options as specified in the regular Find dialog box. So make sure you set those options correctly there first (for example, enabling Search Hidden Text to include all collapsed regions in the search area).

Finding and Highlighting Matching Tokens

Certain tokens always come in pairs. In C#, for example, the “{” token must be closed with a corresponding “}” token. Similarly, the compiler directive “#region” must have its corresponding “#endregion” directive. While navigating your code, you sometimes need to look for a corresponding token. You can do this very quickly by pressing Ctrl-]. This shortcut works only if the cursor is currently located next to either of these tokens, and it jumps to the corresponding one regardless of whether it is the opening or closing token.

If you want to highlight all the code between two matching tokens, press Ctrl-Shift-] to highlight the entire block and move the cursor to the opening token. This shortcut works only if the cursor is located next to either of these tokens (it does not work when you are located inside the region).

Note: This feature only works in C#.

Performing an Incremental Search

For those who are familiar with Emacs in UNIX, you probably know what an incremental search is. Invoked by pressing Ctrl-I, it allows you to find occurrences of a

search key as you type it one letter at a time. After each keystroke, VS.NET immediately highlights the next available occurrence that matches whatever you have typed so far. The more letters you type, the more likely is it that the found occurrence is indeed what you are seeking.

The beauty of an incremental search is that you do not need to enter the entire word to find a specific occurrence; you only need to type the minimum number of characters that would uniquely identify the word for which you are searching.

Once you are satisfied with a found result, press Escape to return to normal editing mode. If you are unsatisfied, press Ctrl-I repeatedly to find the next occurrence that matches your partial search key, or press Ctrl-Shift-I to find the previous matches. You can, of course, simply enter more letters to narrow the search further.

The biggest downside of incremental searches—which, in my opinion, makes them almost useless—is that they ignore collapsed regions, even if you have enabled Search Hidden Text in the Find dialog box.

Searching or Replacing with Regular Expressions or Wildcards

Anyone who has worked with regular expressions knows that despite the fact that they look extremely intimidating, they are extremely powerful tools to find complicated search keys and patterns. Regular expressions allow you to describe a searchable pattern in terms of wildcards, characters, and groups.

This is a built-in feature of VS.NET that many developers often overlook. When you bring up the Search or the Replace dialog box, by pressing either Ctrl-F or Ctrl-H, respectively, note that besides the regular options to refine your search, the last check box allows you to define your search based on regular expressions or wildcards.

In Regular Expression mode, specify the expression using a similar notation you are accustomed to with the `System.Text.RegularExpressions` namespace. If you are like me and always need a cheat-sheet when constructing regular expressions, you will appreciate the arrow button next to the Find What field. Click it to see a list of possible constructs that you can insert into your regular expression (see Figure 24).

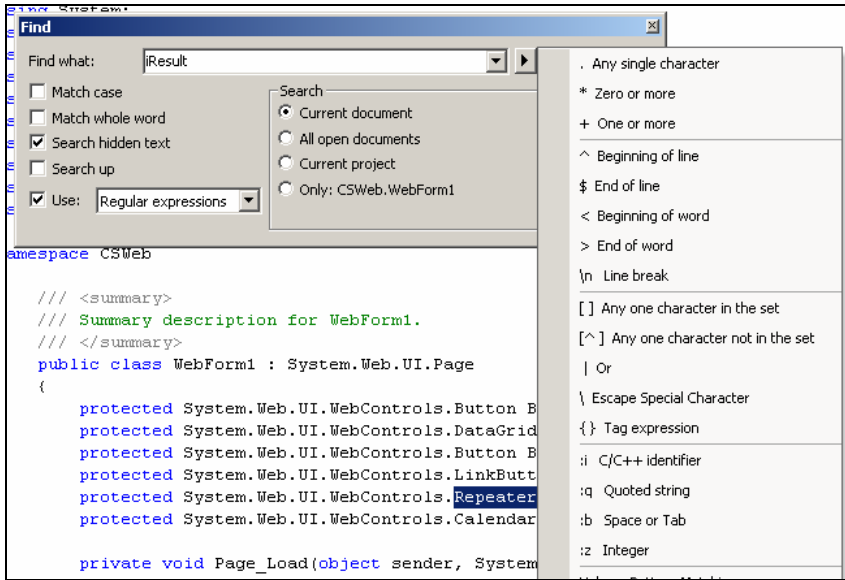


Figure 24 - Search with regular expressions

In Wildcard mode, construct your search pattern using the more commonly known wildcards from MS-DOS, such as “*” and “?”.

If used correctly, these two modes can be very helpful in refining your search algorithm or even as a test bed when developing programs based on regular expressions.

Performing a Global Search or Replace

Every code editor features a search or replace functionality but did you know that VS.NET has a global search and replace feature that spans entire projects and solutions? Press Ctrl-Shift-F or Ctrl-Shift-H to bring up the global search or global replace dialog box, respectively. This is similar to regular search and replace dialog boxes, except that it allows you to specify the scope of the search or replace action over multiple files.

You can perform a global search and replace in just the current document, the current project, the entire solution, or simply any open documents (see Figure 25). You can even filter which files you want to look in based on wildcards.

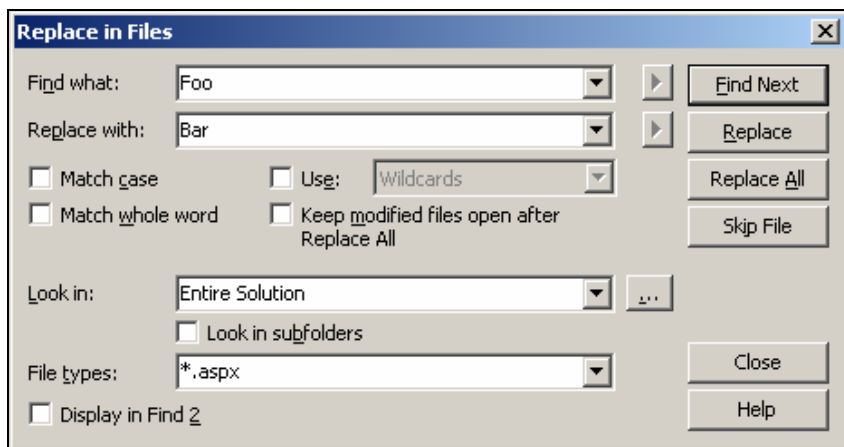


Figure 25 - Global Search and Replace dialog box

Once you start the search or replace action, VS.NET looks in all documents specified and modifies them if necessary. When using the global replace, you will also be prompted to leave modified documents open. This option allows you to undo the replace, because only open documents offer the undo feature. If you don't select that option, the global replace will automatically save the modified files to disk and thereby make this a permanent action.

You can immediately stop a global search or replace anytime by pressing Ctrl-Break.

After doing a search or replace action, you will be presented with a list of occurrences that have been found in the "Find Result" dialog box. As with most lists in VS.NET, you can iterate over this list by pressing F8 or navigate to one occurrence by double-clicking it. If that occurrence is currently located in a collapsed region, double-clicking the same find result in the list again automatically expands the region.

On a new find or replace action, VS.NET clears this window to fill the list with the new results. However, if you want to keep the previous results, check the Display in Find 2 option in the search dialog box to output the result in a second window. This allows you to tab between both result sets.

In VS.NET 2005, all find/replace functionalities are bundled in a single dialog box (see Figure 26), but you can still access the global find/replace functionalities using the drop-down list at the top. All shortcuts remain the same as before.

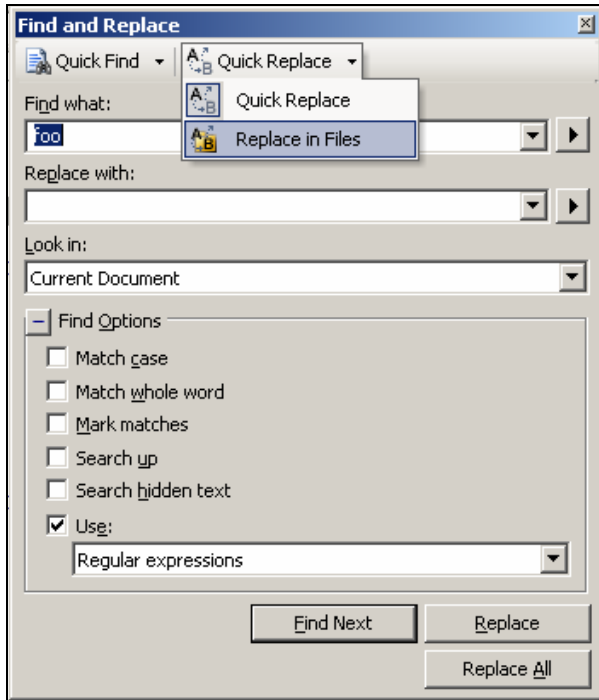


Figure 26 - Global search/replace in VS.NET 2005

Using Bookmarks

When you browse the web, you can save links to your favorite websites as bookmarks. These allow you to return quickly to a given web page. This ability can apply to programming as well.

As you code, there are critical sections of your programming that you return to frequently. Instead of scrolling repeatedly to these places, consider bookmarking those lines by clicking on the blue flag icon in Text Editor toolbar. (You can make this toolbar visible by right-clicking any existing toolbar and selecting Text Editor from the pop-up menu.) You can also place a bookmark by pressing Ctrl-K, Ctrl-K. Not only does this make a bookmark visible on the left side of the code, but you can now jump quickly among other bookmarks by clicking the appropriate flag buttons on the toolbar (see Figure 27) or by pressing Ctrl-K, Ctrl-P (for the previous bookmark) or Ctrl-K, Ctrl-N (for the next bookmark).

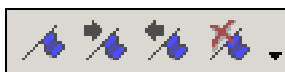


Figure 27 - Create, Next, Previous, and Clear Bookmark buttons

You can clear all bookmarks with the Clear Flag icon or by pressing Ctrl-K, Ctrl-L.

The Find dialog box in VS.NET allows you to bookmark all matching occurrences by clicking the Mark All button.

VS.NET 2005 adds much support for bookmarks. For one, you now have the option of moving to the next or previous bookmark within the same file, using two buttons on the bookmark toolbar (see Figure 28).



Figure 28 - Move to bookmarks in the same file in VS.NET 2005

You can also name your bookmarks. First press Ctrl-K, Ctrl-W to open a new Bookmarks window (also accessible by selecting View > Other Windows >Bookmark Window). This displays all the bookmarks that you have created (see Figure 29). Similar to the Task List, you can double-click a bookmark to jump to its location. You can also rename your bookmark by pressing F2 or using the Rename context menu item when you right-click the bookmark. In addition, you can create folders to categorize your bookmarks. There are also buttons to help you jump to the next or previous bookmark within the same folder.

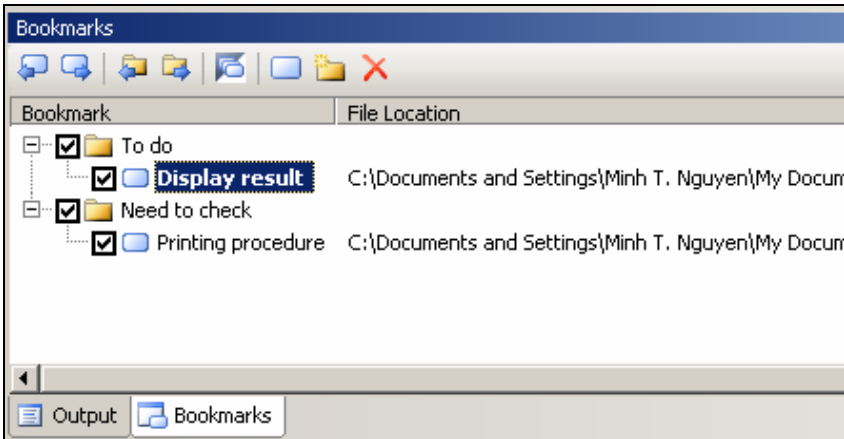


Figure 29 - Manage your bookmarks in the Bookmarks window

The Bookmarks window shows check boxes next to each folder and bookmark. This allows you to disable a bookmark without actually deleting it. Disabled bookmarks are skipped when you use any of the buttons or shortcuts to navigate your bookmarks.

Going to the Definition of a Method

When you are presented with the method invocation of a class, you most likely want to see the source of the method as well. Anyone using Visual Basic 6 appreciates the Shift-F4 shortcut that allows you to jump instantly to the definition of any method. This has been implemented in VS.NET through a context menu. By right-clicking the method inside the text editor and choosing Go to Definition from the pop-up menu (see Figure 30), VS.NET jumps to the source of that method. The default shortcut assigned to this feature is F12.

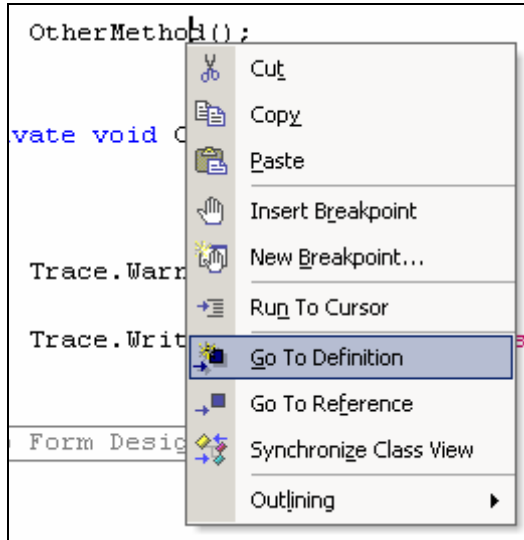


Figure 30 - Go to the definition of a method

When you use this feature for the first time on a project you just opened, there is a significant time delay before it finds the method source. I assume that VS.NET reflects on all referenced assemblies to build its own method-to-line mapping. This preparation is only needed once; subsequent usage of this feature—even on a different method—is pretty instant.

Using Browser-Like Navigation

VS.NET is equipped with browser-like “back” and “forward” buttons in the IDE that allow you to review the most recent cursor locations. The Navigate-Backward and Navigate-Forward buttons are usually located to the right of the Undo and Redo buttons (see top left of Figure 31). You can also access them in the View menu.

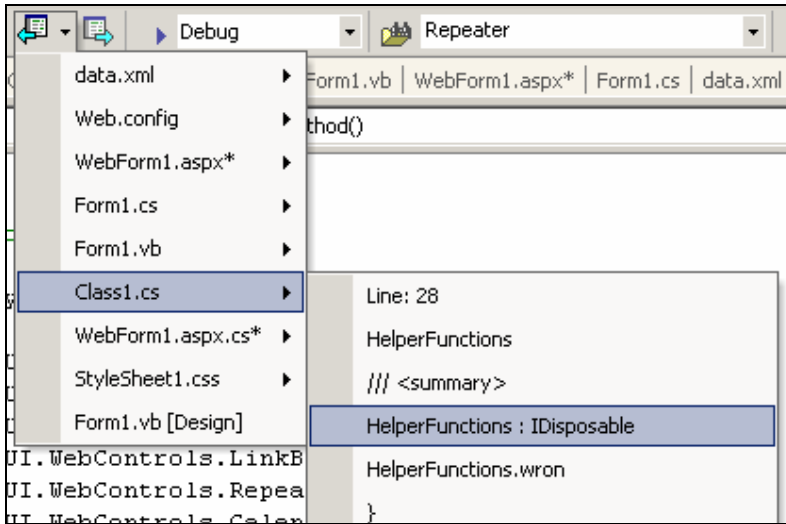


Figure 31 - Browser-like navigation buttons

After using the Go To Definition feature or after switching arbitrarily to another file or even just jumping between different line numbers of the same file, you can jump back to the last edit location by simply clicking the Navigate-Backward button. As with common web browsers, VS.NET keeps a history of your recently accessed locations. The Navigate-Back button also has a drop-down menu, so you can control to which file you jump back to. (The Navigate-Forward does not have this list.)

These Navigate buttons have pre-assigned shortcuts. Navigate back by pressing Ctrl-Hyphen and navigate forward by pressing Ctrl-Shift-Hyphen.

Inserting External Text File

When developers insert code fragments from an entire file, they tend to open the file in Notepad and copy the code from there. Another way to do this is to select Edit > Insert File as Text from within the code editor. There is really nothing fancy about this, except that it saves you the step of opening and closing Notepad.

Inserting JavaScript Tags

Even though the ASP.NET architecture is very powerful, you still need to rely on JavaScript for client-side scripting. When writing JavaScript code in an ASP.NET file, you have to embed the code in the following tags:

```
<script language="JavaScript"> </script>
```

Additionally, web developers tend to place `<!-- //-->` comment delimiters inside the script tag (see Figure 32) to prevent browsers that do not understand JavaScript code from crashing—because everything between `<!--` and `//-->` is interpreted as HTML comments.

```
        <script language="javascript">
<!--
//-->
</script>
```

Figure 32 - Inserted client-side script

If you do this often, you will appreciate that VS.NET inserts these four lines of code for you when you select `Edit > Insert Script Block > Client-Side`. This function is only enabled when you are in the HTML view of your ASP.NET page.

As you might guess, there exists an equivalent `Insert Script Block > Server-side` command as well, which allows you to insert server-side scripts in the ASP.NET file for those who do not use the code-behind model. Unfortunately, this feature is gone in VS.NET 2005.

Outlining HTML/Form Hierarchy

The document outline is a window that displays a hierarchical representation of your web form in tree view form, listing all major HTML items, web controls, and tags (see Figure 33). You can bring up this document outline window by selecting `View > Other Windows > Document Outline` or by pressing `Ctrl-Alt-T`. This document outline is usually updated automatically as you enter new or modify existing HTML code, but you can also force synchronization by selecting `View > Synchronize Document Outline`. Single-clicking any item in this tree automatically positions the cursor to the appropriate HTML tag. The context menu that appears when you right-click any item in the tree allows you even to cut or delete the item.

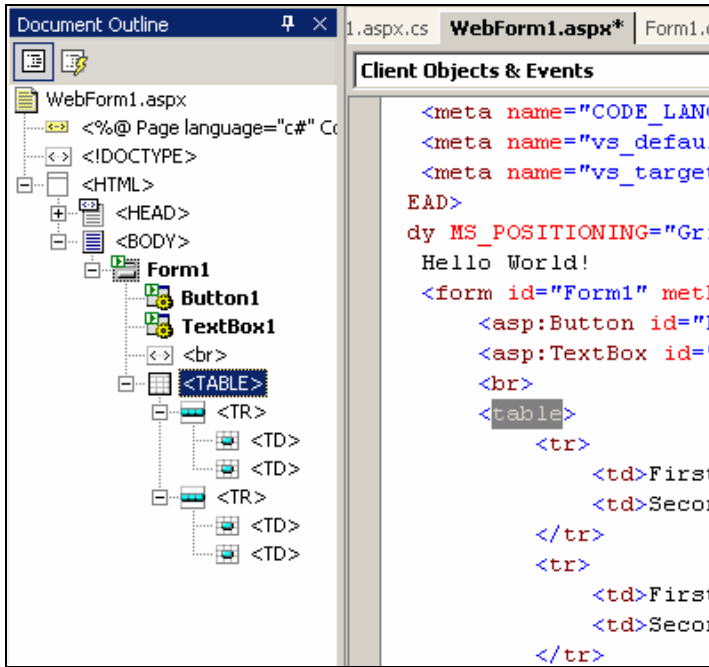


Figure 33 - Document outline displays your web form in hierarchical form

Starting with VS.NET 2005, the document outline also works with Windows forms. Here the tree view displays both the control types and control instance names. Through the right-click context menu you can even rename your controls in this window.

Chapter 2: Exploring the IDE

Visual Studio .NET is an extremely customizable IDE that is loaded with features. No screen real estate is wasted to give you quick access to commonly used commands and activities and to control and modify your project and solutions. The tips in this chapter cover everything from the Solution Explorer to window positioning, from managing macros and modifying menu items to other lesser-known tricks that are highly useful in navigating inside the IDE.

Opening with Default Action

Double-clicking any file in the Solution Explorer opens the file in the default editor. You can change the default editor for a specific file type. By right-clicking the file and choosing Open With from the pop-up menu, you can choose the exact code editor and mode in which to open the file. This includes HTML/XML, form editor, source code editor, binary editor, and so on. In this same dialog box, select the default editor for the selected file extension by clicking the Set as Default button.

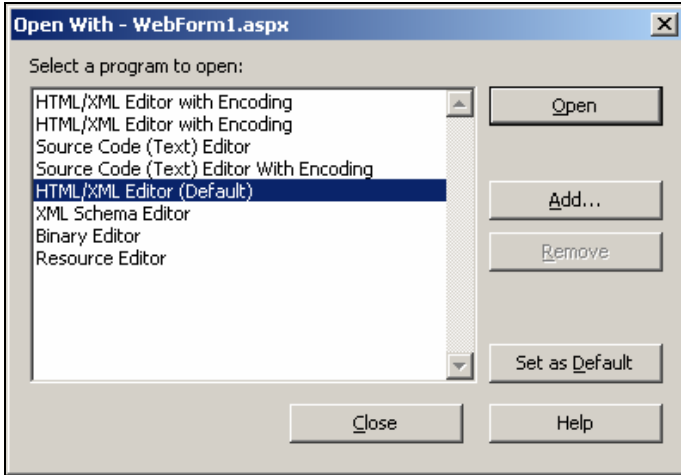


Figure 34 - Choose a default action to open files

The Open With dialog box allows you to customize the default View mode, which is especially useful when selecting web forms or Windows forms, as you might prefer the Code view over the Designer view (or vice versa).

Showing Extra Files

By default the Solution Explorer only shows you files that are included in the particular solution. If you attempt to add existing files to your solution by copying the file into your project folder, you will see that they do not automatically appear in the Solution Explorer. To make them appear, you need to click the Show Extra Files button in the Solution Explorer's toolbar. This button displays all files (and grays out files that are not part of the solution).

You can now include the file into your solution (to make it part of the source control repository as well) by right-clicking the file and choosing Include in Project from the pop-up menu. By the way, you can also include files by dragging them from Windows Explorer directly into your Solution Explorer.

In addition, by showing all extra files, you also have the ability to see code-behind files for your ASP.NET pages. By double-clicking the code-behind file, you will always open the ASP.NET in Code view.

Setting Project Dependencies

In a large solution with many projects and custom build events, it is often necessary to control the build order for your projects. VS.NET is usually smart enough to figure out which project needs to be built first by analyzing the references of each one. Naturally, the first project built is the one referenced first. This algorithm is based on the fact that you have set project references for your projects.

However, there will be times when you have to instruct VS.NET to compile a certain project before another one without having project references. In order to do this, right-click your project that needs to be built last and choose Project Dependencies from the pop-up menu. Here you can set manual dependencies on other projects by check-marking them, essentially enforcing that those checked projects will be built before the current project (see Figure 35). A drop-down menu for the current project allows you to switch to another project's dependencies.

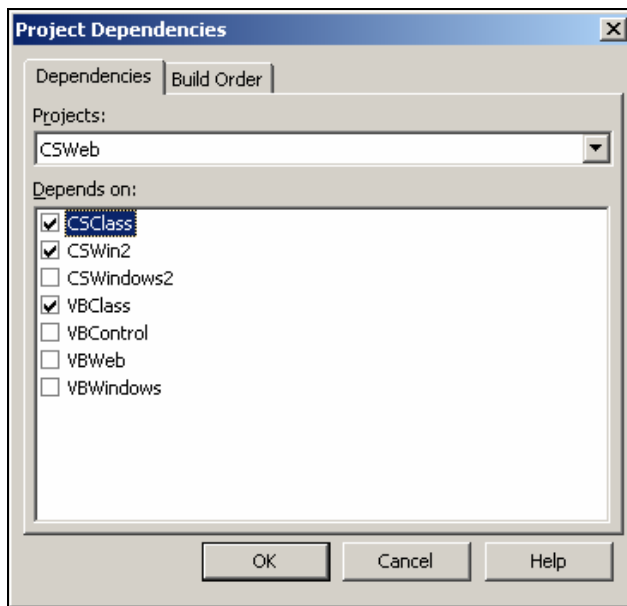


Figure 35 - Setting project dependencies manually

VS.NET prevents you from creating circular references or modifying dependencies that resulted from adding project references. At any given time, you can verify the build order by clicking the read-only Build Order tab.

In VS.NET 2005, the Project Dependencies context menu item in the Solution Explorer does not exist for web applications, so you have to select Websites > Project Dependencies instead.

Embedding Files As Resources

Embedding files as resources allows you to embed any given file directly into your produced assembly. Let's say you need to display a company logo on your Windows application. You could produce a regular Windows assembly and link to an external image that you ship along with your application. Another approach is to embed the image right into the assembly you produce. Not only will you not have to ship the external image but, more importantly, you prevent the possibility of these two files becoming separated.

Before you can embed a file as a resource, it has to be included in your solution. Then you can select the file in the Solution Explorer and change the Build Action property in the Properties window. The build action tells the compiler what it needs to do with the specified file. If you select the Embedded Resource build action, the actual bytes of the file will be stored inside the produced assembly (regardless of whether this is an EXE or a DLL).

At runtime, you can then extract the bytes using the following code:

```
Assembly oAssembly =  
System.Reflection.Assembly.GetExecutingAssembly();  
  
Stream streamOfBytes =  
oAssembly.GetManifestResourceStream("filename.typ");
```

After this retrieves the stream of bytes from the given embedded resource, you have to convert those bytes back into the original file type (using `Image.FromStream()`, for instance, to convert it back into a picture). Notice how this code is orthogonal to the file type being embedded as a resource. As such, you can embed any file type: sound and movie files, PDF files, or even another assembly.

Changing the Font Size of IDE Windows for Demos

Whenever you demo VS.NET or your code, you often need to increase the font size of the text editor so that everyone in the audience can see what you're doing. Most developers have probably already figured out that you can easily increase the font size by selecting Tools > Options > Environment > Font and Colors > Size. This works out great, except that the text in the Output windows, Solution Explorer, Class view, Macro Explorer, or in the file tab titles are still very hard to read.

You can control the size of the text in these elements as well (see Figure 36). In the same settings window, the first drop-down list reads Show Settings For. Change it to read Dialogs and Tools Windows. By setting the font and the size here as well, you control the format of the text elements of the majority of the IDE windows. The changes come into full effect after you restart the IDE. If you want to increase the font of the Output window, change Show Settings For to Text Output Tools Windows.

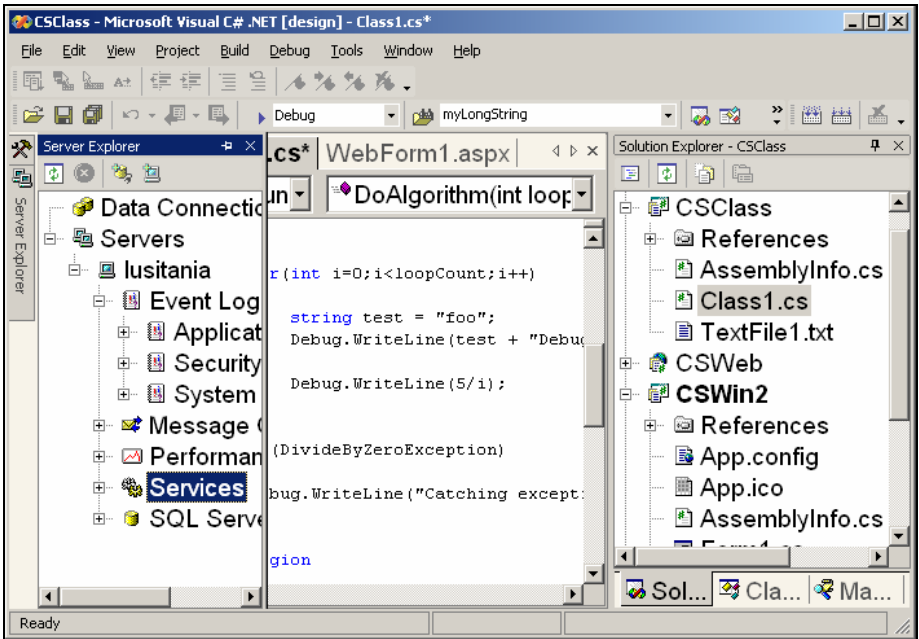


Figure 36 - Change the font size of all IDE windows

If you ever want to reset any of these settings to their default installation values, click the Use Defaults button. This button applies only to the currently selected item in the Show Settings For drop-down list, however, so you may have to repeat this step for every setting if that is what you want to do.

Dragging Files to Obtain a Full Path

Did you know you can drag files from your Solution Explorer directly into your code? If you do this in a source code file, it will simply insert the full path to the selected file into your code. In the HTML editor, VS.NET is even smarter. If you drag another ASP.NET file into the HTML editor, VS.NET inserts the path with anchor tags around it, essentially building an HTML link. If you drag a picture image, for instance, VS.NET inserts the path with the <image> tag.

With VS.NET 2002 and 2003, unfortunately, the link created for the HTML tags are absolute URLs, so most likely you will have to remove the “http://localhost” or “file://” prefix from the URL before deploying your web application. VS.NET 2005 generates only relative URLs.

Moving Any Window Around

Every window in VS.NET is movable, resizable, and dockable: the Solution Explorer or Macro Explorer; the Properties, Task, and Output windows; and even your Toolbox, Server Explorer, and Find/Replace windows. Move any window in VS.NET by dragging the title bar to the desired position. As you drag a window close to a dockable region (such as tabs or near another window frame), a gray outline appears, allowing you to preview the result before dropping the window. Dock and undock windows by double-clicking the title bar. You can also move the order of tabs in your tab windows. This includes the files tabs at the top of your editor.

While the ability to control window positioning gives VS.NET enormous flexibility, it can also be troublesome because the preview outlines are too confusing to make this an intuitive interface. Too often developers struggle to position a single window into a desired location. The VS.NET 2005 version is somewhat more intuitive; it displays “hot spots” that indicate where a given window will be docked when you drop it over that area.

If you ever find yourself messing too much with windows positions, you can always reset all windows positions to their installation defaults by selecting Tools > Options > Environment > General > Reset Windows Layout. With VS.NET 2005, you can also reset by going to Window > Reset Windows Layout.

One neat aspect of moving windows around is the ability to create a split screen. Simply drag the tab of any open file and move it to the right of your editor (to the left of where the Solution Explorer usually resides). This docks your selected file to the right and splits the editor into two vertical screens, complete with their own set of file tabs (see Figure 37). When you are done with editing your files in vertical split mode, simply close the second set by clicking the small X at the top right, or drag the file tabs back to the left along with the other files.

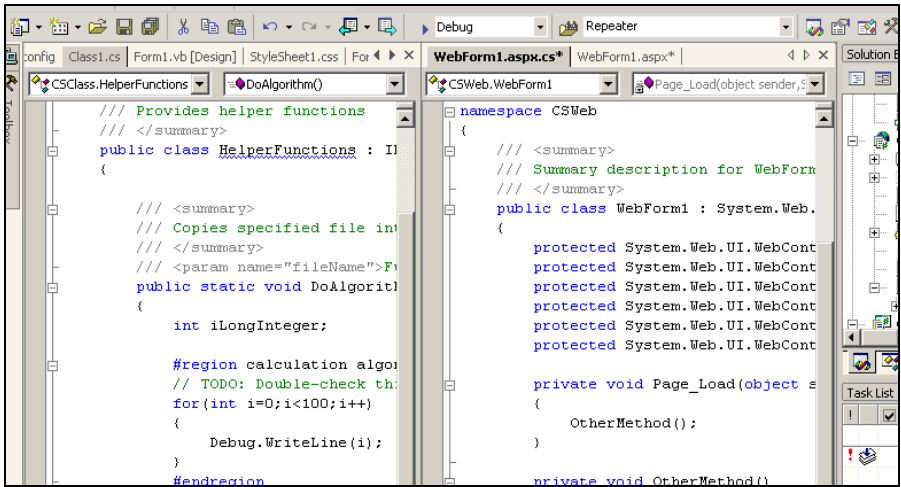


Figure 37 - Move a window to create a vertical split screen

This trick also works to create a horizontal split screen. Simply drag a file tab to the bottom of your editor.

Creating Split Screens in the Same File

The “Moving Any Window Around” trick described previously shows how to create split screens so you can see two files next to each other. What if you want to create a split screen to see two locations of the same file? You could select Window > Split to generate the horizontal divider but there is a faster way.

Right above the vertical scrollbar of the main editor is a very thin, short, rectangular-shaped divider (see Figure 38). If you mouse over that divider, you’ll see that the mouse icon changes to the divider icon. Drag the divider down to the center of the screen to create the split screen (see Figure 39).

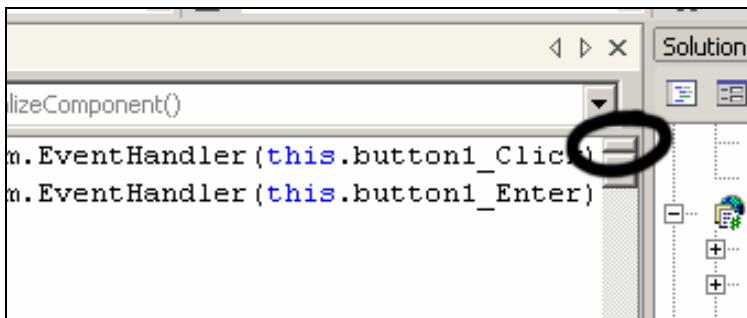


Figure 38 - Drag horizontal divider down to create a split screen

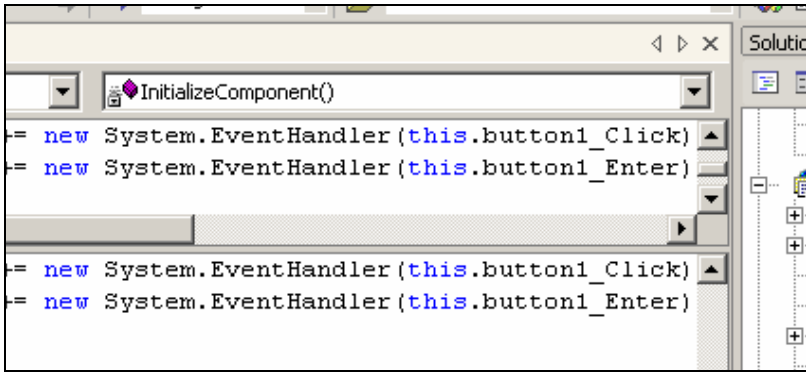


Figure 39 - Split screen in the same file

Once you are done, simply move the divider back to the top of your editor window.

Customizing the VS.NET Menu and Toolbars

You can customize the VS.NET menu by adding and removing commands as well as reordering them. To do so, select `Tools > Customize`. With the `Customize` dialog box open, navigate back to the VS.NET menu. The menu now does not react to left mouse-click events but, instead, shows context menus when you right-click the menu items. Here you can rename, edit, and delete menu items; drag menu items around; or even create your own cascading menu groups.

You can also manage the icons for each menu item by right-clicking the item and selecting `Choose Button Image` from the pop-up menu. If you are not satisfied with the icons in the selection, you can copy icons from other menu items to your newly created menu ones. Simply right-click a menu item with a nice icon, choose `Copy Button Image` from the pop-up menu, and then use the same mechanism to `Paste Button Image` to the menu item you want to modify. To add other commands to a menu, drag them from the `Command` tab directly into the VS.NET menu.

In addition to directly modifying the VS.NET menu items as long as the `Customize` dialog box is open, VS.NET 2005 adds a complete new GUI to modify the menu. The new GUI appears when you select `Tools > Customize > Rearrange Commands`. Here you can move, add, and delete menu items as well as toolbar buttons (see Figure 40).

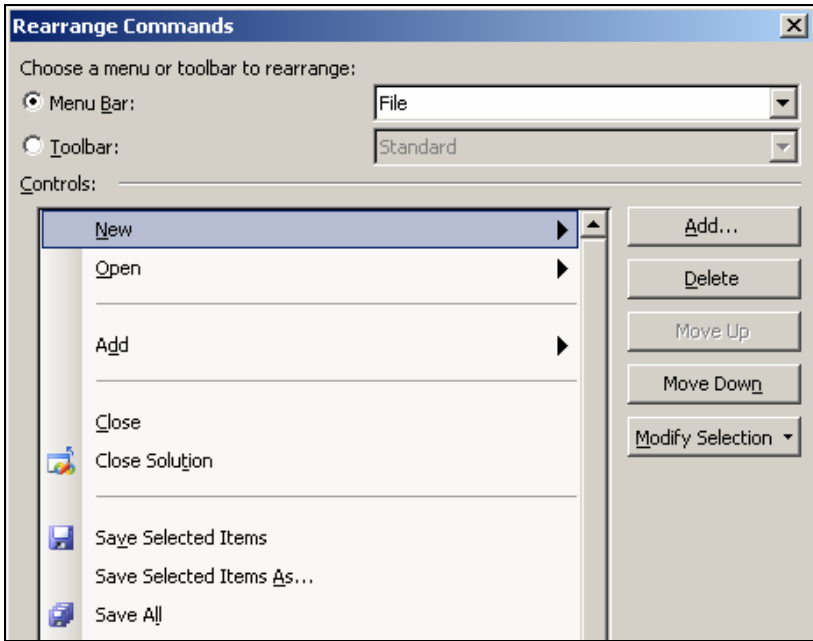


Figure 40 - Customize menu through Rearrange Commands

Adding External Programs to the VS.NET Menu

If you often use third-party programs while developing .NET code, you will appreciate the fact that you can create shortcuts to those programs in the VS.NET menu. For example, if you use Enterprise Manager, Query Analyzer, ILDASM, Reflector for .NET, or other programs frequently, consider selecting Tools > External Tools to add external programs to the menu (see Figure 41). Add new programs to the list that appears under the Tools menu, or specify their parameters and working directory.

You can even set environment variables, such as \$(ItemFileName) for the currently opened file or \$(ProjectDir) for the current project directory, in those fields. If you don't know which environment variables exist, click the expand buttons to the right of the fields.

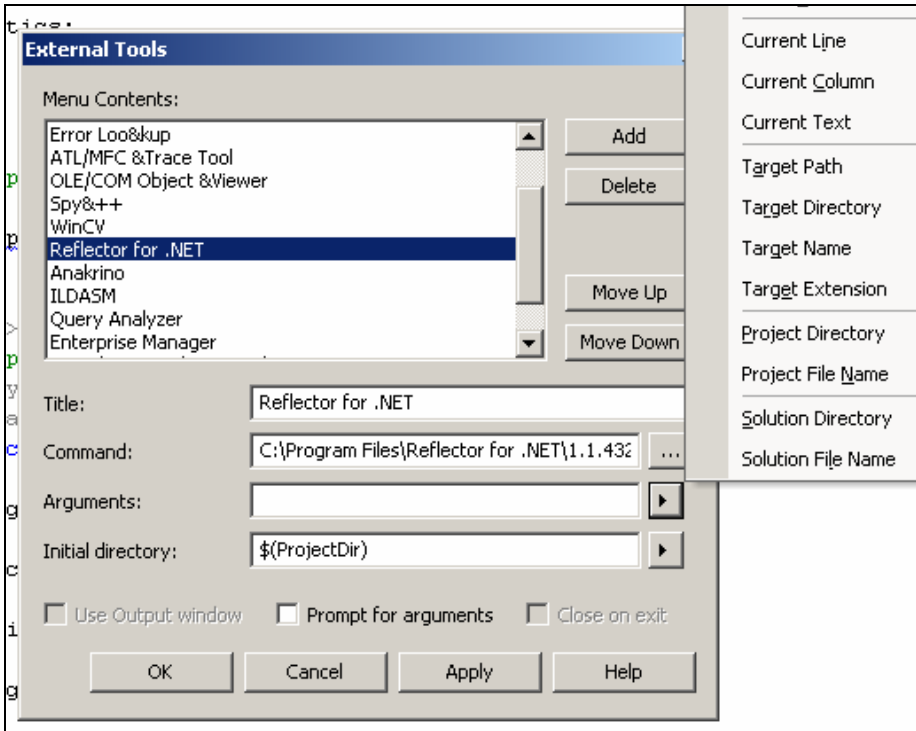


Figure 41 - Adding external tools to the VS.NET menu

Now your favorite programs will appear in the Tools menu, giving you quick and easy access to them. To customize the icon for your new menu item, simply use any trick described in the previous tip.

You can also assign shortcuts to these external tools by selecting Tools > Options > Environment > Keyboard and looking for the many Tools.ExternalCommands commands. Select one and assign the shortcut right there.

Dragging Files from Windows Explorer into VS.NET

Visual Studio .NET fully supports file drag (and drop) actions. You can drag files from Windows Explorer directly into VS.NET. If you drop them in the Solution Explorer under a project, it will first be copied into the same directory as the project and then included into the project. If you drag them into the code editor, VS.NET will either start the default external viewer (for example, Adobe Acrobat for PDF files) or display the file's contents inside VS.NET if it's a file type that it understands.

The best way to drag files from Windows Explorer into VS.NET if you don't have enough screen space is to drag the file into the Windows taskbar at the bottom of your screen and pause for a few seconds over the taskbar for VS.NET. The pause brings VS.NET into focus, so you can drop the file into the appropriate location. (This is a Windows trick, not a VS.NET trick.)

Accessing the Command Window

The Command window is a little, but very powerful, window in VS.NET that allows you to invoke any action you can perform in VS.NET. It has an entire object model to interact with your files, folders, project, macros, and any other IDE object. View the Command window by selecting View > Other Windows > Command Window or by pressing Ctrl-Alt-A. Aside from "Command Window" in the title bar, you can recognize it because it shows a command prompt in the form of ">" (see Figure 42).

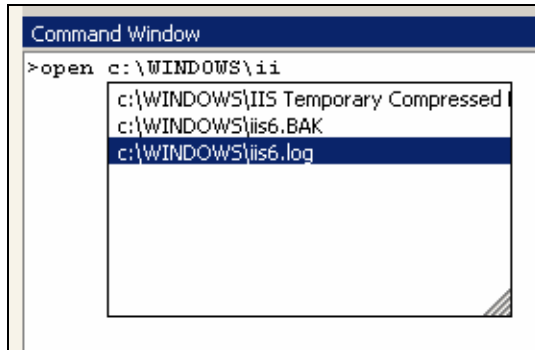


Figure 42 - Opening files through the Command window

Here is where you can invoke VS.NET-specific commands. "CloseAll," for instance, closes all currently opened windows. "Build.BuildSolution," as you might expect, builds the current solution. If you wonder where these commands come from, go to Tools > Options > Environment > Keyboard. There you can see all the available commands in VS.NET.

In most cases, you invoke these commands by typing them in the following form:

```
Category.CommandName
```

Commands are grouped logically into categories, allowing you intuitively to find the correct command name of any specific action.

It helps that IntelliSense works in the Command window. You can resize the IntelliSense drop-down menu to your desired size. Additionally, if you type the **open** command, you will see that IntelliSense inspects your file system to display possible files and folders on your hard drive so you don't have to jump to Windows Explorer to locate the file.

Aliasing Your Favorite Commands

If you find yourself executing a command in the Command window many times, consider creating an alias for this. The syntax of an alias is the following:

```
alias [NewName] [OldName] [Arguments]
```

For example, to create an alias that opens your design document, type the following:

```
alias DesignDoc open "c:\Your Folder\Your Design Document.doc"
```

The good thing about aliases is that they are permanent. Therefore, you can use them even if you restart VS.NET. Simply type **alias** to view all aliases. Delete an alias as follows:

```
alias -d DesignDoc
```

Switching to Immediate Mode from the Command Window

Another nice feature of the Command window is the ability to switch quickly to the Immediate window and vice versa. The Immediate window is where you type variable names when debugging your program to get the variable value in return (see “Using the Immediate Window to Display Variables and Execute Methods” in Chapter 3).

Switch from the Command window to the Immediate window by typing **immed**. This command switches the Command window into “immediate mode.” You will recognize the Immediate window (besides its name in the title bar) by the lack of the “>” command prompt.

During your immediate session, you can run a single command at any time by prefixing it with the command prompt (“>”). If you want to switch back permanently to the Command window, you have to issue the **cmd** command—again, prefixing this with the command prompt so that VS.NET won’t think that you are querying a nonexistent variable named “cmd.”

Using the Command Window in Find a Drop-Down List

If the previous tips have not convinced you enough to use the Command window—because you first have to open the Command window either through the menu or the previously mentioned shortcut—you will like the fact that VS.NET has another, even faster way to issue commands.

Press Ctrl-/ to turn the Find drop-down list at the top of the code editor into a “mini-Command window” with the command prompt “>” already displayed (see Figure 43). This mini-Command window contains the same full features, including IntelliSense, and all the aliases you created before.

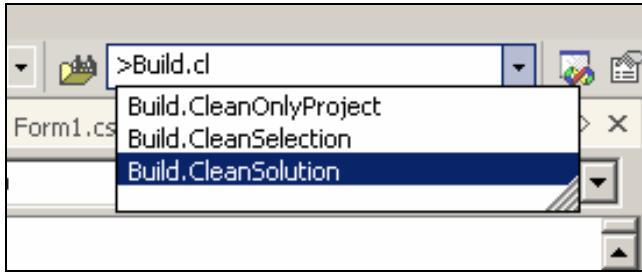


Figure 43 - Using Find drop-down list as a mini Command window

Using the Built-in Web Browser

Most ASP.NET developers are aware of the built-in Visual Studio .NET web browser if they debug a web application internally by setting the default browser to the internal one. Set the default browser by right-clicking any .aspx file and choosing Browse With from the pop-up menu. Specify whether a new instance of Internet Explorer should be spawned or whether the internal web browser should be used. The internal browser appears as another file tab inside VS.NET.

You can also use the internal web browser for non-debugging activities. Press Ctrl-Alt-R to open the internal web browser, complete with an Internet Explorer-like toolbar at the top where you can navigate to any URL, interact with your Favorites folder, retrieve or save URLs, or even access your browser history (see Figure 44). You can call up the Favorites folder window by pressing Ctrl-Alt-F.

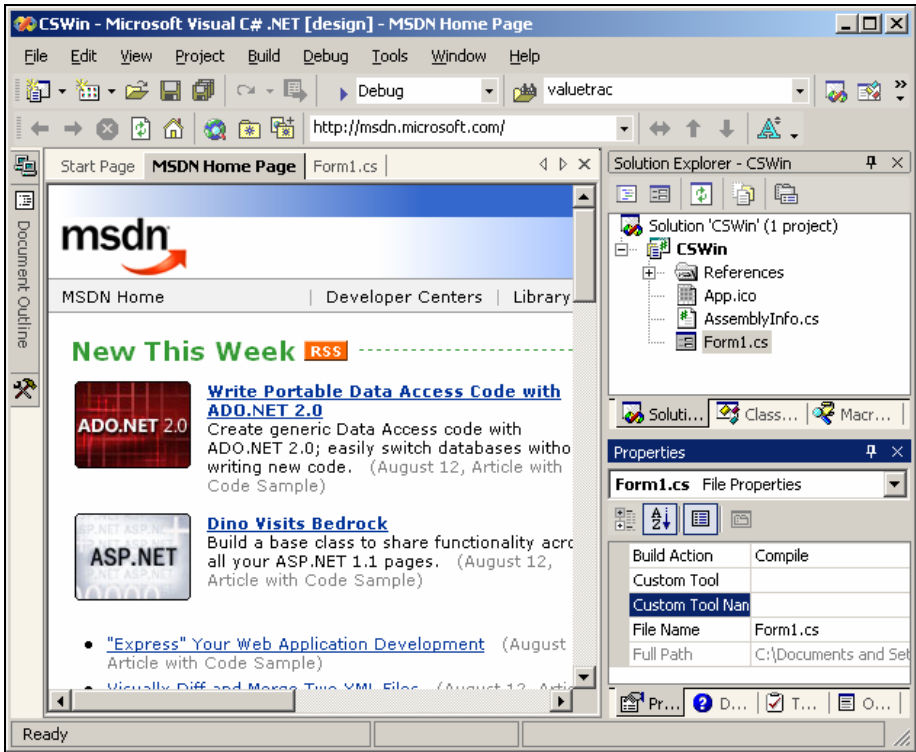


Figure 44 - Browsing the web from inside VS.NET

Using Full-Screen Mode

As in most other Microsoft products, full-screen mode allows you to hide virtually everything except the main editor, so that the entire screen shows the main view. Enter full-screen mode by going to View > Full Screen or pressing Ctrl-Shift-Enter. The main menu is still visible at top, and a floating button that closes full-screen mode is also available. Developers can hide the Close Full Screen mode button—in which case you must memorize the keyboard shortcut that returns to normal mode or select View > Full Screen again.

Full-screen mode is available for any view, including the HTML, Designer, and XML views.

Copying the Fully Qualified Name of a Class

The Class view (select View > Class View or press Ctrl-Shift-C) is a hierarchical view of all your classes and namespaces in your solution. You can drill down to any class and its members and navigate to the member definitions by double-clicking them. Another useful feature allows you to extract the full namespace of any class or member. Highlight the class or the class member and press Ctrl-C. This copies the complete namespace of the selected item to the Clipboard. This comes in handy when you have a complex or deep namespace structure.

If you intend to paste the namespace into the VS.NET code editor, there is no need to copy it to the Clipboard first. Simply drag a class or member of a class from the Class view directly into your code and watch VS.NET paste the complete namespace and member name there.

Recording and Replaying a Temporary Macro

Have you ever encountered a situation while writing code where you repeat a series of keystrokes over and over again? Maybe you are in the middle of reformatting several dozen lines of code and you press the exact same sequence of keystrokes repeatedly for each line? If so, consider using a temporary macro. Think of macros as scripts that allow you to interact programmatically with the VS.NET environment to automate tasks. Fortunately, macros are written in VB.NET so you don't have to learn another language to use them. In fact, you don't even have to learn how to write a macro to use one. Just record a macro and let VS.NET write the code for you.

Select Tools > Macros > Record Temporary Macros or press Ctrl-Shift-R. The floating Recorder toolbar appears (see Figure 45) and records every single keystroke, menu event, and action you perform, and then writes them into a temporary macro using VB.NET. Perform the steps you need to complete a single iteration of your repetitious task. When you are done, click the Stop Recording button (it looks like the VCR record button) on the floating toolbar to stop the recording (or press Ctrl-Shift-R again).



Figure 45 - Pausing, stopping, and canceling the current macro recording session

You can replay the temporary macro as often as you wish by selecting Tools > Macros > Run Temporary Macro or pressing Ctrl-Shift-P each time.

Saving, Editing, and Debugging Macros

Surprisingly enough, the temporary macro created in the previous tip survives restarts of VS.NET because it is actually saved as a macro in your Macro Explorer. The next time you record a new temporary macro, however, it will be overwritten.

If you open up the Macro Explorer window (select View > Other Windows > Macro Explorer or press Alt-F8), you will see that your temporary macro is saved under the RecordingModule item of the MyMacros package. Double-clicking any listed macro will execute it. VS.NET ships with hundreds of useful macros found in the Samples macro package.

By right-clicking any macro and choosing Edit from the pop-up menu, you can examine the VB.NET code that defines it. Unfortunately, you cannot write macros in C# or any other .NET language.

However, you can debug a macro. Set a breakpoint and invoke the macro. The macro editor will break at the set breakpoint, giving you the same rich environment to debug the macro as you would a regular .NET application, including the ability to move the cursor around and watch variable values. In fact, another instance of VS.NET is actually fired just to debug your macro.

The default location of the physical files of saved macros is the My Documents\Visual Studio Projects\VSMacros.XX directory, where XX is your version of VS.NET.

Assigning Shortcuts and Menu Items to Macros

After you develop your wonderful macro, one way to invoke it is by double-clicking it in the Macro Explorer. A faster way is to provide it with a shortcut or menu item. To create a shortcut, select Tools > Options > Environment > Keyboard. Here you can find a list of all possible VS.NET commands (see Figure 46). Among them will be your macro listed under Macros.MyMacros.YourModule.YourMacroMethod.

Once you select your macro, you can assign it a shortcut key by typing your desired shortcut in the Press Shortcut Key(s) text box and confirm it by clicking Assign. The drop-down list on the left allows you to specify the scope of this shortcut (that is, determine in which editor this shortcut will be available).

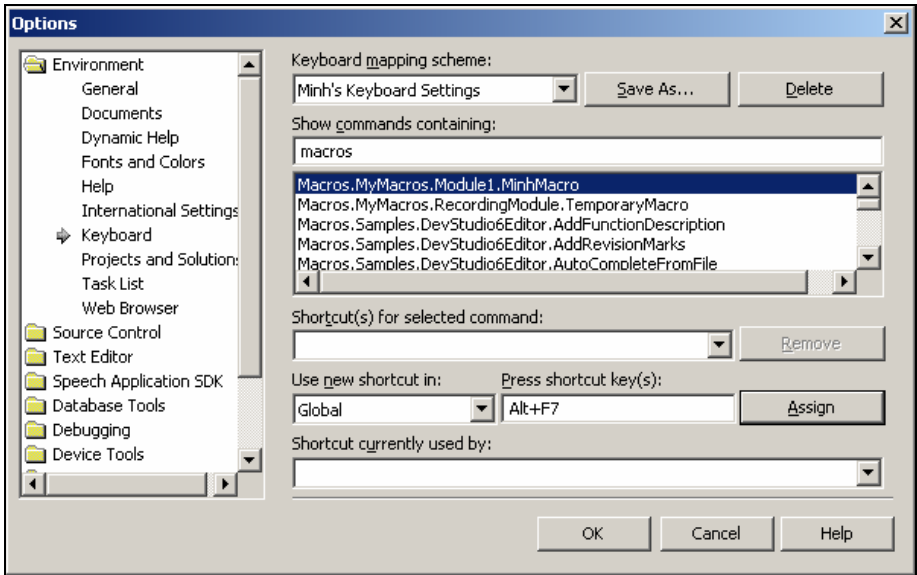


Figure 46 - Assigning keyboard shortcuts to macros

To create a button for your macro, select Tools > Customize. (You can also get to the Customize dialog box by double-clicking an empty space in the top area where the button bars are located.) On the Commands tab, select the Macros category. Now simply drag your macro from the Commands list to any toolbar to create a button (see Figure 47) or to any menu to create a menu item. As with any other button or menu command, you can right-click it to change its label or icon (including copying an icon from another button or menu item).

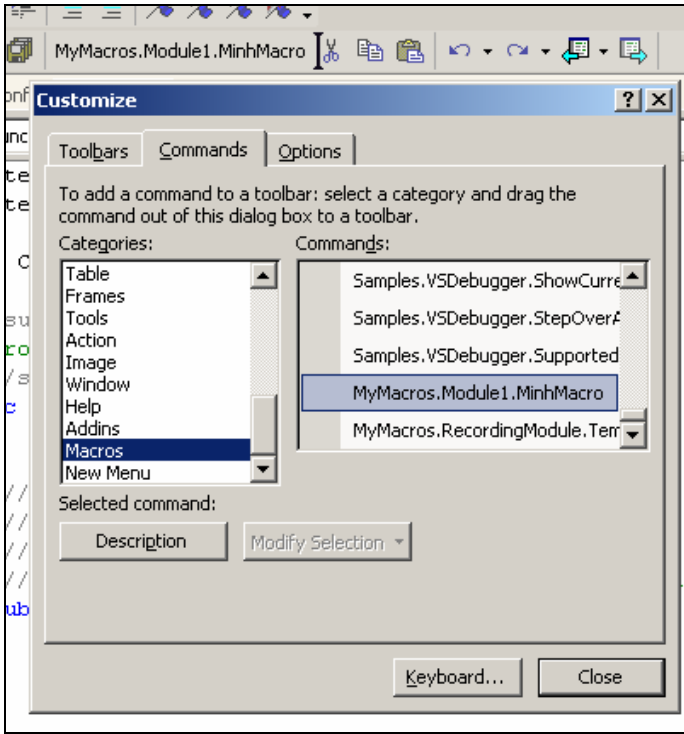


Figure 47 - Moving a macro to a toolbar

Changing Properties of Several Controls

When you design your Windows forms, you use the Properties window to modify a control's behavior and appearance. The Properties window, however, is adaptable when you select several controls at the same time. By selecting a series of controls (either by holding down Ctrl or Shift when selecting controls or by drawing a selection rectangle with your mouse), the Properties window automatically displays the properties that are common to all of the selected controls. With all controls selected, any change you make in the Properties window affects all selected controls.

This comes in handy, for instance, after you drag a series of text boxes from the Toolbox onto your form and want to get rid of the default "TextBox1," "TextBox2," etc. values. Select all the text boxes, change the Text value to a single space by pressing Spacebar, and then change it back to an empty string by pressing Delete. (You have to do this twice because the initial values of each text box differ originally, so the Text property displays an empty string as the "common value".) This deletes the default text in all of them.

Locking Controls

When laying out your windows controls on your Windows forms, you can easily move the controls around or create event handlers by simple dragging and double-clicking. However, this simplicity has its shortfall, too. You can move things around accidentally very easily. This is not optimal if you already finished designing your Windows forms.

In order to prevent this from happening, you can lock your form. While in the Designer view, right-click anywhere on your form and choose Lock Controls from the pop-up menu (see Figure 48). Although you can still add event handlers and modify a control's appearance, you can no longer move or resize a control accidentally. A thin, black outline appears around each selected control to indicate that it is locked and unmovable. If you want to return to the Designer view, right-click your form and choose Lock Controls from the pop-up menu again.

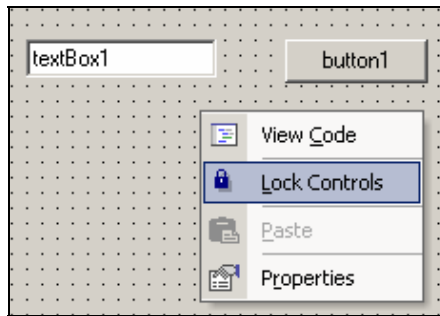


Figure 48 - Locking windows controls in the Designer view

Toggling the Description in the Properties Window

The Properties window not only displays all properties of a selected control. The Description pane at the bottom briefly describes the active property. As you select different properties, the Description box informs you what the selected property does. You can resize that panel if you want. At first, it serves a useful purpose. But as you become an experienced .NET developer, it just takes up valuable screen real estate. Turn it off by right-clicking the property name and choosing Description from the pop-up menu. You can always turn it on later the same way.

Change Drop-Down List Values in the Properties Window

Whenever a property only accepts a finite set of values, the value field becomes a drop-down list, from which you make your choice. For instance, the `FormBorderStyle` property of a Windows form only accepts `None`, `FixedSingle`, `Fixed3D`, `FixedDialog`, `Sizable`, `FixedToolWindow`, and `SizableToolWindow`. To select the appropriate item, you have to open the drop-down list and select the style you want.

Anytime you have a drop-down list in the Properties window, you can iterate over the list more quickly by simply double-clicking the property or its corresponding drop-down list. Without expanding the list first, double-clicking it sets the value to the next available item in the list (or to the first item if the current value is the last one).

I find this trick extremely useful when switching Boolean values because a double-click changes the value quickly from `True` to `False`, or vice versa.

Adding and Removing Event Handlers Through the IDE

Adding default handlers through the IDE is really easy. In most cases, all you do is double-click a control. This creates the necessary code for the default event handler. What about non-default events? How do you add them? More importantly, how do you remove event handlers nicely? In `C#`, removing an event handler requires not only the removal of the method itself but the removal of the code that hooks an event handler to an event, often found in the `InitializeComponents()` method.

The proper, but relatively hidden, way to add and remove event handlers in `C#` is to use the Properties window. Select the control and then click the Events button in the Properties window (the yellow, “Harry Potter”-like thunderbolt). The Property window displays all the events that the selected control exposes, along with any event handler that is already hooked up to them.

What most developers don’t realize is that the event handler fields are clickable (see Figure 49). If you double-click an empty field, the event handler is created for you. This allows you to choose which event you want to subscribe to. If you already have written an event handler and you simply want to hook it to an event, you can also use the drop-down button next to the selected field that automatically lists all matching event handlers.

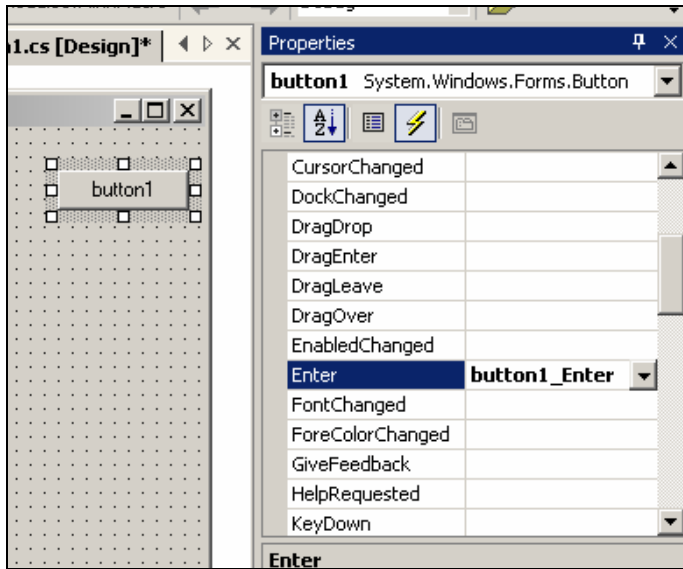


Figure 49 – Adding or removing event handlers through the IDE

Deleting an event handler is as simple as deleting the value in the event field. This also removes the event handler subscription you have in the `InitializeComponents()` method.

As for VB.NET, most developers know how to add non-default events. In the Code view, you can use the top left drop-down list to select a control and the top right drop-down list either to jump to an event handler or create one if it does not exist yet. Because event subscription is taken care of by VB.NET's "Handles" keyword, deleting an event handler is as simple as deleting the method itself.

If you wish to access the same event list in the Properties window for VB.NET, however, you have to upgrade to VS.NET 2005 because that's when VB.NET starts to have the same feature.

Selecting Control Through a Drop-Down List

When there are many controls on a Windows form, it can become troublesome to find a specific control, much less select it. This problem often occurs when many panels overlap one other or when the Windows form becomes too crowded to isolate a specific control that you want to modify.

In this case, select the control through the drop-down list that appears right above the Properties window. This drop-down list is only populated in the Designer view. It contains all the controls that exist on the Windows form. To select a certain control, you just need to know its ID and data type.

Adding an Installer Through the Designer for Windows Services

When you write a Windows service, you often need to create an installer class that facilitates the installation of your service. Traditionally, you add these installer classes by going to the dialog box that appears when you select Project > Add New Item. However, because most Windows services need an installer class, VS.NET adds a handy shortcut.

If you view your Windows service in the Designer view, you just see an empty pane with a few hyperlinks to open up the Server Explorer and Toolbox or to switch to the Code view. However, if you right-click this designer pane, the pop-up menu that appears shows the Add Installer item (see Figure 50). Select it to add the installer class to your Windows service.

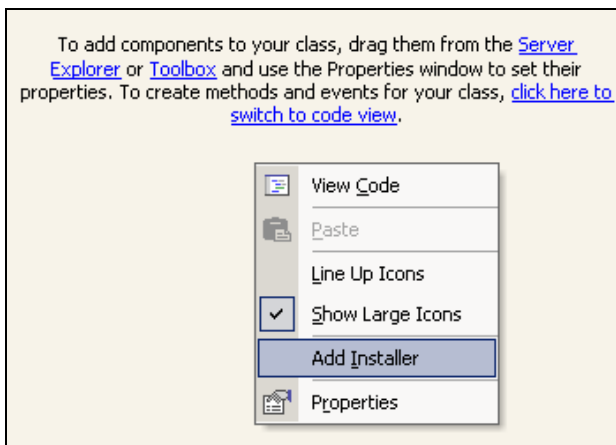


Figure 50 - Adding installer class from the Designer view

Chapter 3: Compiling, Debugging, and Deploying

Not only is VS.NET a great editor, it is also a great compiler, debugger, and profiler. There are many hooks into VS.NET that allow you to control your compilation procedure. The following are absolutely essential in trying to locate and fix a bug: analyzing your code, attaching to running processes that you want to debug, and changing code and variables at runtime. These are just a few topics I cover in this chapter that you need to know when it comes to compiling and debugging your programs.

Linking Files Instead of Copying Them into a Project

Whenever you add another existing item to your project, VS.NET always makes a physical copy of the file in the project's directory first, and then adds that copy to the project. This is, however, not always what you want. To link to an outside file without copying it into to your project, select Project > Add Existing Item. Select the file you want to link to and, instead of clicking Open, click the drop-down list next to the Open button to see the Link option (see Figure 51).

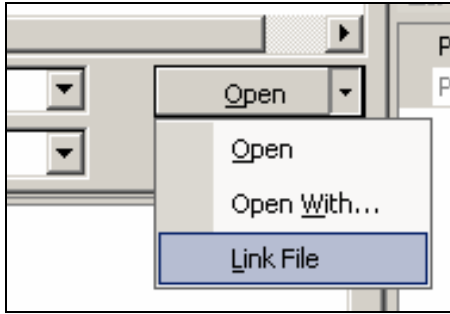


Figure 51 - Linking files instead of copying them

Choose Link File to add the selected file to your project without copying it as well. In the Solutions Explorer, linked files are identified by a small shortcut icon next to the file icon (similar to shortcut icons in Windows Explorer).

This is a great feature for sharing files among many projects. In a solution with many projects, for instance, you can create a “global” AssemblyInfo.cs file and have all your projects link to that file instead of their individual ones. This way, you can easily set the version number for all projects in a central location.

Setting the Default Namespace and Assembly Name

If you follow the official naming guidelines suggested throughout the industry, you would declare your classes in your own company and project-specific namespace. Typically, you end up with the following namespace hierarchy (at a minimum):

MyCompanyName.MyProject.MyClass

When you add new classes with the Add New Item dialog box, VS.NET does not place your new class in any project namespace. By default, it places it in the top-level

namespace, which usually means the name of your assembly. You might want to consider setting the default namespace when you create new projects.

If you select Project > Properties > Common Properties > General (or Project > Properties > Application in VS.NET 2005), you can specify the default namespace in the Root Namespace field (for VB.NET) or Default Namespace field (for C#). This namespace can be many levels deep; new classes added through the VS.NET dialog box will be placed in that specified namespace. In addition, you can also control the name of the assembly that is being generated by specifying it in the Assembly Name field. While Windows applications typically use one word for the assembly name, Control Library projects should be named using the same guidelines as the namespace.

Generating Compiler Warnings and Errors

Compiler directives are special keywords placed in your code that is parsed and handled by the compiler. These keywords allow you to control and fine-tune the compilation process. C++ developers are familiar with these compiler directives, and this idea has now been ported to C#.

For instance, you can specify warning compiler directives in C# as follows:

```
#warning This code is untested
```

Now when you compile your project, the text “This code is untested” appears as a warning in the Task List (see Figure 52). Warnings do not change the compilation behavior, so this change is all right. This is a great technique for commenting code and warning coworkers on a big project about possible problems. Please note that if the project compiles successfully, the Task List will not be set in focus, so you will not automatically see these warnings unless you manually switch to the Task List.

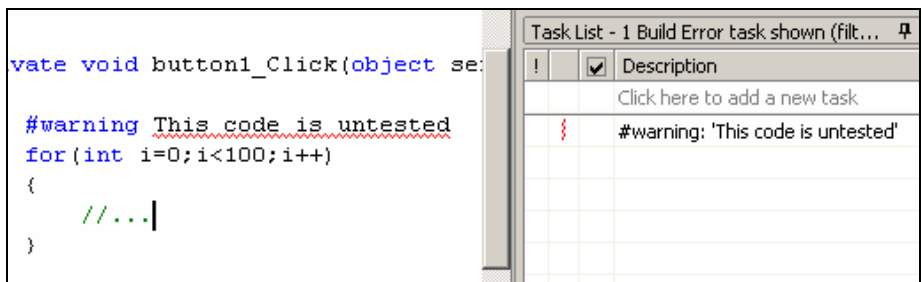


Figure 52 - Specifying a compiler warning directive

A more severe case is to place an error compiler directive in the code:

```
#error don't build solution until I finish this method
```


The error directive forces the specified text to appear in the Task List as well. In addition, however, it forces the compilation to fail. Obviously, you would use this compiler directive only temporarily to prevent compilations to begin with.

In VS.NET 2005, you need to select View > Other Windows > Error List to see the warnings and errors generated by these directives.

Generating Compiler Warnings Through the Obsolete Attribute

Another commonly used way to display warnings in VS.NET at compile-time is to set an Obsolete attribute to a method. Throughout your product development cycle, there will be occasions when a certain method becomes obsolete. Maybe the old method is not useful anymore, it's inefficient, or it has been replaced by another method. If you can't modify those methods, you're stuck with writing another implementation of the method using a slightly different name or signature. For the sake of compatibility, you do not want to remove the old method and break your code. This is where the Obsolete attribute comes in handy:

```
[Obsolete("Use the new MyMethodEx instead!")]
public void MyMethod()...
<Obsolete("Use the new MyMethodEx instead!")> _
Public Sub MyMethod()
```

Setting the Obsolete attribute as above makes a warning message appear in the Task List stating that the particular call to a method is obsolete. The warning message also includes your personalized message that you pass as the attribute's argument (such as, "Use the new MyMethodEx instead!").

As with the warning compiler directives, this method does not change the compilation behavior at all. You also have to activate the Task List to see these warnings. Unlike warning compiler directives, however, the warning only appears if there is code that tries to invoke the obsolete method. If you don't refer to these methods anywhere in your code, these warnings will never appear.

Setting Pre/Post Compile Build Steps

When you compile your solution, VS.NET only compiles each project and produces the desired assemblies. However, if you have complex build requirements, involving additional tasks, you can automate that process in VS.NET as well. These pre/post compile build steps are only available with C#, non-web projects in VS.NET 2003 and 2005. Most C++ developers are familiar with this feature because it comes from a direct port of the old Visual C++ IDE.

Select Project > Properties > Common Properties (or Project > Properties > Build Events in VS.NET 2005). In the Build Events dialog box, you can specify a Pre-Build Event Command Line and a Post-Build Event Command Line. These DOS commands will be executed before and after each build. Click the button with the ellipsis (...) on it to open a command-line editor, where you can specify multiple commands line by line. These commands may be regular DOS commands, such as COPY and MOVE, or calls to batch files—even other Windows programs. All commands run in the order listed and the default output console is the Output window in VS.NET during the build process.

In the command-line editor, instead of hard-coding paths to your project directory and assemblies, use the macros provided for you when you click the Macros button at the bottom. These predefined, commonly-used, and solution-related variables—such as \$(ProjectPath) or \$(TargetFileName)—will be replaced at build-time with the actual values.

A common way to use this feature is to copy certain data files, such as images and documents, automatically to the appropriate bin directory upon each build. Using more complex batch files and parameters, you can now create very customized build processes and reports.

Setting the Assembly Output Path

When you build a project, the produced assemblies are typically placed in the \bin\Configuration subfolder of your project folder, where the configuration folder is typically Debug or Release.

However, this is just a default setting. You can actually specify another directory where you want to place the produced assemblies and external files. Select Project > Properties > Configuration Properties > Build (in VS.NET 2005, select Project > Properties > Compile for VB.NET projects or Project > Properties > Build for C# projects). Place either a relative or absolute path in the Output Path field. This setting is used at the next build.

These configuration-specific properties allow you to specify a different output path for each configuration. For instance, you could set the default output path for the Debug release as the usual bin subfolder, while directing the release build directly to a network share on your internal company network.

Setting the .NET Framework Version for Your Assembly

The .NET Framework has a great side-by-side installation feature. This means that you might have multiple versions of the .NET Framework installed on a given computer, without any of them interfering. By default, all non-web applications always use the

.NET Framework with which they were compiled (if available, of course), whereas web applications by default always use the most recent version of the .NET Framework.

You can specify which .NET Framework is supported and required for your assembly by modifying the application configuration file (MyApplication.exe.config or Web.config). In a nutshell, what you need to do is insert the appropriate Configuration/startup/supportedRuntime and Configuration/startup/requiredRuntime XML tags in the configuration file and set its version attribute to the specific .NET Framework version. This way, for instance, you can force a Windows application to use an older version of the .NET Framework.

This configuration modification is easy in VS.NET 2003. For C#, go to Project > Properties > Common Properties > General > Supported Runtimes. For VB.NET, go to Project > Properties > Common Properties > Build > Supported Runtimes. Here you can set the supported and required runtime versions for your assembly (see Figure 53). To verify that your assembly is picking up the correct version, check the Version property in the .NET Framework class System.Environment.

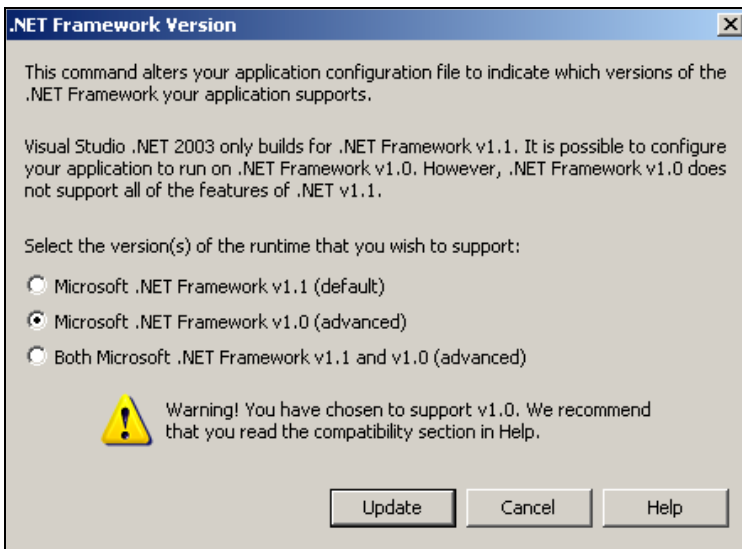


Figure 53 - Setting the .NET Framework version for your assembly

Note that you cannot do this for class library projects because the configuration files are only available for executables and Web applications (just set it on the project that is referencing your class library). Also note that supporting the 1.0 Framework, even though you compile your assembly against version 1.1 (which VS.NET 2003 always does), is an unsupported environment. Simple programs most likely will work, but for more complex programs you are strongly advised to check the compatibilities manually in case your code uses version 1.1–specific features.

Deploying ASP.NET Web Applications

Whenever you deploy an ASP.NET web application to the web server, you have to choose the files carefully that you want to send over FTP. One question that newbies often ask is what files they have to deploy. For instance, you do not need to—and it’s recommended that you don’t—deploy any of the code-behind source code files (because they are already compiled into the DLL file) or the project and solution files. In most cases, all you need to deploy are Web.config, Global.asax, and any files ending with .aspx, asmx, ascx, or .dll. However, you can allow VS.NET to make this tedious selection for you.

Selecting Project > Copy Project allows you to copy your web project to a web server using FrontPage extensions or through a network share (see Figure 54). I usually copy the project to a temporary virtual directory using the File Share option because, by default, it’s set up for the local development computer. At the bottom of the Copy Project dialog box I specify to copy “only files needed to run this application.” By selecting this option, VS.NET finds out which files need to be deployed and copies only those. Now I just need to FTP the complete content of the temporary directory to the web server.

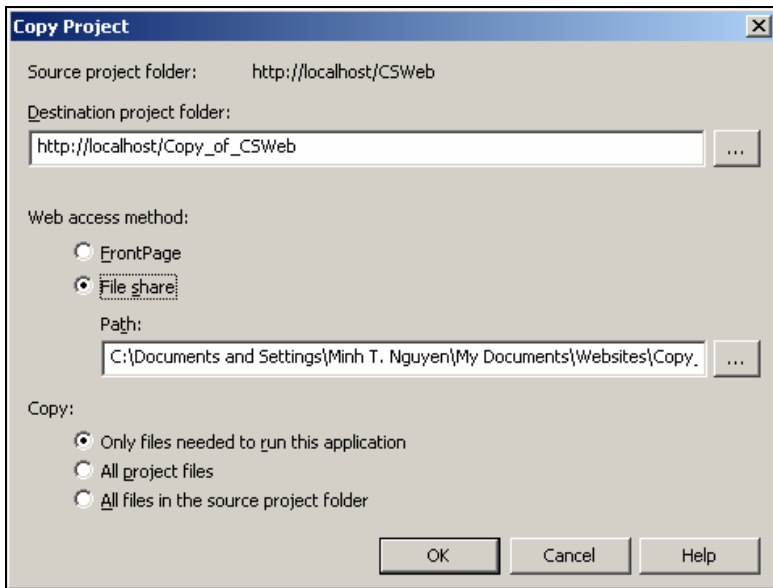


Figure 54 - Letting VS.NET decide which files to deploy

Moving the Next Statement During Debugging

While VS.NET 2002 and 2003 do not implement the oft-requested feature of Edit-and-Continue when debugging a program, they do implement certain debugging features that, unfortunately, many developers often overlook.

One of these features is the ability to move the current execution point to a different location. As you step through your program one line at a time and, for instance, need to jump a few lines back, you can right-click an arbitrary line and choose Set Next Statement from the pop-up menu (see Figure 55). This forces the debugger to jump to that line and continue debugging “normally” from there.

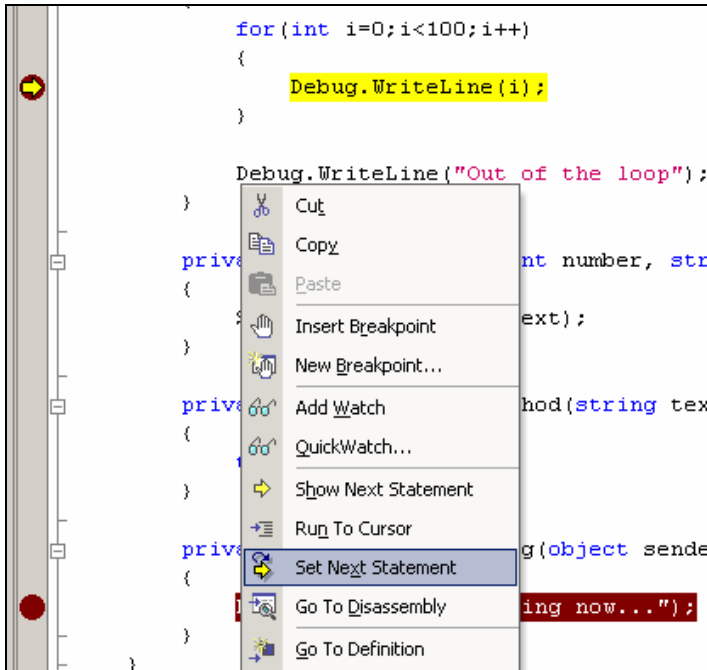


Figure 55 - Jumping out of a For statement during debugging

A faster way to do this is to drag the yellow arrow to any line. Not only can you jump back, but you can also jump forward in and out of control statements. You cannot jump out of the current stack frame, however, so you are limited to moving inside your current method.

This feature should be taken with a grain of salt. Moving the current execution line can bring your program into states that under normal execution would never happen. Still, it's an extremely useful feature to rerun certain code lines without restarting your debugging session.

Changing Variable Values in the Watch Window

Besides moving the next-statement pointer, you can change variable values at debug-time. While debugging your application, you probably have moved your variables of interest into the Watch window (hopefully by dragging your variable there). The Watch window does more than display the current variable value and type; the value field is also editable.

For most value types this is as straightforward as entering the new value. (Unfortunately, DateTime variables can be problematic because you need to change its internal tick value.) As for reference types, you can re-reference variables to other variables. Let's say you have two instances of hash tables in your Watch window, named *foo* and *bar*. Setting the variable *foo* to the reference *bar*'s hash table is as easy as typing **bar** in *foo*'s value field. Naturally, you can only change a reference variable to another reference variable of the same type (or its derived types).

Just as with the previous tip, you should take this tip with a grain of salt because it can bring your program into states that under normal circumstances would never be reached.

Executing SQL Procedures Through the Server Explorer

The SQL Server tree branch in the Server Explorer allows you inspect and analyze a SQL Server instance. Besides the general features of inspecting a database table and Excel-like modifications of table contents by editing rows, the Server Explorer has other useful features.

VS.NET has limited capabilities of editing stored procedures. Right-click any stored procedure and choose Edit Stored Procedure from the pop-up menu. Unfortunately, this feature does not compete well with the Enterprise Manager because error messages regarding syntax error are too general. Nevertheless, it's useful enough to view, edit, and modify stored procedures.

The ability to execute stored procedures at design-time can be very helpful. Right-click a stored procedure and choose Run Stored Procedure from the pop-up menu. VS.NET inspects your stored procedure's parameter list and, if necessary, displays the Run Stored Procedure dialog box where you can enter each parameter's value. Now execute your stored procedure and see the results.

Using the Immediate Window to Display Variables and Execute Methods

The Watch window is a great place to see and modify the values of variables at debug-time. In addition to the Watch window, the user has also access to the Immediate window, which allows you not only to query variable values but also execute methods.

Get to the Immediate window by selecting Debug > Window > Immediate, pressing Ctrl-Alt-I, or typing **immed** in the Command window. You can recognize the Immediate window by the “Command Window – Immediate” in the title bar (see Figure 56).

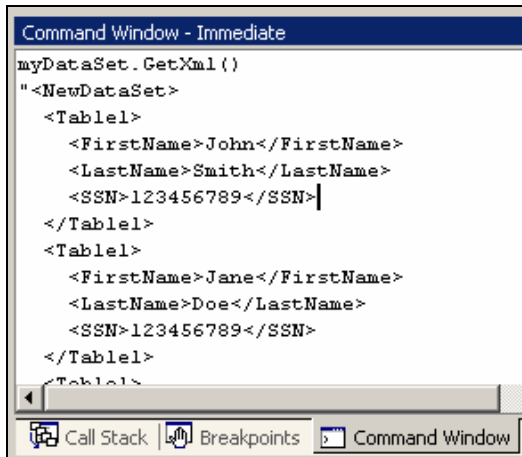


Figure 56 - Using the Immediate window to execute methods

If you type a variable name in the Immediate window and press Enter, you will either see the value of a value-type variable or see a name-value list of all members of the reference-type variable. This also includes private members of your class.

However, the Immediate window is more than just a Watch window clone. You can also evaluate expressions at runtime. Type this, for example:

```
dtMyTable.Rows.Count < 5
```

It returns a Boolean value, depending on the size of your DataTable. In addition, you can actually invoke methods on your object instances. Because VS.NET 2002 and 2003 do not have great visualizations of DataTables at debug-time, a lot of developers use the Immediate window to call the GetXml() method of the DataSet to see the contents of DataTables. This works great in VS.NET 2002, but in VS.NET 2003 new lines in the XML string are unfortunately written as their escape sequences (\r\n), making it a bit hard to read.

At any time, you can browse through previously issued queries and commands by pressing the cursor-up and cursor-down buttons. Starting with VS.NET 2003, the Immediate window also supports IntelliSense.

Customizing the Call Stack

A stack trace is a visual representation of the current hierarchy of method invocations as VS.NET steps through your program. As you debug your program, you step into methods and methods within methods. The stack trace keeps track of all these different levels. If you select `Debug > Windows > Call Stack` or press `Ctrl-Alt-C`, you will see the current stack trace. It displays each method invocation on its own line, along with line-number and argument values. Each new method invocation is known as a stack frame.

The stack trace is a widely known tool and has been around in Visual Studio for a long time. The beauty of the stack trace window is that you can now double-click any line in the stack trace to make VS.NET immediately jump to the method invocation on that particular level of your program. This allows you to identify how you get to the current execution point and also inspect the arguments that have been passed to the methods.

What most developers don't know is that you can customize the Call Stack window. If you right-click the call stack you can customize what appears there (see Figure 57). In addition, you can copy a stack frame to the Clipboard by pressing `Ctrl-C` and send the information regarding a single method invocation to your coworker. You might as well send your coworker the entire call stack by pressing `Ctrl-A` first (or choosing `Select All` from the context menu that appears after you right-click), before copying the selection to the Clipboard.

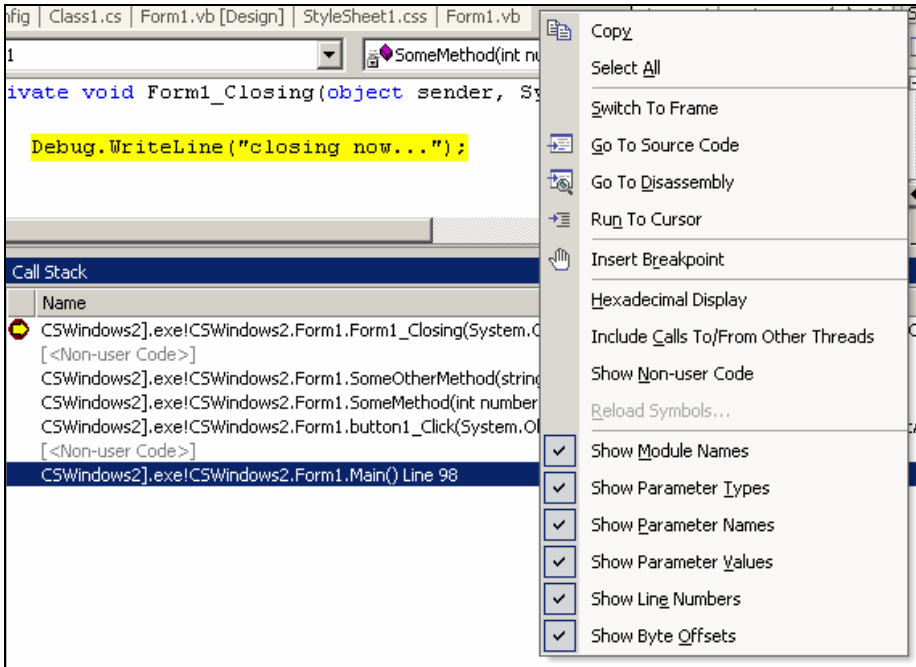


Figure 57 - Customizing the Call Stack window

Placing Program Command-Line Arguments into Project Properties

You often pass parameters to console programs as command-line arguments. In most shell-based environments, such as DOS, you do this by specifying the string values separated by spaces following the console program name.

When you debug your console program through the IDE, you can set these command-line arguments as well. Go to Project > Properties > Configuration Properties > Debugging (in VS.NET 2005 it's Project > Properties > Debug). Under the Start Options section, you can now specify the command-line arguments as well as the working directory. You do not have to specify the program name again, just a space-separated list of string arguments.

Windows forms applications can also accept command-line arguments. Non-console assemblies can use the `Environment.GetCommandLineArgs()` method to reference these arguments.

Attaching VS.NET to an Already Running Process

When you press F5 to debug your program, VS.NET builds your project (if necessary) and starts the program in debug mode. This means that VS.NET is attached to the program so that it can react to breakpoints and other debug-related methods, provided that the project was built with the debug release.

There are cases, however, where you need, or want, to debug an already running process that has not been started with VS.NET. This is possible—again, as long as the process was built in the debug release.

To do this successfully, first open the project for the program that is already running. Go to Tools > Debug Processes to see a list of all active processes on your machine (see Figure 58). In the Processes dialog box, select the process you are interested in debugging and click Attach. On the next screen, you will be asked what program types you want to debug. VS.NET is smart enough to preselect this list for you, so you typically won't need to modify anything here. After clicking OK, VS.NET attempts to attach to that running process. You will see the result of the action by the fact that the program in question is listed at the bottom in the Debugged Processes list.

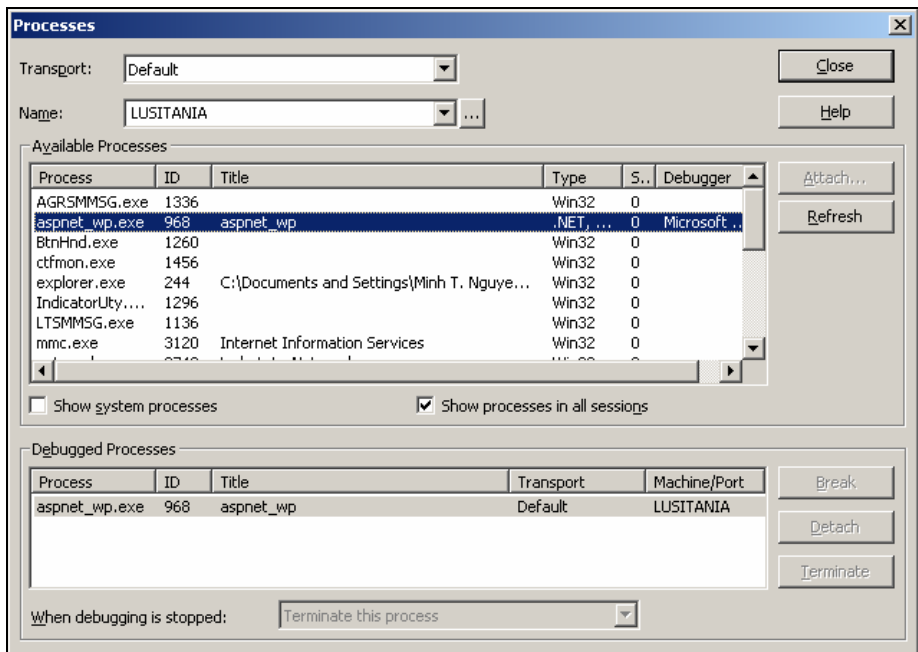


Figure 58 - Attaching VS.NET to the ASP.NET worker process

At this point, you might have also noticed that VS.NET has switched the environment to debug mode and made the Watch and Output windows visible. You are now debugging that running process. This means breakpoints will be hit and you can inspect

all your variable values or do anything else you would do if you were to start the program from inside VS.NET.

You might question the usefulness of this tip, given the long series of steps you need to follow to attach to a running process. However, this technique allows you to debug Windows service–based applications. With Windows service applications, you can't simply press F5 because they need to be installed and started first through the administrative tools. If you build and install your service in debug mode, you can then debug your Windows service using this technique.

Another wonderful thing that you can do with this technique is to debug through SQL Server stored procedures. Provided that SQL Server debugging components have been installed and that you have the necessary debug rights, you can attach to the SQL Server process to step through stored procedures by setting breakpoints in the stored procedures in the Server Explorer.

For web applications, I find this technique even more useful—and often faster. Imagine you are debugging a web form and you press F5. VS.NET attaches itself to the ASP.NET worker process (`aspnet_wp.exe` for all Windows operating systems except Windows Server 2003) and navigates first to the Start page. If the Start page is not what you want to debug, you have to navigate to the second page. If you are using FormsAuthentication, you also need to log in; and if your second page requires you to fill out a form, you need to perform this step to get to the interesting part of the code you wanted to debug. The problem is that you have to repeat all these steps on every new debug session.

A much easier way to debug your page is to attach to the ASP.NET worker process after rebuilding your code. First navigate to your page in question using an external browser window. Fill out the form information (if any) and then attach to the worker process to start debugging your page. When you're done with debugging, click the Stop button to exit the debugging session but keep the external browser window open. After you have modified your code and recompiled it, all you need to do is to attach the debugger again and refresh the external browser to repost the information. There is no need to log in or fill out the form fields again.

This technique makes debugging web applications so much easier, that I have stopped using the old-fashioned F5 method altogether.

Debugging Several Projects Inside the Solution

In a multiproject solution, VS.NET typically only starts the project that you have marked as the “startup project.” That project is marked in the Solution Explorer with bold letters. If you start the other projects through Windows Explorer, you will see that VS.NET does not hit any breakpoints for those projects because VS.NET was not attached as a debugger to them.

You can debug those programs anyway, using the previous tip, “Attaching VS.NET to an Already Running Process.” However, if you right-click your project and select Debug > Start New Instance from the pop-up menu, VS.NET starts that project and attaches

itself to that program as well. You can repeat this step several times to start multiple instances of your program and still debug them all. This helps in debugging multithreaded client-server scenarios.

Additionally, you can tell VS.NET which projects you want to start on each new debug session (see Figure 59). Right-click your solution and choose Set Startup Projects from the pop-up menu. By default, VS.NET uses the Single Startup project, where only one project is started. Simply switch to Multiple Startup Projects and modify the Action value for each property: None, Start, or Start Without Debugging. In addition, you can control the order by which these multiple projects start by clicking the Move Up or Move Down button to position your projects in the list. In a client-server scenario, you can use this to make sure that the server program is started before the client program.

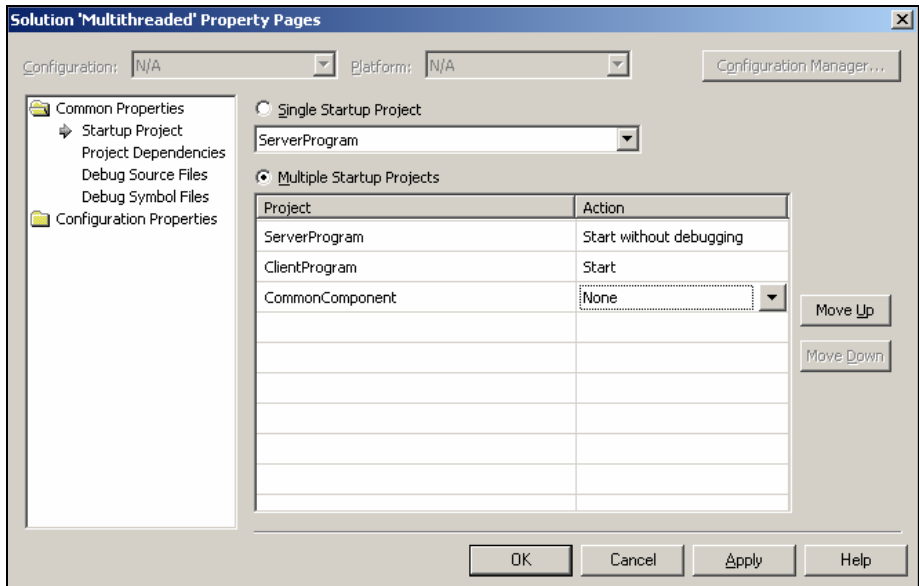


Figure 59 - Debugging several projects at the same time

Breaking Only for Certain Exception Types

A good program usually catches all possible exceptions that can be thrown at runtime. However, this makes it a bit difficult for developers to debug a complex program that is still in development. Because there aren't any unhandled exceptions, VS.NET never catches an exception or prompts the user to break into the code whenever a specific exception is being thrown.

Fortunately, there is a setting in VS.NET that allows developers to specify the exceptions that they are interested in. If you go to Debug > Exceptions (or Debug > Exceptions > Break on Specific Exceptions in VS.NET 2005), you will be presented

with a tree view–style list of all possible exceptions that VS.NET can hook into (see Figure 60). In addition to the many Common Language Runtime exceptions, you can hook into C++, Native Run-Time checks, and Win32 exceptions. The shortcut for this dialog box is Ctrl-Alt-E.

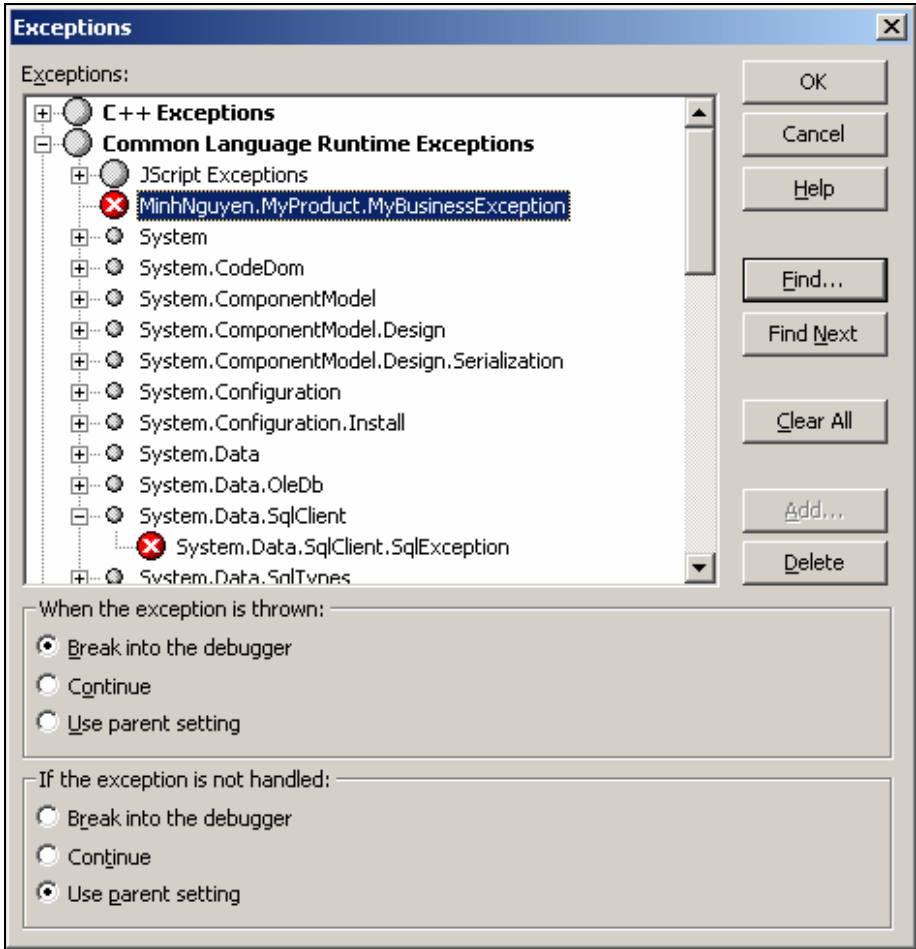


Figure 60 - Breaking when certain exceptions are thrown

Given this list you can now set, for each possible exception, exactly when to break into the debugger. You can either hook into the debugger when a specific exception is thrown or when an exception is not handled. You can also set the default action that VS.NET takes if the conditions for a certain exceptions are met: break into the debugger, simply continue (program execution should just continue with the regular exception handling process), or use the parent setting of the exception (do whatever is specified for the parent class of the chosen exception).

In addition to the predefined .NET exceptions, you can hook into your own .NET exceptions. By clicking the Add button in the Exceptions dialog box, you can specify the complete, fully qualified string that defines your .NET exception (for example, “MyCompany.MyProduct.MyBusinessException”).

The VS.NET 2005 version of the Exceptions dialog box is a bit more intuitive (see Figure 61). The two conditions (when the exception is thrown and when it is not handled) are presented as two different columns. There is no choice on what action to take if the condition is met; it simply breaks into the debugger when that happens.

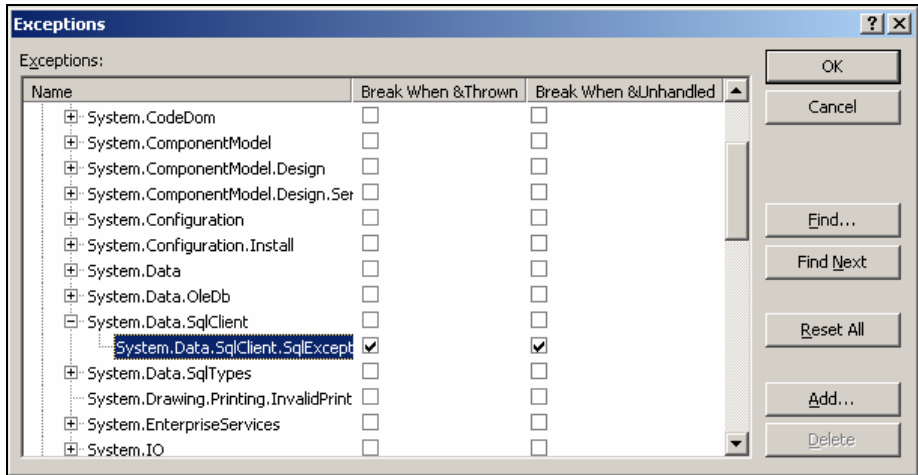


Figure 61 - Breaking for certain exceptions in VS.NET 2005

Breaking Only When Certain Conditions Apply

Most developers add or remove exceptions by clicking the gray vertical bar to the left of the editor. Clicking it adds and removes the red circle that indicates a breakpoint. By doing so, many developers never encounter the very useful conditions that you can set for breakpoints. Select Debug > New Breakpoint (or press Ctrl-B). In VS.NET 2002 and 2003, you can get to the same window by right-clicking an existing breakpoint’s red circle and choosing Breakpoint Properties from the pop-up menu. On some occasions, you might not see that menu item, even though you are right-clicking precisely on an existing breakpoint. In that case, I recommend removing the breakpoint entirely and setting a new one in the same location to pull up the New Breakpoint window.

Two buttons stand out at the bottom of the Breakpoints window. This is where you specify a condition under which a breakpoint becomes active. First enter a .NET expression. This can either be simply a variable name (“myBoolVariable”) or a more complex .NET expression (“((System.DateTime.Now.Second % 10) == 0”). You can choose to break into the debugger if the expression evaluates to True or when the expression value changes. Naturally, for the first option, the expression has to evaluate

to a Boolean value. For the second option, your expression can be anything. VS.NET breaks into the debugger only if the runtime value of that expression changes from the last time it passes by this conditional exception (this implies that program execution has to pass by this code segment at least once previously, before it can recognize a change in value).

Given the flexibility of the expression, this feature can be very powerful. For instance, you can debug a snapshot of a DataSet only if the DataTable row size is greater than 0.

In VS.NET 2005, all the above-mentioned conditions are accessed differently. First set your breakpoint as you would normally do. Then right-click your breakpoint. From the context menu, choose Condition to get to the same screen (see Figure 62).

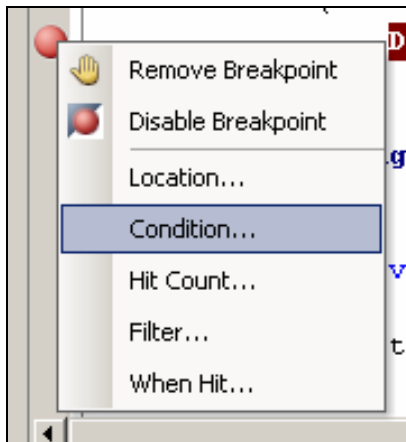


Figure 62 – Setting condition for breakpoints in VS.NET 2005

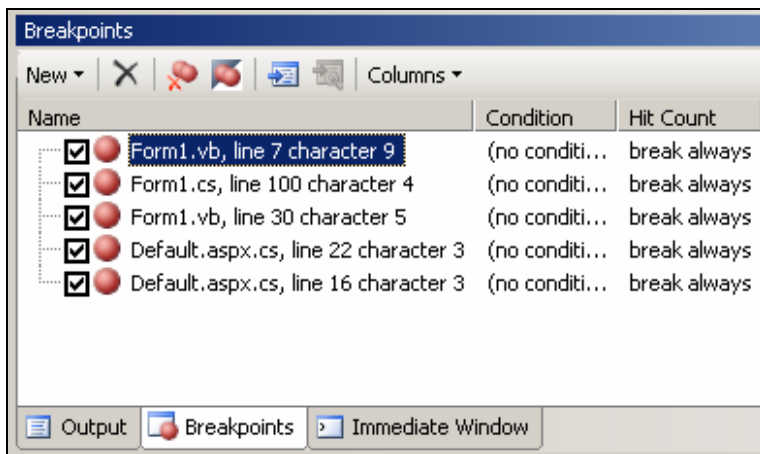


Figure 63 - Breakpoints window in VS.NET 2005

You can also see and modify the condition in the Breakpoints window in VS.NET 2005. Open that window by selecting Debug > Windows > Breakpoints or pressing Ctrl-Alt-B. There you will see a list of all breakpoints that you have set, along with their conditions. You can disable breakpoints from this window as well, using the check boxes, or jump to their location in the file by double-clicking them.

Debugging ASP.NET Web Application Through Trace.axd

Tracing web applications has become a lot easier ever since VS.NET allowed you to debug the actual ASP.NET code. Gone are the `Response.Write("here")` days of the classic ASP world. Even without VS.NET, however, ASP.NET has an extremely useful tracing feature: the trace.axd HTTP handler.

To make use of this feature, go into your Web.config file and modify the configuration/system.web/trace node. Setting its “Enabled” attribute to True starts a specific HTTP handler on your web application that listens to the virtual file trace.axd (<http://localhost/YourCurrentWeb/trace.axd>).

This URL outputs a list of the last HTTP requests, including their associated file that handled the request and the time. If you zoom into a specific request by clicking the View Details link, you’ll be presented with a vast amount of web-related information, including the HTTP header collection, all server variables, cookie and session state information, the execution flow of your program, as well as your control tree and its occupying ViewState and render size information (see Figure 64). The information you find on this page is specific to each request and is extremely useful. You can quickly find out which web control uses the most ViewState or renders with the most HTML code. You can also emit trace messages to this listing by using the `System.Web.TraceContext` class. Note that this is not the `System.Diagnostics.Trace` class but the one to which you can get to by going to `this.Trace` in your web page or web control.

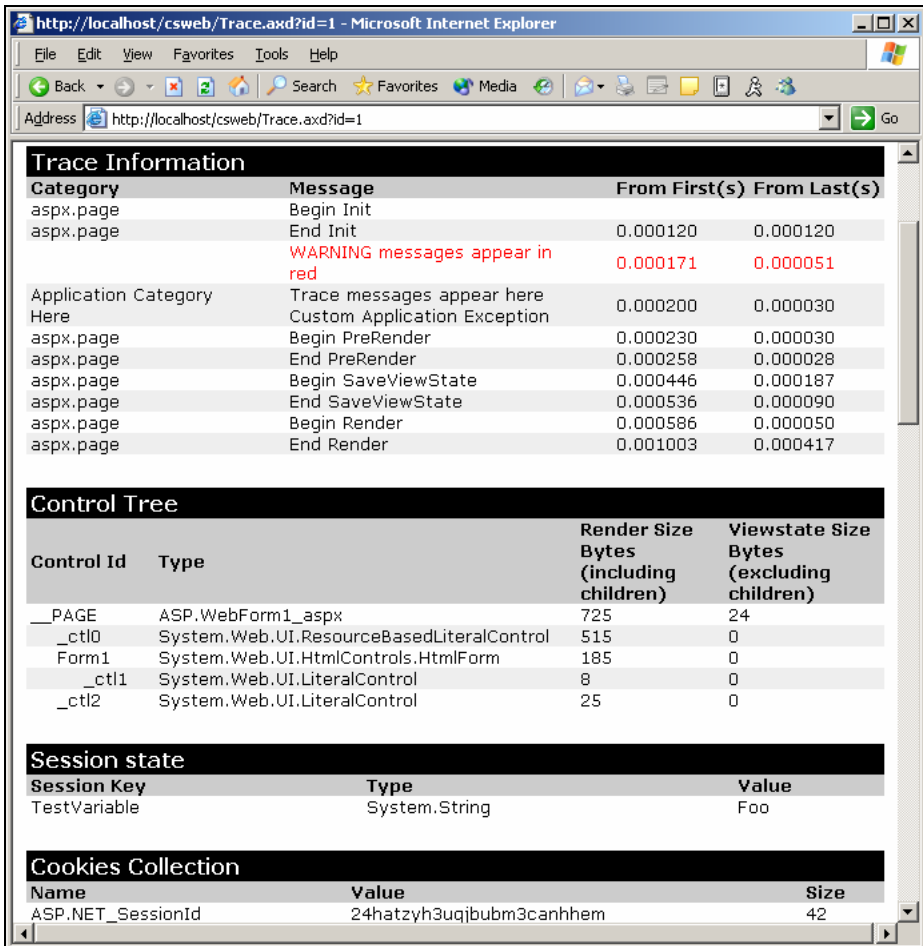


Figure 64 - Trace.axd displaying useful debugging information

The TraceContext class contains a Write() method as well as a Warn() method. Both methods send a string to the trace page, along with its time-relevant information. Both methods provide overloads for you to specify a category as well—just a string that appears in a separate column on the trace page. The Warn() method displays the message in red.

Note: The From First(s) column specifies, in seconds, the amount of time that has passed since the very first trace information was emitted—essentially, the time since BeginInit. The From Last(s) column specifies, in seconds, the time that has passed since the previous message trace has been emitted. This is the time difference relative to the previous line. Consequently, the FromFirst value should equal the previous FromFirst value plus the current FromLast value.

The trace node in Web.config also allows you to specify the maximum amount of requests that are traced. The number you specify in the requestLimit attribute defines the number of requests after which tracing stops. Unfortunately, there is no way to indicate a sliding window mechanism, where old trace information would be dumped upon reaching the maximum amount of requests, to ensure continuous logging.

The localOnly attribute is an important one. In most cases it should be set to True. Doing that will allow only the currently logged-on user to see this page. Anyone who tries to pull up the trace.axd HTTP handler with a URL other than through the localhost URL will encounter an error message. This makes sure that only the local developer can see the trace information, and not any unknown Internet user.

Lastly, the pageOutput attribute specifies whether you want trace information to be emitted on the actual page in addition to the trace.axd. I usually don't recommend this because it often negatively interacts with the actual HTML content of the page.

Saving Any Output Window

The Output window (Ctrl-Alt-O) shows a lot of trace information regarding your program execution. It lists whenever the .NET Framework loads a DLL for your application and, probably more importantly, all the messages that you have emitted with System.Debug.WriteLine. If you ever want to save all these trace logs, you can of course copy and paste everything into a Notepad file. However, you could skip Notepad altogether because the Output window behaves like the main editor in VS.NET. This means you can press Ctrl-S to save the entire output to a file. You can search through the Output window using Ctrl-F and even apply some of the other editor tips and tricks I described in Chapter 1, such as Ctrl-C for copying an entire line or Ctrl-R, Ctrl-R for word-wrapping (although VS.NET 2005 now offers a button for word-wrapping in the Output window).

Chapter 4: Using VS.NET 2005

VS.NET 2005 is an enormous improvement over VS.NET 2002 and 2003. Microsoft made an amazing tool even better. The jump to VS.NET 2005 is so big compared to the jump from VS.NET 2002 to 2003 that it requires a chapter of its own. This chapter concentrates on all the new tips and tricks not found in previous versions. This does not mean that the first three chapters are not applicable to VS.NET 2005 anymore. In fact, most of them still work or have been greatly improved (unless otherwise noted).

This chapter does not cover version 2.0 of the .NET Framework and its new classes and syntax additions. Naturally, VS.NET 2005 is tied very much to version 2.0 of .NET, and occasionally I dive into version 2.0-specific code, but this is only done to demonstrate a new IDE feature. Whenever I simply refer to VS.NET in this chapter, I naturally mean VS.NET 2005.

This book is based on VS.NET 2005 Beta 1. The final version of VS.NET 2005 might differ slightly from how I describe it in this chapter.

Refactoring Code

One of the most-quoted tips for VS.NET 2005 is probably the code refactoring features. Code refactoring is the process of restructuring your code for the purpose of cleanup without changing the business logic. This involves renaming symbols, factoring duplicate code segments into single methods, and so on. There are many refactoring actions you can perform with VS.NET. Simply right-click a selection and choose Refactor from the pop-up menu. You are now presented with a list of possible refactoring methods you can apply to your selection (see Figure 65).

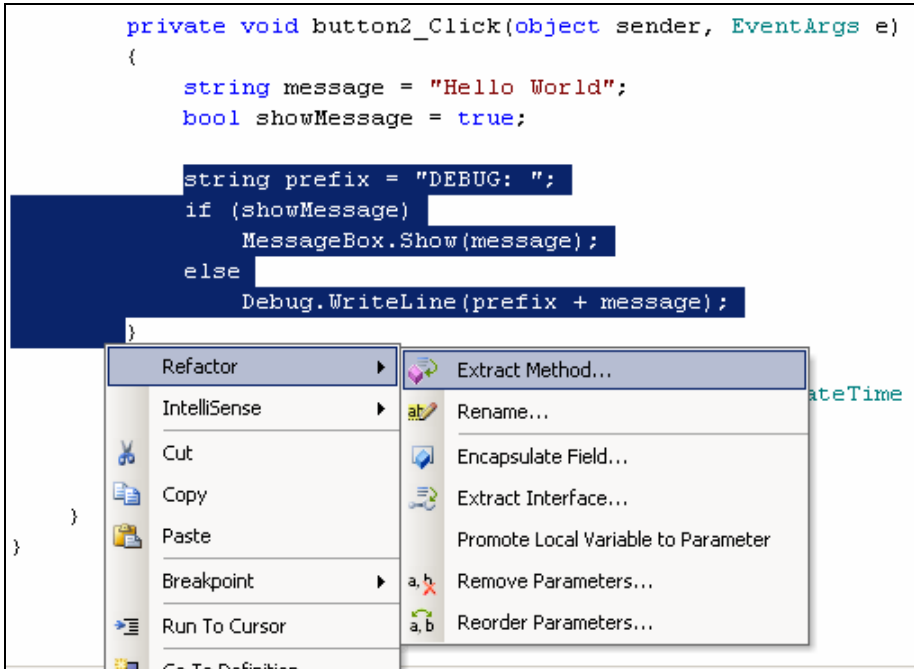


Figure 65 - Extracting method from a selection

Extract Methods (Ctrl-Alt-M, M) is the process of creating a new method and moving your selection into this new method. In place of the old selection, the invocation to the new method will be generated (see Figure 66). This feature is more than just moving code into a method. VS.NET inspects your selected code, looks for variables references that are not defined in the selection, and converts them into parameters.

```
private void button2_Click(object sender, EventArgs e)
{
    string message = "Hello World";
    bool showMessage = true;

    NewMethodName(message, showMessage);
}
private static void NewMethodName(string message, bool showMessage)
{
    string prefix = "DEBUG: ";
    if (showMessage)
        MessageBox.Show(message);
    else
        Debug.WriteLine(prefix + message);
}
```

Figure 66 - New method after extraction with parameters

Rename (Ctrl-Alt-M, R) allows you to rename a symbol (a variable, constant, enum, and so on) and automatically perform the necessary query-replace throughout your entire solution to update references to this symbol. In the past, developers had to do a global query-replace manually, which unfortunately also replaced occurrences of the variable name in string literals or when they were part of another variable name. The Rename feature in VS.NET 2005 allows you to exclude string literals or comments so you can update only real variable references.

Encapsulate Field (Ctrl-Alt-M, E) allows you to create properties including their Get and Set functions around a private field, and automatically update all references throughout your solution to refer to the new property instead of the old field.

Extract Interface (Ctrl-Alt-M, I) allows you to select from all the existing properties and methods of the current class, and generate an interface for your solution into a separate file. The interface that is automatically created also includes the proper Get and Set methods for properties and signatures for methods. The new interface bears the same name as the current class, with the exception that the letter *I* is prefixed to abide by the .NET-naming convention for interfaces.

Finally, the self-explanatory methods **Promote Local Variable to Parameter** (Ctrl-Alt-M, P), **Remove Parameters** (Ctrl-Alt-M, V) and **Reorder Parameters** (Ctrl-Alt-M, O) are great additions when working with methods. To remove or reorder parameters, just right-click the method name.

Generating Method Stubs

VS.NET has the ability to generate method stubs for you, given a method invocation. Sometimes as you develop code and already know what a certain method looks like, you might find yourself writing the method invocation code before actually writing the method body. In that case, you can simply right-click the method invocation and choose

Generate Method Stub from the pop-up menu (see Figure 67)—or use the new smart tag for that.

```
public class Foo
{
    public Foo()
    {
        string lastName = "Doe";
        string firstName = "John";
        int SSN = 123456789;

        Person newPerson = CreatePerson(firstName, lastName, SSN);
    }
}
```

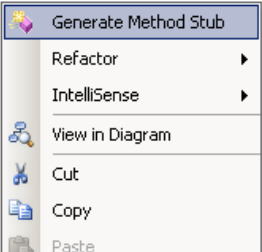


Figure 67 - Generating method stubs from a method invocation

VS.NET creates a method shell that matches the exact same method signature as the invocation (see Figure 68). Not only are the formal parameters of the same type as the actual parameters passed to the method, but the formal parameters have the same name (provided that variables were passed to the method, not actual values).

```
public class Foo
{
    public Foo()
    {
        string lastName = "Doe";
        string firstName = "John";
        int SSN = 123456789;

        Person newPerson = CreatePerson(firstName, lastName, SSN);
    }

    private Person CreatePerson(string firstName, string lastName, int SSN)
    {
        throw new NotImplementedException();
    }
}
```

Figure 68 - Newly generated method with the same signature

In the method body, VS.NET inserts code to raise a `NotImplementedException` in case the developer forgets to fill the method with the real business code before building the application.

Using Error Correction Suggestions

You know how a word processor’s spell-checker allows you to right-click a misspelled word to see a list of possible corrections? VS.NET has a similar feature for incorrect syntax. Let’s say you refer to the `Timer` class but forget to include the correct namespace at the top of your file. The word `Timer` appears with a blue squiggly line, indicating that VS.NET cannot find the `Timer` class.

At the end of the blue squiggly line appears a short red line that, when moused over, becomes a smart tag that suggests all the `Timer` class alternatives to choose from: `System.Timers.Timer`, `System.Threading.Timer`, `System.Windows.Forms.Timer`, and even my user-defined `MinhNguyen.Timer` (see Figure 69). Once you make a selection, VS.NET replaces the unresolved reference to `Timer` with the fully qualified class name. VB.NET also provides suggestions for incorrect syntax, such as forgetting the “`End Try`” statement in a `Try-Catch`.

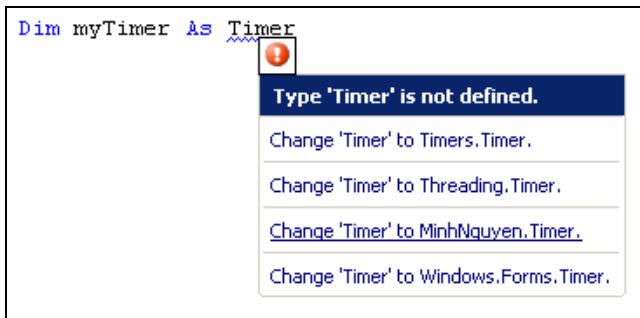


Figure 69 - VS.NET suggesting ways to resolve classes

In C#, you can use the blue smart-tag that appears on the left when you click the word or simply right-click the underlined word and choose `Resolve` from the pop-up menu. The suggestions appear as submenu items. You have the option to add the corresponding using-statement or simply use the fully qualified class name.

These suggestions also include user-defined classes, and not only those that exist in the .NET Framework.

Using Predefined Code Snippets

Code snippets are popular, predefined text templates from which developers can choose, rather than type manually. For instance, instead of typing all the code you would need for a `foreach` loop, you can simply type **foreach**. IntelliSense displays the `foreach` item with the new code snippet icon (see Figure 70). If you choose that code snippet item from the IntelliSense list and press `Tab`, VS.NET inserts that predefined code snippet for the “`foreach`.”

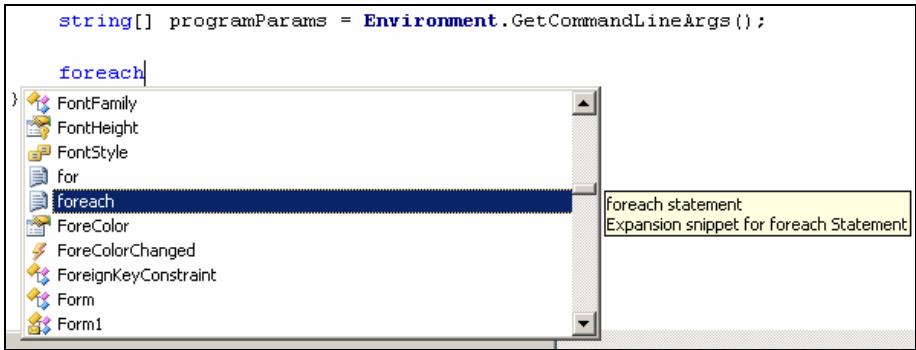


Figure 70 - Code snippets in IntelliSense (document fragment icon)

In addition to expanding a predefined code snippet, VS.NET highlights in yellow those placeholders in the code snippet that you—as the user of the code snippet—have to change to customize to your specific usage. Press Tab to jump between these placeholders. In our Foreach example, the three items you have to change are the object type, object name, and name of the collection you want to loop over. VS.NET 2005 displays a highly customized IntelliSense that lists only the items that can be legally inserted into the placeholders (see Figure 71). For the collection_to_loop placeholder in the Foreach example, that would only be objects that implement the IEnumerable interface.

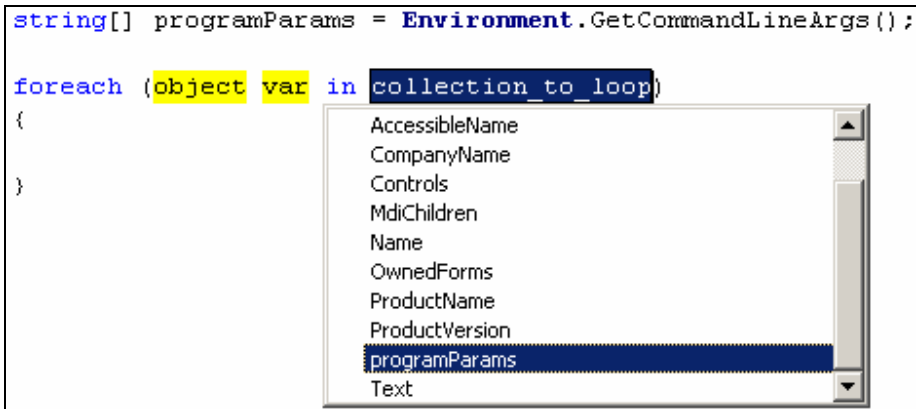


Figure 71 - Foreach code snippet expanded

You can also insert a code snippet directly by right-clicking anywhere in the code editor and selecting IntelliSense > Insert Expansion in C# (or Insert Snippet in VB.NET) from the pop-up menu. This lists only code snippets in IntelliSense. VB.NET's predefined list of code snippets far outweighs that in C#. It includes code expansions for real-life best practices, from database access and cryptography to graphics, math, and IO operations—and much, much more. In addition, all code snippets in VB.NET are categorized neatly into logical folders.

In the previous example, we have expanded the word “foreach” with the complete Foreach code snippet. Another way of inserting code snippets in C# is to surround a selection. In this case, you highlight a text segment, right-click it, and select IntelliSense > Surrounds With from the pop-up menu. This lists only the code snippets of type SurroundsWith, which are basically code snippets that have a defined main body section. If you choose “foreach,” for instance, all the selected text will be put into the Foreach loop’s main body.

As I mentioned before, VS.NET 2005 is shipped with many predefined code snippets from which you can choose. You can see the list of all code snippets and their preassigned shortcuts by going to Tools > Code Snippets Manager or by clicking Ctrl-K, Ctrl-B.

The Code Snippet Manager contains all the predefined code snippets for each .NET language, as well as their preassigned shortcuts (see Figure 72). Note that you can create your own code snippet if you are familiar with the code snippet XML schema used by VS.NET.

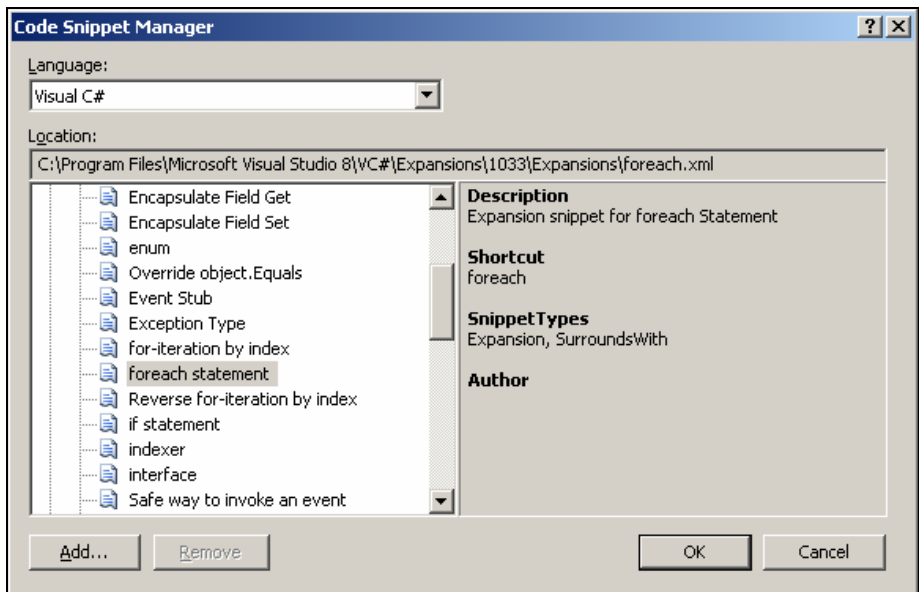


Figure 72 - Using the Code Snippet Manager

It’s even easier to create a code snippet with VB.NET. Simply highlight your full code segment, right-click it, and choose Create Snippet from the pop-up menu. This brings your selection into the snippet editor, allowing you to edit and modify it. In this editor, you can right-click a selection and choose Create Replacement from the pop-up menu to define and name the placeholders mentioned before. Here you can also define your snippet’s name, shortcut, description, required references, and so on.

Aligning UI Elements Automatically

If you are positioning UI elements in a Windows form, you definitely have noticed various colored lines that appear on the form as you move or resize elements (see Figure 73). This allows you to snap your UI element to vertical or horizontal lines. Solid blue indicates lines to which other UI elements have already been snapped; they help you align elements consistently. Green dotted lines indicate the default margin between the UI element you are moving or resizing and the elements around it; they help you maintain uniform spacing between elements. Finally, solid red lines indicate that the text inside the current element is aligned with an adjacent UI element or its text.

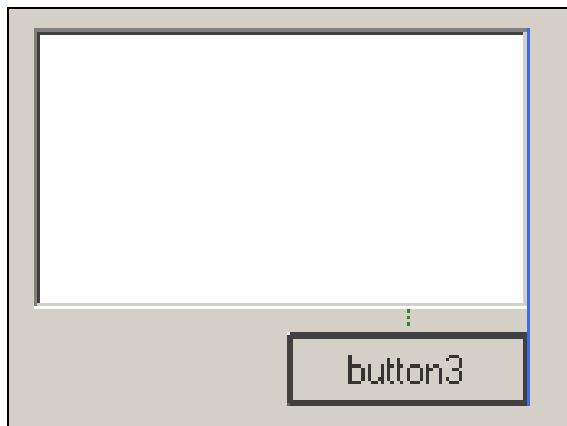


Figure 73 - Colored lines (solid blue and dotted green) for aligning UI elements

If you want to position UI elements without snapping to these colored lines, press **Alt** to turn off automatic alignment temporarily. You can also switch back to the grid in VS.NET 2002 and 2003, where all UI elements are aligned to a predefined grid. Simply go to **Tools > Options > Windows Forms Designer > General** and change **LayoutMode** back to **SnapToGrid**. Note that after changing that value, you need to close and reopen the Designer view to use the newly selected layout mode. In **SnapToGrid** mode, you can press the **Ctrl** key to move elements without snapping them to the grid.

Adding a Standard Menu Strip

Standard Windows applications use a common set of top-level menu items. In most cases, they are **File**, **Edit**, **Tools**, and **Help**. VS.NET 2005 allows you to add these default menu items to your own Windows forms applications.

Drag a **MenuStrip** to your Windows form. With the **MenuStrip** selected, the description panel below the **Properties** window shows a **Insert Standard Items** link (see Figure 74). If you click that link, VS.NET inserts these standard items onto your **MenuStrip**. Menu items you insert contain the default submenu items as well. For instance, the **File** menu

includes the usual New, Open, Save, Save As, Print, Print Preview, and Exit items, along with its default shortcuts, hot keys, and icons.

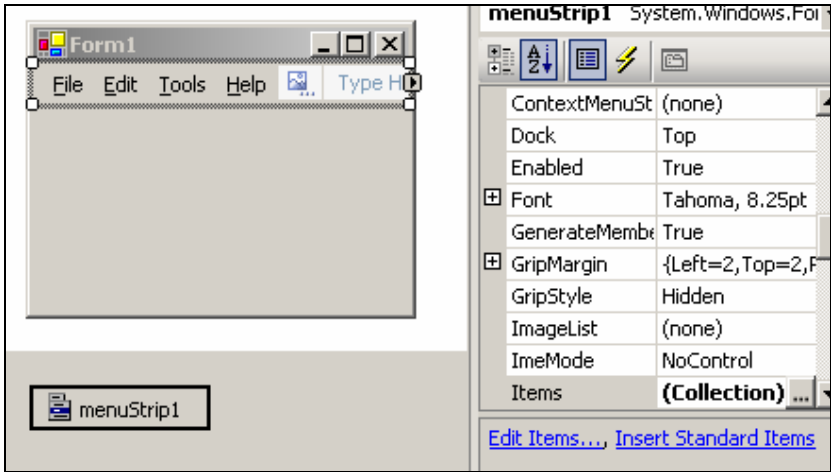


Figure 74 - Standard menu items automatically inserted

Editing UI Element Properties with the Property Editing View

When you first create your Windows forms application, you most likely drag your buttons, text boxes, and other UI elements onto your form. When you do so, all these elements have default names such as “Button1,” “Button2,” “TextBox1,” and so on. Changing these default names or their text values requires you repeatedly to select an element and change its Name or Text property in the Property window. With many elements in a complex Windows forms application, that can take awhile.

VS.NET 2005 introduces a fast way to set these “popular” properties for your controls. Right-click your form and choose Property Editing View from the pop-up menu, or click the Property Editing View button on the Layout bar (see Figure 75). This switches your Forms Designer to Quick Edit mode, where you can change the property of all controls in back-to-back fashion.



Figure 75 - Property Editing View button on the right of the Layout bar

Decide which property you would like to change by choosing it from the drop-down list on the new Quick Edit Mode tab. Whichever property you choose, the Forms Designer displays the value of that property for each control inside editable text boxes that hover

over the UI element. As you edit one of those text boxes, the property of the underlying UI element changes and the next text box gets the focus (see Figure 76).

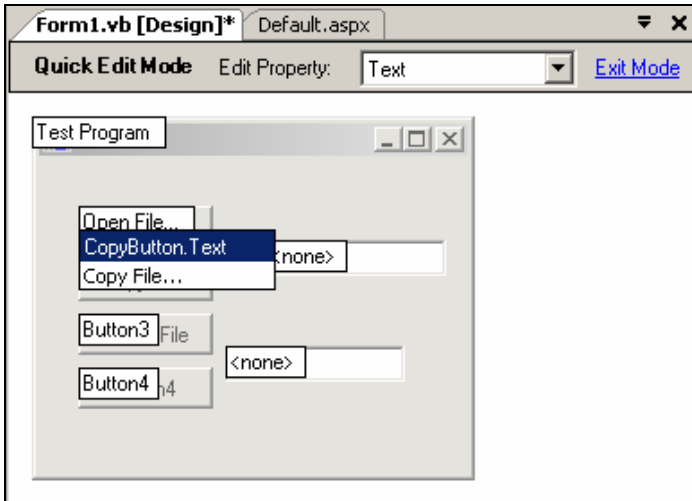


Figure 76 - Changing the Text property of each control back to back

This method allows you to change a common property for all elements quickly and easily because you don't have to lift the mouse or scroll from item to item. Simply type the new value, press Enter, and repeat for each control. It really doesn't get any easier than that.

Controlling C# Code Formatting Precisely

Developers have different preferences when it comes to C# formatting standards. By "formatting standards" I mean those little rules about whitespace, indentation, new lines, wrapping, and so on. It seems to me that developers always follow their own pet rules. This is not so much a problem with VB.NET because, by default, it auto-formats your code to the VB.NET standard as you enter a new line. With C#, on the other hand, there are many different ways to write the same code.

VS.NET 2005 now allows you to control precisely how your C# code is formatted. If you select Tools > Options > Text Editor > C# > Formatting, you will be presented with a wide array of settings to determine your preferred C# formatting directions, including indentation, new lines, spacing, and wrapping. There doesn't seem to be a single thing you cannot define. Most of these settings are self-explanatory, but in case you aren't sure about something, a preview window displays the choices you have made using a C# code example.

Setting the Tab Order of Controls

The tab order is the order by which controls on the form receive focus as you press the Tab key. You can control this order by setting the Tab Index property of each control to a number that corresponds to the position in this order. This can prove difficult at times because you don't know—and can't see—the other controls' tab index unless you select them. VS.NET 2005 introduces a new way to set the tab order: the Tab Order button on the Layout bar (see Figure 77).



Figure 77 - Tab Order button on the left of the Layout bar

Clicking the Tab Order button displays the tab index for all UI elements on the form. Here's the beauty of this mode: Not only do you now see all the tab indices, but you can repeatedly click on each UI element to set the tab order in linear fashion. The first element you select has a tab index of zero. The next one you select has a tab index of one, and so on. As you set the index for each control, the background color of the tab index caption switches from blue to white, so you can keep track of which UI elements you have already tagged. To prevent you from accidentally selecting a wrong UI element, a gray rectangle surrounds the element you mouse over for better identification.

When you are done setting the tab order, click the Tab Order button again or simply press the Escape key. Unfortunately, this tab order feature works only for Windows forms, not web forms.

Performing a Class, Instance, and Method Search

In a solution with many projects, it is sometimes hard to locate a class, instance, or method. If you know the name of a class, instance, or method, but don't know where it is located, you could always use the global find feature.

A better way is to use the new search feature in the Class view. Open up the Class view (click the tab usually right next to the Solutions Explorer or press Ctrl-Shift-C) and note the search window at the top. You don't have to type the full name of the item you are looking for; a partial search term works too. The Class view filters the items to show only those that match your partial search key.

Viewing Code Definitions

The Code Definition view is a window new to VS.NET 2005 that allows you to view the definition of a class as you move your cursor over a type. View this window by selecting View > Other Windows > Code Definition View or pressing Ctrl-Shift-D. By

default, it appears as another window below the main text editor. As you move your cursor around and inside a word that defines a class (see Figure 78), the Code Definition view displays all the properties, methods, indices, and other members of the class—all of which are fully equipped with XML comments (if they exist). This tool is similar to the output of the WinCV utility that accompanied the initial version of VS.NET.

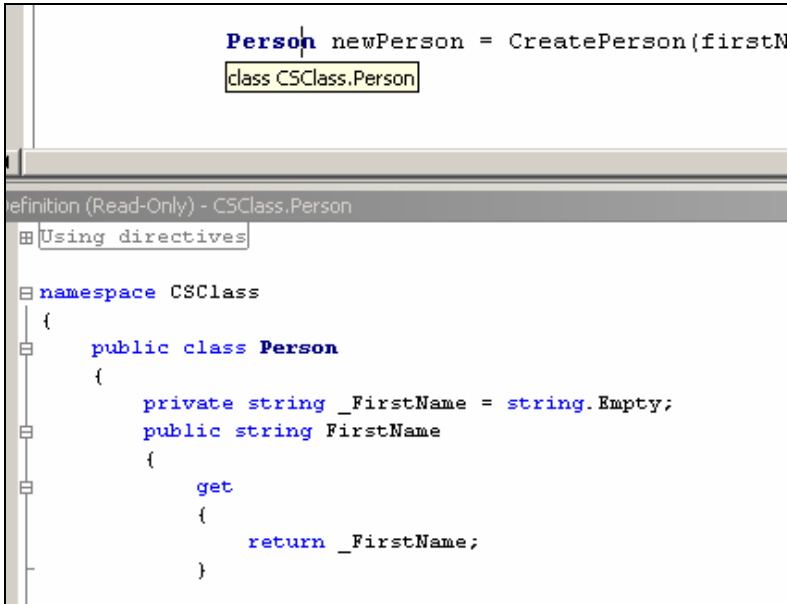


Figure 78 - Code Definition view displaying class information

The Code Definition view also works for user-defined classes. In fact, if the class is part of your solution, it displays the actual source code of that class—not just what it can get through reflection.

Editing Web Controls in the HTML View

In VS.NET 2002 and 2003 you can modify web controls only in the Designer view. This is not a pleasant way to work because of the additional rendering process and often incorrect auto-formatting of your HTML code. This has now been greatly improved. For one, your HTML code is no longer formatted unless you specifically request it. Second, anything that you can do in the Designer view, you can now do in the HTML view.

For instance, you can drag controls from the Toolbox directly into the HTML editor. The necessary HTML or ASP.NET tag will be inserted for you. While the cursor is located in a certain web control, you can also modify the control's properties in the Properties window. Changes immediately appear in the HTML view as attributes. Setting a property to blank removes the attribute from the HTML or ASP.NET tag.

Validating HTML Code for Accessibility

Enterprise web applications should be designed with disabled persons in mind. A set of comprehensive rules and guidelines exist that you should follow when writing HTML code to allow disabled persons to navigate your website with their particular tools. For example, image tags should always include a defined ALT attribute. The string inserted in the ALT attribute can be read through speakers to visually impaired visitors. There are many websites out there that can check your website against these rules, but this functionality is now built into VS.NET.

In the HTML view, simply click on the Accessibility button (see Figure 79) or select Tools > Check Accessibility. You can also right-click your page in the Solutions Explorer and choose Check Accessibility from the pop-up menu.



Figure 79 - Accessibility button

In the Accessibility Validation dialog box, you can choose the set of rules against which to validate your HTML code (see Figure 80). WCAG stands for Web Content Accessibility Guidelines (read more at www.w3.org/WAI/GL/). In these guidelines each suggestion—referred to as a “checkpoint”—is assigned a priority. Priority 1 checkpoints are the most important. If your HTML code has a Priority 1 violation, web users cannot surf your website with their tools. Priority 2 suggestions are also strongly recommended, but leaving them out will not render your website useless. Priority 3 checkpoints are recommendations only, not requirements.

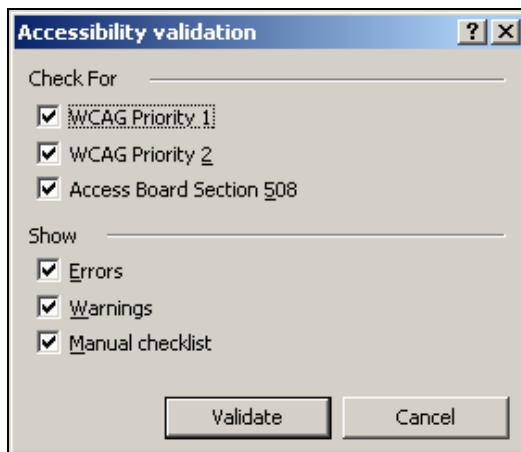


Figure 80 - Checking HTML for accessibility

Access Board Section 508 refers to the amendments to the Rehabilitation Act passed by the United States Congress in 1998 that require federal agencies to make their websites

accessible for people with disabilities (read more at www.section508.gov). Between WCAG and Section 508, the rules are very similar and, naturally, overlap. If you create a website that will be used by a federal agency, be sure to check the Access Board Section 508 option.

Once you click Validate, VS.NET validates your HTML code and displays all errors and warnings in the Task List. If you have never validated your website against these standards before, prepare yourself for a long list of suggestions!

Working with Different .NET Languages

In VS.NET 2002 and 2003 you are not able to create an ASP.NET website using different .NET languages. The website has to be written entirely in C# or entirely in VB.NET, or in an ASP.NET-supported .NET language. This is no longer the case with VS.NET 2005. If you right-click your website project and choose Add New Item from the pop-up menu, you will notice that the new Add New Item dialog box prompts you for the language you want to use for the new file. You can even have a content page written in one language that uses a master page driven by another (see Figure 81).

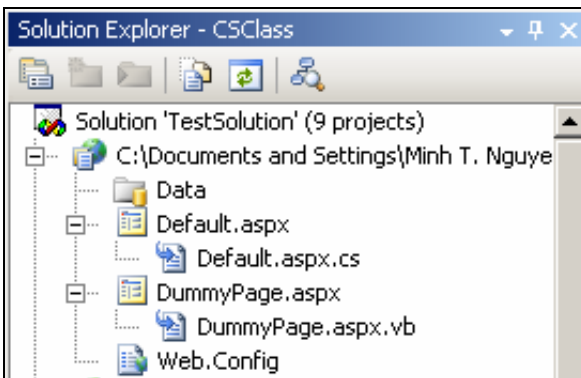


Figure 81 - Website using different .NET languages

It is unlikely that a single developer will choose to write a website in several different languages, but this feature at least allows the possibility of importing pages from another project into your current project without worrying about language differences.

Opening Web Projects Through FTP

In case you haven't noticed yet, project and solution files are not required anymore to open a web project in VS.NET 2005. As a result of this, there are now several other methods to open websites. If you select File > Open > Website, you will be presented with several options.

You can access them through the file system (simply pointing to a folder), through IIS (pointing to a virtual directory), or through FTP. When connecting via FTP, VS.NET downloads first the list of files and directories that make up the website. Then when you open a website file, it downloads the actual file into a temporary location and opens it from there (see Figure 82). Saved files are then uploaded to the web server, making the change take effect immediately.

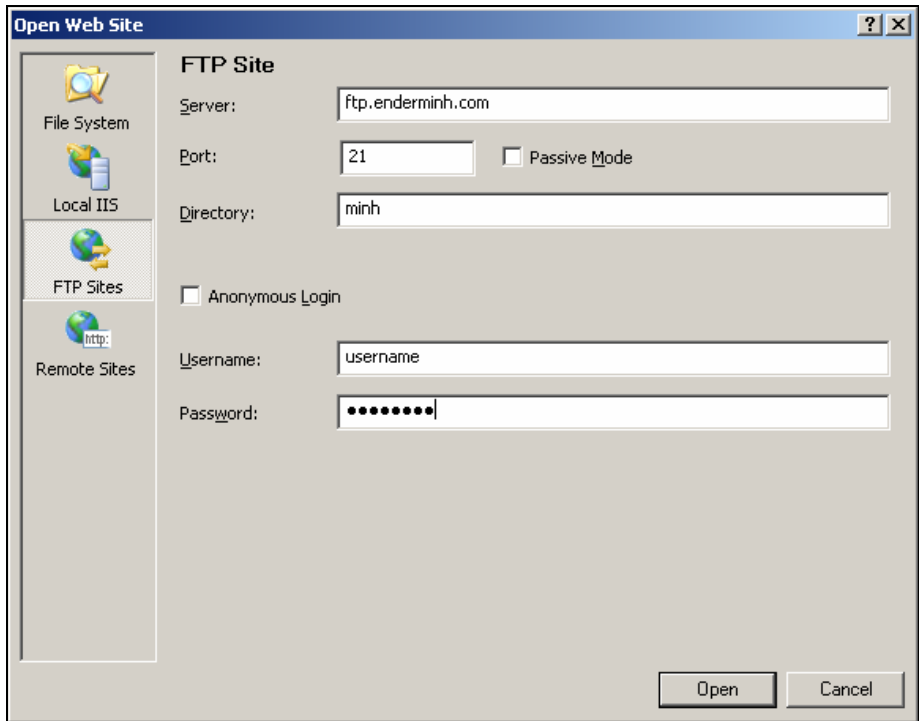


Figure 82 - Opening a live website via FTP

Granted, this feature would not be used in a real enterprise situation, but it can help to modify a simple hobby website quickly and easily. Most notably, you don't have to go through a third-party FTP client just to modify a few web pages.

Importing and Exporting IDE Settings

VS.NET is an extremely powerful tool. There are many things in the IDE that you can customize to suit your own needs. The problem with all this customization is that you become accustomed to your particular settings. Moving from one machine to another can be troublesome because you are not able to move your IDE settings along with you.

VS.NET 2005 allows you to export your IDE settings to an XML file (the extension is actually ".vssettings") so you can import it into another instance of VS.NET on another

computer. Simply select Tools > Import > Export Settings. In the tree view shown in the Import/Export Settings dialog box, you are presented with all the customizable options you can export (see Figure 83). Check the ones you want to be part of your profile and export them to the .vssettings file.

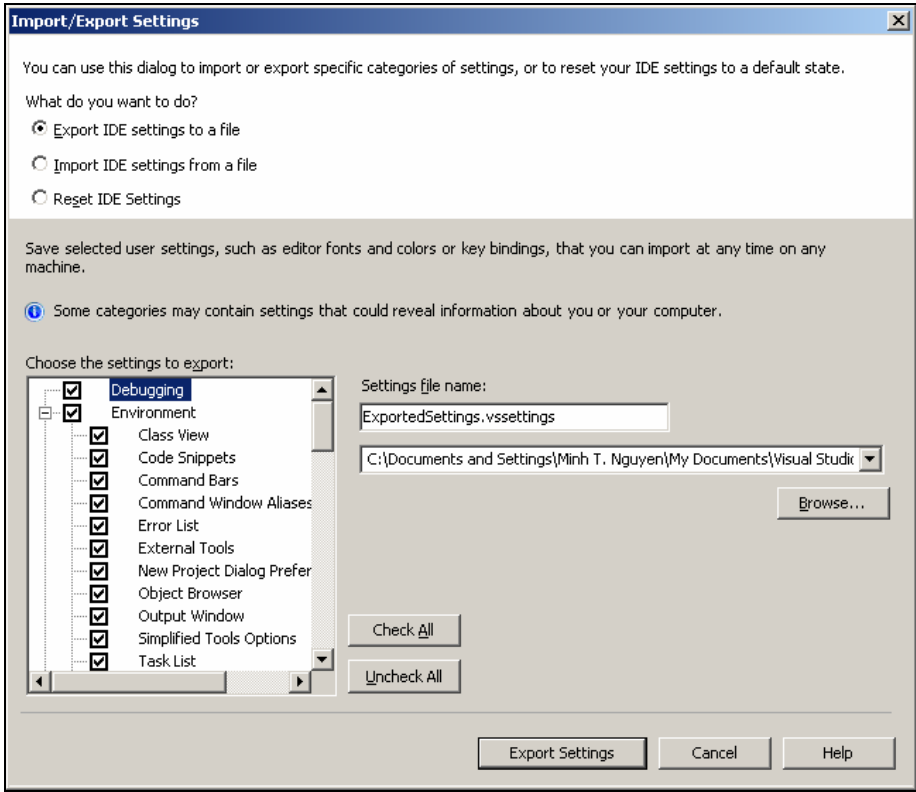


Figure 83 - Exporting or importing IDE settings

When you import the .vssettings file to another VS.NET IDE, you can select which settings you want to import and which ones to ignore.

In the same dialog box you can also reset your complete VS.NET IDE to a particular profile. These might be custom profiles that you saved before. You can also reset to the default installation settings (which is just another regular .vssettings file).

Reading this, you might now consider having a dedicated person on your team create a master .vssettings file and then e-mail it to all your team members so that they can import it individually. As it turns out, there is another application of this feature. Simply create the single .vssettings file and place it on a well-known network share on the intranet. Then ask every developer to go to Tools > Options > Environment > Import > Export Settings > Team Settings. There they have to turn on Track Team Settings File and point it to that shared .vssettings file. Next time they start their IDE, it will detect the file and import it. The beauty of this is feature that another trusted team lead

can export a version of the shared .vssettings file and overwrite it, so that the IDEs of each developer will detect that change and import it upon the next startup.

Closing All Other Windows

It's not uncommon to have a lot of files open at the same time when you develop your program. After working for a while, you might have several dozen files open and want to close all of them except the one you are currently working on. With previous versions of VS.NET, you have to close them all through Windows > Close All Documents and then open the file you are working on again.

VS.NET 2005 allows you to right-click one of the file tabs and choose Close All But This from the pop-up menu, which does exactly what it says (see Figure 84). Other menu options new to VS.NET 2005 include Open Containing Folder, which fires up Windows Explorer and opens the folder in which your file is located; and Copy Full Path, which copies the full file path of the selected file into the Clipboard.

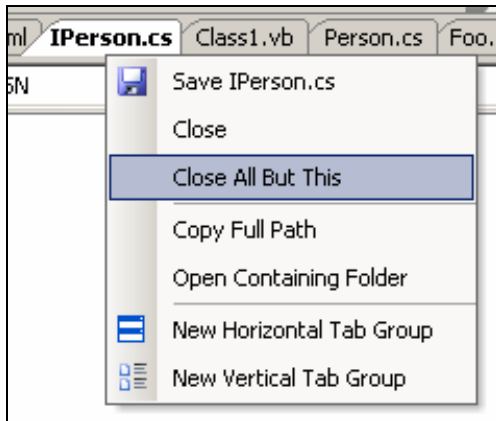


Figure 84 - Closing all other windows except the current one

Showing Shortcuts for All Buttons

I am a strong supporter of using (and memorizing) shortcuts whenever they are available. Keyboard shortcuts prove to be faster than fumbling with the mouse. It helps that many VS.NET menu and submenu items show an associated keyboard shortcut as well because every time you click the menu item, you are reminded of the shortcut. Maybe next time you won't use the menu anymore.

This reminder is also available for the toolbar buttons. Select Tools > Customize and check both the Show ScreenTips on Toolbars and Show Shortcut Keys in ScreenTips

options. Now as you mouse over a button, the ToolTip that appears after a small delay will also show the button's keyboard shortcut, if available.

Building Selected Subset of Projects

The Build menu in VS.NET 2005 displays several new actions you can take. Besides the usual Build and Rebuild actions, and the new Publish (for ClickOnce technology) and Clean actions, you can now choose to build only a selected subset of the projects in a solution.

Usually VS.NET is smart enough to realize which projects have changed when you do a regular build. However, sometimes you want to force a rebuild of several other projects as well. Before VS.NET 2005, this was possible only by using Rebuild All, which was time-consuming if your solution contained many projects. With VS.NET 2005, you can now highlight several projects (either by pressing the Ctrl key while selecting individual projects or by pressing the Shift key to make a contiguous selection, as in Windows Explorer). Once you have highlighted several projects, go again to the Build menu. Now the usual single project menu items have been replaced with items that apply to the selected projects: Build Selection, Rebuild Selection, Clean Selection, and Publish Selection (see Figure 85).

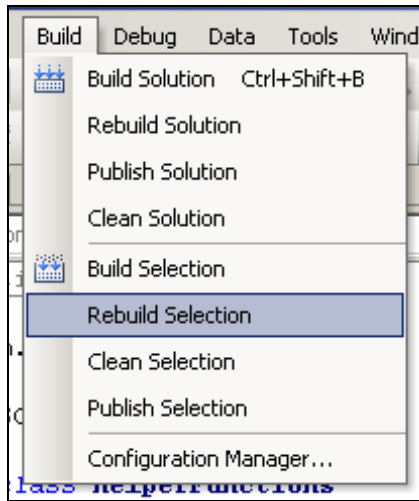


Figure 85 - Build menu changes after selecting multiple projects

Using Edit-and-Continue in VB.NET

One of the biggest complaints about VS.NET 2002 and 2003 is that they do not have the Edit-and-Continue functionality of VB.NET. For those of you who don't know,

Visual Basic 6 (the primary IDE for VB developers before VS.NET) allowed developers to modify code during a running debug session, so that VB6 would immediately pick up the code change and execute it without having to restart the session. This useful feature helped developers fix and test code during a single debugging session. Although this feature disappeared in VS.NET 2002 and 2003, it has returned in VS.NET 2005. Simply modify code when a breakpoint is hit and you'll see that VS.NET uses the new code right away.

***Note:** This feature does not work with C#.*

Expanding Variable Members While Debugging

Previous versions of VS.NET allow you to mouse over variable names at debug-time and get the string representation of your variable. While this is a great feature, it immediately becomes useless for custom classes if you don't override the `ToString()` method. VS.NET 2005 improves upon this feature by allowing you to expand your variable information and dig into your variable's internal fields. Whenever you mouse over a class instance (of any type), you can now click the plus sign to see your instance's internal fields. You can even expand the internal class member variables.

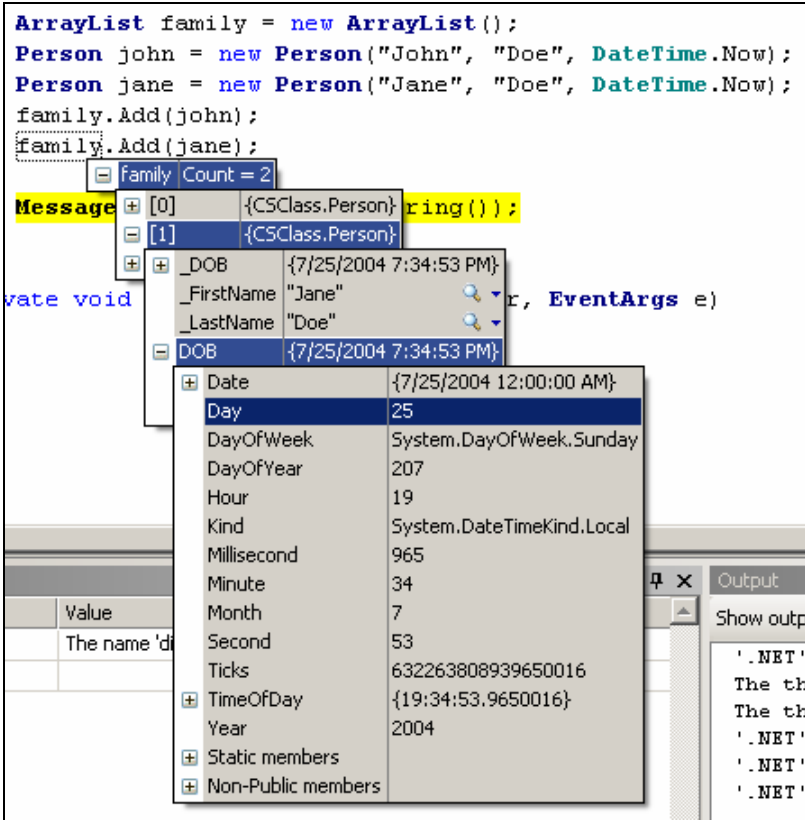


Figure 86 - Expanding variable field members

Using Data Visualization

In addition to expanding objects as you mouse over them while debugging, VS.NET 2005 features helpful data visualization windows for common variable types that are hard to visualize in pure string format. Mouse over a DataSet or DataTable, for instance, and see a small magnifying glass icon to the right of the variable value. Click the drop-down menu on the magnifying glass to see the different visualizers available for your data (see Figure 87). For DataSet and DataTable, the single visualizer displays the data in tabular format (see Figure 88). This is certainly easier than debugging and drilling down to each row and column, as you have to do in VS.NET 2002 and 2003.

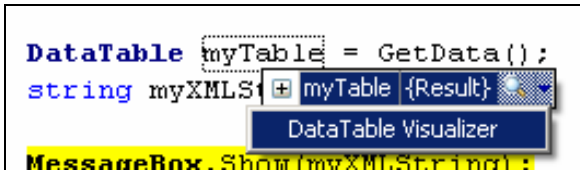


Figure 87 - Pulling down the magnifying glass to select a data visualizer

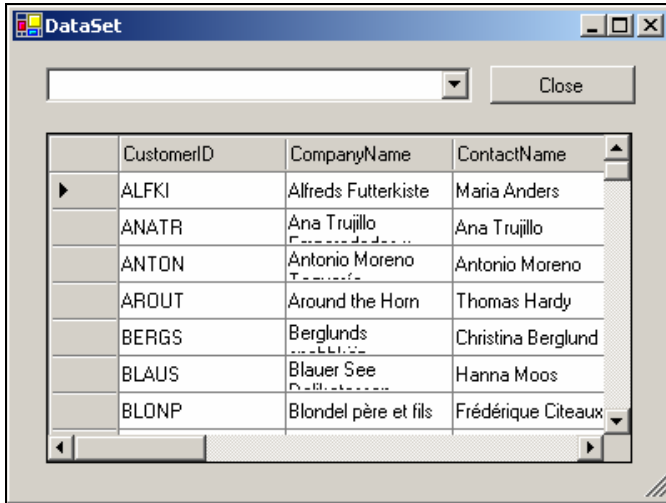


Figure 88 - Data visualization for DataSets and DataTables

There are other built-in data visualizations. When you mouse over a string, you can choose from among three data visualizers. The Text Visualizer simply displays long strings with word-wrapping. The XML Visualizer displays strings containing XML fragments in hierarchical form (as they would appear if you viewed an XML file in Internet Explorer). The HTML Visualizer parses the string and renders the data as a small web page.

Instead of clicking the drop-down icon next to the magnifying glass (which prompts you to choose from all available visualizers), you can just click the magnifying glass itself, which automatically chooses the last-chosen visualizer for your specific data type.

Chapter 5: Other .NET Tips and Tricks

This chapter contains tips and tricks related specifically to .NET. Many of them are not related to VS.NET per se. However, I decided to include them in this book because they apply to most VS.NET developers. Developers who are new to .NET might not be familiar with some of these tips and tricks.

Obfuscating Your .NET Assemblies

When you compile .NET source code, it isn't compiled into native machine code but rather into Microsoft Intermediate Language (MISL). The .NET Framework on a specific machine handles the job of compiling MISL code into native machine code—a process known as just-in-time compilation. This implies that MISL is a very open format. Anyone who understands MISL can inspect, read, and understand your program.

Because your code won't be compiled into native machine code, your assembly is therefore vulnerable to decompilation—the process of analyzing your assembly and inferring the original source code from it. This process is not very hard; in fact, many utilities can do this job for you very effectively: Just feed the utility your compiled executable and out comes the original C# or VB.NET source code. Of course, the produced source code differs from your own, actual source code but you can definitely read and understand the code.

To remedy this problem, you need to obfuscate your code after compiling it. Obfuscation is the process of changing your source or intermediate language code without changing the logical flow of the overall program. This involves such things as renaming variables names, putting several unrelated methods into a single overloaded method, changing easy-to-read switch/case/if statements into hard-to-read goto statements, and so on. While these steps alone might not prevent decompilation, the source code that those utilities produce will be much more cryptic.

VS.NET 2003 and 2005 ships with a third-party obfuscator called Dotfuscator. You can run it by selecting Tools > Dotfuscator Community Edition.

No Rebuilding After HTML Changes

I cringe whenever I see web developers rebuild an entire web application during debugging just because they edited the HTML part of the application. There is no need to rebuild and restart the web application when you modify the HTML part of an .aspx file. This includes changing web control attributes, even though that can modify the web page output tremendously. Whenever you change only the HTML, simply refresh the browser without restarting the debugging session. The change will appear immediately.

Iterating Over Strings on a Char per Char Basis

Strings are funny variable types in .NET. They are reference-type variables, even though they behave like value-type ones. Although you can change a string's value through code, they remain immutable (a new string is created whenever you “change” a string).

One cool thing about strings is that they often behave like an array of characters. This means that you can iterate over a string on a character-by-character basis using a simple Foreach loop:

```
foreach(char myChar in myString) { ... }
```

Also, you can get the seventh character in a string by indexing it with the bracket:

```
char myChar = myString[6];
```

Using Inline Strings as Object Instances

To make strings seem even more esoteric, inline strings behave like object instances. If you type a string with quotes in your editor and follow it by a period, IntelliSense will appear and allow you to choose from among the same methods as for a string variable:

```
string prefixRemoved = myString.Substring("INFORMATION:  
".Length);
```

Adding App.config to Your Application

Calls to the `System.Configuration.ConfigurationSettings.AppSettings` collection always inspect an assembly's `.config` file. For web applications, this is the `Web.config` file; for Windows applications, this is the `MyWindowsApp.exe.config` file. By default, VS.NET creates the `Web.config` file for a new empty web application. However, for Windows applications, you have to add that file manually.

Many developers create a text file in the `MyWindowsApp.exe.config` name format and place it in the output folders of project. This is problematic, however, because you have to do this twice: once for the Debug folder and once for the Release folder. You also have to change the name of the `.config` file whenever you change the name of the output assembly.

There is a better way to manage the `.config` file. Right-click your project and select `Add New Item` from the pop-up menu. In the `Add New Item` dialog box, navigate to the `Local Project Items > Utility` folder and choose `Application Configuration File` (with VS.NET 2005, simply choose the “app” item). This adds a text file named `App.config` to your project (see Figure 89).

***Note:** Some VS.NET installations do not show the `Application Configuration File` item in the `Add New Item` dialog box. In those cases, you have to create a text file called `App.config` and place it in the project's root folder.*

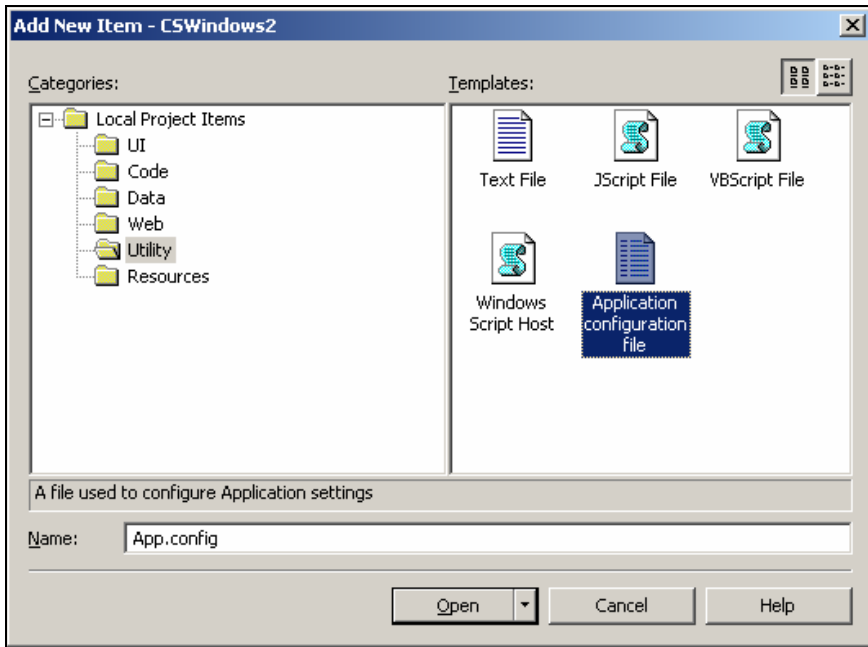


Figure 89 - Adding an application configuration file

The good thing about this is that whenever you build the project, the `App.config` file is copied to the correct output folder (Debug, Release, or whatever your current configuration might be) and renamed appropriately (“`MyWindowsApp.exe.config`”). This allows you easily to control your configuration in a central place in VS.NET.

Note: The `App.config` file is only applicable to main assemblies and not class library DLLs.

Using Intermediate Language Disassembler to Inspect a .NET Assembly

During the VS.NET installation, you can choose to install the .NET Framework SDK as well. The SDK contains many useful utilities. One of them is Intermediate Language Disassembler (`ildasm.exe`), usually located in one of the VS.NET subdirectories:

VS.NET 2002: C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin

VS.NET 2003: C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin

VS.NET 2005: C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin

The Intermediate Language Disassembler allows you to open and inspect a .NET assembly (see Figure 90). You can see all the namespaces and classes defined in the

assembly and read the Microsoft Intermediate Language code. Reading the MSIL code can be very helpful in fine-tuning your application if it is performance-critical.

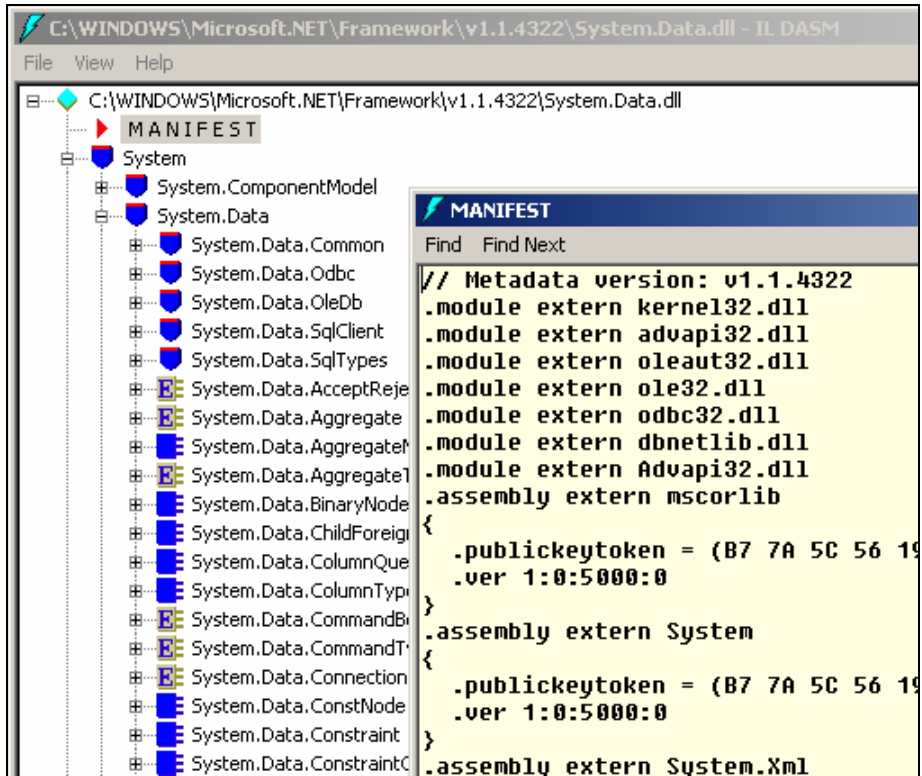


Figure 90 - Using ILDASM to inspect a .NET assembly

The ILDASM utility is also useful for determining the .NET Framework version that the compiler used to build a given assembly. Look at the assembly's manifest section and see which version of the System assembly it is referencing. A version number of 1:0:5000:0 means it was compiled with .NET Framework 1.0, while a version number of 1:0:3300:0 indicates it was compiled with version 1.1. Not surprisingly, .NET Framework 2.0 has a version number starting with "2:0."

Using Windows Class Viewer to Reference a Class

In the same directory as the ILDASM utility in the .NET Framework SDK you will find another useful utility called Windows Class Viewer (WinCV.exe). This program acts as a quick search for any .NET class defined in the .NET Framework. Search for a class using a part of its name and it will list all the possible matches. The search is extremely

fast and results automatically update as you narrow down the search by typing more letters.

Once you select a class, you will be presented with C++ header-like information, including all the constructors, methods, properties, events, and other member fields (see Figure 91).

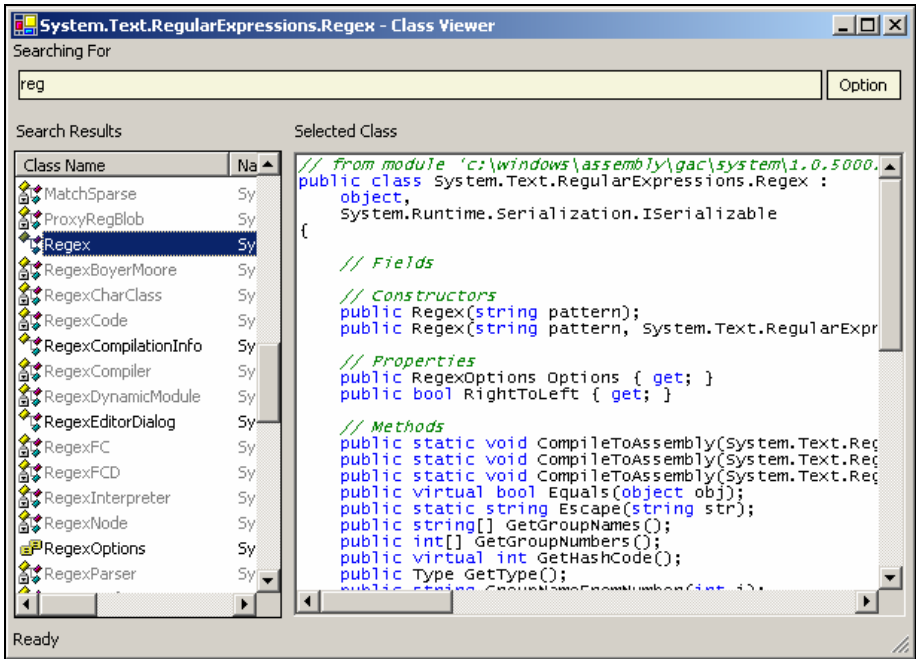


Figure 91 - WinCV.exe displaying all member fields of a class

Note: VS.NET 2005 offers you something very similar when using the Class Definition view.

Running aspnet_regiis to Fix an IIS Installation

The golden rule when installing VS.NET or the .NET Framework SDK on a new computer on which you want to develop web application is that IIS has to be installed first, because the .NET installation installs and registers the necessary ASP.NET ISAPI extension DLL. If you install IIS after you install .NET, IIS will not work correctly with ASP.NET.

Instead of reinstalling .NET, you can use a very useful utility that's bundled with the .NET Framework SDK. In the .NET Framework directory (typically it's C:\Windows\Microsoft.NET\Framework\v1.1.4322, or whatever version of .NET you are running), there is a small command-line utility called aspnet_regiis.exe.

Run that utility as follows to redo the IIS part of the .NET installation:

```
aspnet_regiis -i
```

Note: *This trick is more than a “configure IIS for .NET” feature; use it to “fix” IIS whenever ASP.NET doesn’t work on your computer anymore.*

If you are missing just the client-side script files that ASP.NET uses (the JavaScript files often found in the /aspnet_client/system_web/version folder), just run the utility like this:

```
aspnet_regiis -c
```

Lastly, this utility is useful for enabling ASP.NET in IIS 6. As you know, Windows 2003 Server has ASP.NET disabled by default. You can enable ASP.NET by selecting IIS > Extensions but if you need to enable it through script, just run the utility as follows:

```
aspnet_regiis -enabled
```

Precompiling Your ASP.NET Web Application

Whenever you use VS.NET to compile your web application, all it does is compile your code-behind classes into assemblies. Upon accessing the just-compiled website for the first time, you’ll find that the ASP.NET engine has to go through its compilation as well, which can cause the very noticeable (and much-maligned) initial delay. The .NET Framework 2.0 installation now provides you with several tools to perform this ASP.NET compilation.

Go to the virtual URL, <http://localhost/MyWebApplication/precompile.axd>. Just as with the trace.axd HTTP handler you encountered in “Debugging ASP.NET Web Application Through Trace.axd” in Chapter 3, this virtual URL is also implemented using an HTTP handler. The handler goes through your entire web application and precompiles all the pages to avoid this initial delay. So after publishing your website, remember to run this HTTP handler.

You can also automate the complete compilation using the new `aspnet_compiler.exe` tool. It is located in the .NET Framework 2.0 directory (C:\Windows\Microsoft.NET\Framework\v2.0.*).

Run the utility as follows:

```
aspnet_compiler -v /MyWebApplication
```

This command compiles your entire web application (the same process that VS.NET uses) and precompiles the web application so you won’t experience the initial delay when you first hit your website.

Setting ASP.NET Versions for Web Applications

By default, ASP.NET web applications always use the latest .NET Framework version available on a given machine (as opposed to Windows applications, which always try to use the .NET Framework version with which it was compiled). The problem is that after you install .NET Framework 2.0, all of your web applications will use version 2.0 of the .NET Framework, which might not be what you want.

You can do something about this. The .NET Framework 2.0 installation adds a new tab to the virtual directory's Properties dialog box in IIS. Simply go into IIS, right-click your virtual directory, and choose Properties from the pop-up menu. On the ASP.NET tab you can now explicitly set the version against which you want your web application to run (see Figure 92).

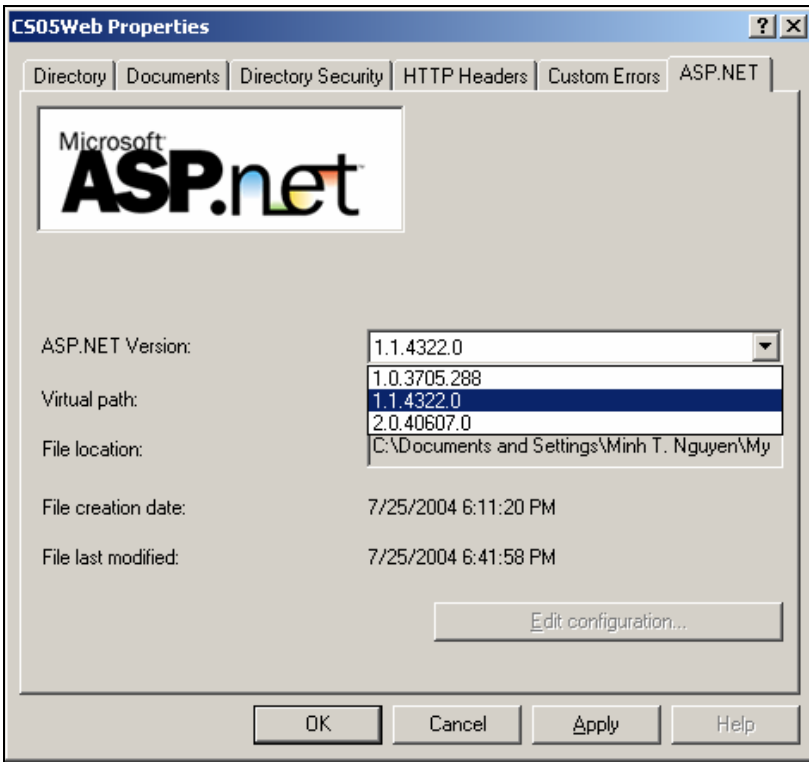


Figure 92 - Setting the ASP.NET version for your web application in IIS

Clearing the Assembly Cache Manually

Whenever web pages embed Windows forms applications on the page, the application is first downloaded to your computer, cached, and run from there. While the application

appears embedded on the page (like Java applets), the code is run from your local computer (with all the necessary high-security settings, of course). When the original author decides to update the application, a new build with a new version number should force Internet Explorer to download the new version.

Sometimes you want to force a fresh download of the new Windows forms application. This typically applies to developer computers, where caching hinders the daily development of special types of web applications. Even clearing the Temporary Internet Files folder doesn't help. Instead, you need to manually clear the download assembly cache, which typically sits in C:\Windows\Assembly\Download. You do this by running the .NET utility called gacutil.exe, which you can find in the directory C:\Windows\Microsoft.NET\Framework\v1.0.3705.

Run the utility as follows:

```
gacutil.exe -cdl
```

Using Unicode for Strings, Labels, and Inline .NET Code

The .NET Framework is fully Unicode-compliant. A string in .NET is a Unicode string, and there is nothing special you need to do to handle Unicode strings. But there is much more to it than that. Not only can a string hold Unicode values, but VS.NET itself is fully Unicode-enabled.

I develop a few Vietnamese applications. Developers often ask me what I do to display a Windows forms application with Vietnamese characters or how I store Vietnamese characters in the database. The truth is that there is nothing special I do to make it happen. It just works.

While designing my Windows forms application in VS.NET editor, I simply turn on the language tool (such as the Global Input Method Editors of Windows or even third-party tools such as VPSKeys for Vietnamese) and set button and label values with Unicode strings.

Even better, because the editor in VS.NET is Unicode-compliant, I can type string literals in Unicode, provided that I save my .cs/.vb file in Unicode-compliant format (Unicode, UTF-8, and others) by selecting File > Advanced Save Options.

To make things even more spectacular, I could even use Unicode for the actual code. This way, code comments, region labels, variable names, class names, even method names can include Unicode characters (see Figure 93).

```

private void button1_Click(object sender, System.EventArgs e)
{
    ThửUnicode("Nguyễn Trí Minh");
}
void Form1.ThửUnicode(string tênVàHọ)
Thử viết tiếng việt
/// <summary>
/// Thử viết tiếng việt
/// </summary>
/// <param name="tênVàHọ">tên và họ</param>
private void ThửUnicode(string tênVàHọ)
{
    // viết tiếng việt bằng unicode
    MessageBox.Show(this, "Xin chào thế giới", tênVàHọ);

    HindiEnum myEnum = HindiEnum.हिन्दी;
    中文
}

```

Figure 93 - Using Unicode in symbol names, strings, comments, and so on

Rethrowing the Same Exception

Whenever an exception needs to be logged, developers tend to use code similar to the following to log an exception without eating it up:

```

Try
{ ... }
catch (Exception ex) {
    Log(ex);
    throw ex;
}

```

This code works, and the exception won't be eaten up because it's being rethrown at the end. However, developers don't often realize that rethrowing the same exception in this way causes .NET to clear the exception's stack trace. When you inspect the exception's stack trace property it will seem as though the exception was originally thrown for the first time inside that catch block.

To throw the same exception correctly in a catch block, simply use “throw;” as follows:

```
Try
{ ... }
catch (Exception ex) {
    Log(ex);
    throw;
}
```

This rethrows the same exception that was just caught without clearing the stack trace.

Afterword

I hope you have enjoyed reading this book and discovering these various tips and tricks, and will start applying them to your work. It might take a while to remember all of them, so I encourage you to refer to this book every once in a while to refresh your memory.

If you have any comments or corrections, or if you know of another trick that you think should be part of this book, please do not hesitate to e-mail me.

English is my third language, and I want to thank my editor, Stefan Gruenwedel, for his valuable help. I also want to thank Janyne Ste. Marie for her great work on the index.

Happy coding,

Minh T. Nguyen
nguyentriminh@yahoo.com

Index

A

- Accessibility, validating, 87–88
- Accessibility Validation dialog box, 87
- Add dialog box and class members, 12
- Add New Item dialog box, 54–55
- Aliases, 42
- App.config, adding, 99–100
- Application configuration file,
 - modifying, 58
- Arguments, command line, 64
- Aspnet_regiis, running, 102–103
- ASP.NET Web applications
 - debugging, 43, 71–73
 - deploying, 59
 - FTP and, 88–89
 - precompiling, 103
 - tracing, 71–73
 - versions, setting, 104
 - VS.NET, attaching, 65–66
- Assemblies
 - .Net Framework
 - inspecting, 100–101
 - obfuscating, 98
 - version, setting, 57–58
 - output path, setting, 57
- Assembly cache, clearing, 104–105
- Assembly names, setting, 54–55
- Attributes
 - localOnly described, 73
 - Obsolete and compiler warnings, 56
- Autocomplete, 16

B

- Blocks, formatting, 12–14, 86
- Bookmarks, 25–26
- Bookmarks window, 26
- Breaking, 67–71

- Breakpoints window, 69, 70, 71
- Browsers, 27–28, 43–44
- Build Action property and file embedding, 34
- Build Events dialog box, 57
- Build menu, 92
- Build order, controlling, 33–34, 56–57
- Build Order tab, 33
- Buttons
 - creating, 47
 - shortcuts, showing, 91–92
- Bytes and file embedding, 34

C

- C#
 - blocks, formatting, 12–14
 - build order, controlling, 56–57
 - code blocks, commenting, 5
 - code snippets, inserting, 81
 - compiling in, 55–56
 - Edit-and-Continue function, 93
 - errors correction suggestions, using, 79
 - event handlers, adding/removing, 50–51
 - formatting, controlling, 84
 - interface methods, implementing, 16–18
 - object member fields, viewing, 2
 - string literals, selecting, 8
- Call stack, customizing, 63–64
- Class Definition view, 102
- Classes
 - adding, 54–55
 - definitions, viewing, 85–86
 - field members, adding, 12
 - names, copying, 45
 - referencing, 101–102
 - resolving, 79
 - searches, 85, 101–102

- Class Library projects, 58
- Class view and searching, 85
- Client-side scripting and JavaScript tags, 28–29
- Clipboard Ring described, 9
- Code
 - behind file, showing, 32–33
 - blocks, commenting, 5
 - collapsing, 6
 - compatibilities, checking, 58
 - definitions, viewing, 85–86
 - error compiler directives in, 55–56
 - refactoring, 76–77
 - and regions, 7
 - snippets, using, 79–81
 - and the Toolbox, 8–9
- Code Comment Web Report, contents of, 3
- Code Definition view, 85–86
- Code Snippet Manager, 81
- Code view and event handlers, 51
- Commands, 42
- Command window
 - accessing, 41
 - drop-down lists, finding, 42–43
 - and Immediate mode, 42
- Comments, adding, 2–5
- Compatibilities, checking, 58
- Compiler functions
 - ASP.NET Web applications, 103
 - automating, 103
 - build steps, setting, 56–57
 - directives in, 55–56
 - obfuscating, 98
 - warnings, generating, 55–56
- Conditions for breakpoints, setting, 69–71
- .config files, managing, 99–100
- Configuration file, modifying, 58
- Control Library projects, naming, 55
- Controls
 - editing, 48, 86
 - locking, 49
 - properties, changing, 48
 - selecting via drop-down list, 51
 - tab order of, 85

D

- DataSet and data display, 94–95
- DataTable and data display, 94–95
- DataTables, viewing contents of, 62–63
- Data visualization, 94–95
- Debugging
 - ASP.NET Web applications, 43, 71–73
 - command-line arguments, placing, 64
 - and Immediate mode, 42
 - macros, 46
 - multiple projects, 66–67
 - next statement, moving, 60
 - and SQL Server, 66
 - and the stack trace, 64
 - variable members, expanding, 93–94
 - variable values, changing, 61
 - VS.NET, attaching, 65–66
- Dependencies, setting, 33–34
- Description, toggling, 49
- Designer view, 52
- Documents, formatting, 12–14. *See also* Text
- Dotfuscator, 98
- Drop-down lists
 - controls, selecting, 51
 - finding, 42–43
 - values, changing, 50

E

- Edit-and-Continue function, 92–93
- Editor, changing, 32
- Elements, UI
 - aligning, 82
 - properties, editing, 83–84
- Encapsulate Field method described, 77
- Errors
 - correction suggestions, using, 79
 - generating, 55–56
- Event handlers, adding/removing, 50–51

Exceptions

- adding/removing, 69
 - rethrowing, 106–107
 - runtime, 20–21
 - types, breaking for, 67–69
- Exceptions dialog box, 68, 69
- Expressions, evaluating, 62
- Extract Interface method described, 77
- Extract Methods process described, 76

F

- Favorites folder window, 43
- Field members and IntelliSense, 15
- Files
- configuration, modifying, 58
 - dragging, 35–36, 40–41
 - embedding, 34
 - extra, showing, 32–33
 - linking, 54
 - macros, saved location of, 46
 - opening, 32, 41
 - split screens, creating, 37–38
 - text file, inserting, 28
 - windows, closing, 91
- Find dialog box, 26
- Font size, changing, 34–35
- Formatting standards defined, 84
- Forms, Windows
- command-line arguments, placing, 64
 - controls, 49, 51
 - downloads, forcing, 105
 - hierarchy, outlining, 29–30
 - menus, adding, 82–83
 - properties, changing, 48
 - views, switching, 20
- FromFirst column, 72
- FromLast column, 72
- FTP and Web projects, 88–89
- Full-screen mode, 44

G

- Global unique identifiers (GUIDs), creating, 19
- GUIDs (Global unique identifiers), creating, 19

H

- Hierarchy, outlining, 29–30
- HTML
- hierarchy, outlining, 29–30
 - rebuilding and, 98
 - tags, collapsing, 7
 - text, pasting as, 12
 - validating for accessibility, 87–88
- HTML Editor
- blocks, formatting, 12–14
 - files, dragging in, 35–36
 - Web controls, editing, 86
- HTML Visualizer, 95

I

- Icons, managing, 38, 40
- IDE restarts and task shortcuts, 5
- IDE settings, importing/exporting, 89–91
- IIS installation, fixing, 102–103
- Images, embedding, 34
- Immediate mode from the Command window, 42
- Incremental searches, 21–22
- Installers, adding, 52
- Instances, searching, 85
- IntelliSense
- across projects, applying, 3–4
 - code snippets and, 79–80
 - and the Command window, 41
 - described, 2
 - and field members, 15
 - and the Immediate window, 63
 - inline strings and, 99

- methods, overriding, 18
- parameter information and, 15

Interfaces, 16–18, 77

Intermediate Language Disassembler, 100–101

J

JavaScript tags, inserting, 28–29

L

Labels and Unicode, 105–106

Languages and the .Net Framework, 88

Language tool and Unicode strings, 105

Line numbers, going to, 20–21

localOnly attribute described, 73

M

Macro Explorer window, 46

Macros

- and the build order, 57
- recording/replaying, 45
- saving/editing/debugging, 46
- shortcuts, assigning, 46–48

Menus

- customizing, 38–40, 46–48
- standard, adding, 82–83

Methods

- definitions, going to, 27
- executing, 62–63
- extracting, 76–77
- interface, implementing, 16–18
- overriding, 18
- and regions, 6
- searches, 85
- stubs, generating, 77–78

N

Namespaces, 45, 54–55

Navigate-Backward button, 27, 28

Navigate-Forward button, 27, 28

Navigation, browser-like, 27–28

.Net Framework

- assemblies
 - inspecting, 100–101
 - obfuscating, 98
 - version, setting, 57–58
- exceptions and, 69
- languages and, 88
- and Unicode, 105–106
- version, setting, 57–58

Next statement, moving, 60

O

Objects, 2, 99

Obsolete attribute and compiler warnings, 56

Open With dialog box, 32

Outlining menu described, 7

Output path, setting, 57

Output window, saving, 73

Override command defined, 18

P

Paragraphs, commenting, 5. *See also* Text

Parameter information and IntelliSense, 15

Placeholders, identifying, 80

Procedures, editing stored, 61

Processes dialog box, 65

Programs, external adding to menus, 39–40

Project properties and command-line arguments, 64

Projects

- dependencies, setting, 33–34

- subsets, building, 92
- Promote Local Variable To Parameter
 - method described, 77
- Properties
 - changing, 48, 83–84, 85
 - command-line arguments, placing, 64
 - creating, 77
- Properties window
 - controls
 - changing, 48, 86
 - locking, 49
 - description, toggling, 49
 - drop-down list values, changing, 50
- Property Editing View button, 83
- Public override command defined, 18

R

- Rectangular sections, creating, 19–20
- Regions
 - collapsed and incremental searches, 22
 - creating, 6–7
 - temporary, 7
- Reminders, listing, 4
- Remove Parameters method described, 77
- Rename method described, 77
- Reorder Parameters method described, 77
- Replace dialog box, 22
- Replacing, 22, 23–25
- Runtime exceptions and line numbers, 20–21

S

- Search dialog box, 22
- Searches
 - class/instance/method, 85, 101–102
 - global, 23–25
 - incremental, 21–22
 - wildcards, 22–23

- words, 21
- Selections, hiding, 7
- Server Explorer and SQL procedures, 61
- Services, Windows, 52, 66
- Shortcuts
 - assigning, 40, 46–48
 - IDE restarts and, 5
 - viewing, 91–92
- Solution Explorer
 - files
 - dragging, 35–36, 40–41
 - embedding, 34
 - linking, 54
 - startup project listing, 66
 - validating HTML for accessibility, 87–88
- Split screen, creating, 36–38
- SQL Server, 61, 66
- Stack frame described, 63
- Stack trace
 - clearing, 106–107
 - debugging, 64
 - described, 64
- Strings
 - inline as object instances, 99
 - iterating over, 98–99
 - literals, selecting in C#, 8
 - Unicode, 105–106
- Symbols, 77, 105–106
- Synchronization, forcing, 29

T

- Tab Index property, setting, 85
- Task shortcuts, placing, 5
- Text
 - autocomplete, 16
 - blocks, formatting, 12–14
 - characters ignored, 10
 - files, inserting, 28
 - font size, changing, 34–35
 - front/end switching, 8
 - pasting as HTML, 12
 - rectangular sections, creating, 19–20

- single line functions, 10
- single words, selecting, 8
- strings, iterating over, 98–99
- transpositions of, 9–10
- words, searching for, 21
- word-wrapping, toggling, 14
- XML editing, 10–11

Text Editor toolbar and bookmarks, 25–26

Text Visualizer, 95

Toolbars

- customizing, 38–39, 46–48
- shortcuts, viewing, 91–92

Toolbox, 8–9

Trace.axd, 71–73

TraceContext class described, 72

Trace logs, saving, 73

U

- UI elements, 82, 83–84
- Undo feature and replacing, 24
- Unicode, 105–106

V

- Validation for accessibility, 87–88
- Variables
 - displaying, 62–63
 - members, expanding, 93–94
 - values, changing, 61
- View mode, customizing, 32
- Views, switching, 20
- VS.NET, attaching, 65–66

W

- Warn() method described, 72
- Watch window, 61, 62
- Web applications
 - debugging, 43, 71–73
 - deploying, 59
 - FTP and, 88–89
 - precompiling, 103
 - tracing, 71–73
 - versions, setting, 104
 - VS.NET, attaching, 65–66
- Web browser, using, 43–44
- Web Content Accessibility Guidelines, 87
- Web controls, editing, 48, 86
- Web pages, 3, 59
- Windows, 36, 91
- Windows Explorer and file dragging, 40–41
- Windows Forms. *See* Forms, Windows
- Windows Services. *See* Services, Windows
- Words. *See* Text
- Word-wrapping, toggling, 14

X

- XML
 - blocks, formatting, 12–14
 - comments, adding, 2–3
 - editing
 - in tabular mode, 11–12
 - in text mode, 10–11
 - nodes, inserting, 11
- XML Visualizer, 95
- XSLT transformations, creating, 3

Visual Studio .NET Tips and Tricks

Covers versions 2002, 2003, and 2005

Visual Studio .NET is one of the most versatile and extensible programming tools released by Microsoft. The number of features and shortcuts available in VS.NET is truly immense, and it grows tremendously with each release. Developers who are unaware of these timesaving features surely miss out on opportunities to increase their programming productivity and effectiveness.

Visual Studio .NET Tips and Tricks explains how to use VS.NET efficiently. Organized into short and easy-to-grasp sections, and containing tips and tricks on everything from editing and compiling to debugging and navigating within the VS.NET IDE, this book is a must-read for all .NET developers, regardless of expertise and whether they program in C#, VB.NET, or any other .NET language. This book covers the Visual Studio .NET 2002, 2003, and 2005 Beta 1 releases.

In this book you'll find the following:

- ▶ More than 120 tips for editing and writing your code, navigating within the IDE, and compiling, debugging, and deploying your application
- ▶ Section dedicated to VS.NET 2005
- ▶ Keyboard shortcuts for the majority of tips
- ▶ More than 90 figures and screenshots

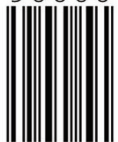
About the Author: Minh T. Nguyen is a website development engineer with Expedia.com. He has worked with Visual Studio .NET since its beta stages and regularly gives workshops and writes articles for the .NET community.

ID: 77316
www.lulu.com
The Marketplace
for Digital Content

ISBN 1-4116-1396-1



90000



9 781411 613966