Visual Studio .NET Tips and Tricks

Abstract

This book provides you with the information you need to effectively use Visual APL in the Visual Studio development environment.

Navigation

To navigate this book, you may either use the tree view on the left to navigate chapter by chapter, section by section, or view the entire book on one page in "Visual Studio .NET Tips and Tricks" in the left hand tree view.

Introduction

Visual Studio .NET comes complete with many features and functions that dramatically increase our efficiency as developers.   As a powerful code editor, compiler, and debugger, it contains features to stress-test, analyze, and optimize your code, and allows easy integration with code documentation, reporting, or smart-device programming, such as the Pocket PC.

Because of the sheer number of features that Visual Studio .NET contains, it is a challenge for .NET developers to become familiar with all of its features, shortcuts, and functionalities. The beginner developer will find a virtual treasure trove of features with which to start, while advanced Visual Studio .NET users will appreciate the many new features and improvements the new Visual Studio .NET 2005 brings.

Most of these features and functionalities are documented, and are accessible through the VS.NET main menu or context menus. But, because of the vast number of features with which VS.NET is equipped, developers don't always know them or use them. This guide should help familiarize developers with the tips and tricks that are at their disposal in this powerful tool and their specific application to the Visual APL language.

Chapter 1: Editing Code

While you create code, there are many techniques and shortcuts that allow you to write and navigate through your code quickly and easily. This chapter introduces many of the tips and tricks you will need for such tasks as code navigation, performing complex find-and-replace searches, and generating code.

1.0 - Inserting Comment Tokens (Ctrl-K,Ctrl-H)

This feature enables you to write a comment and find it again easily.   To see a list of all reminders you placed in your code (see Figure 1):

Without recompiling, select View > Show Tasks > All

To insert task shortcuts in your code:

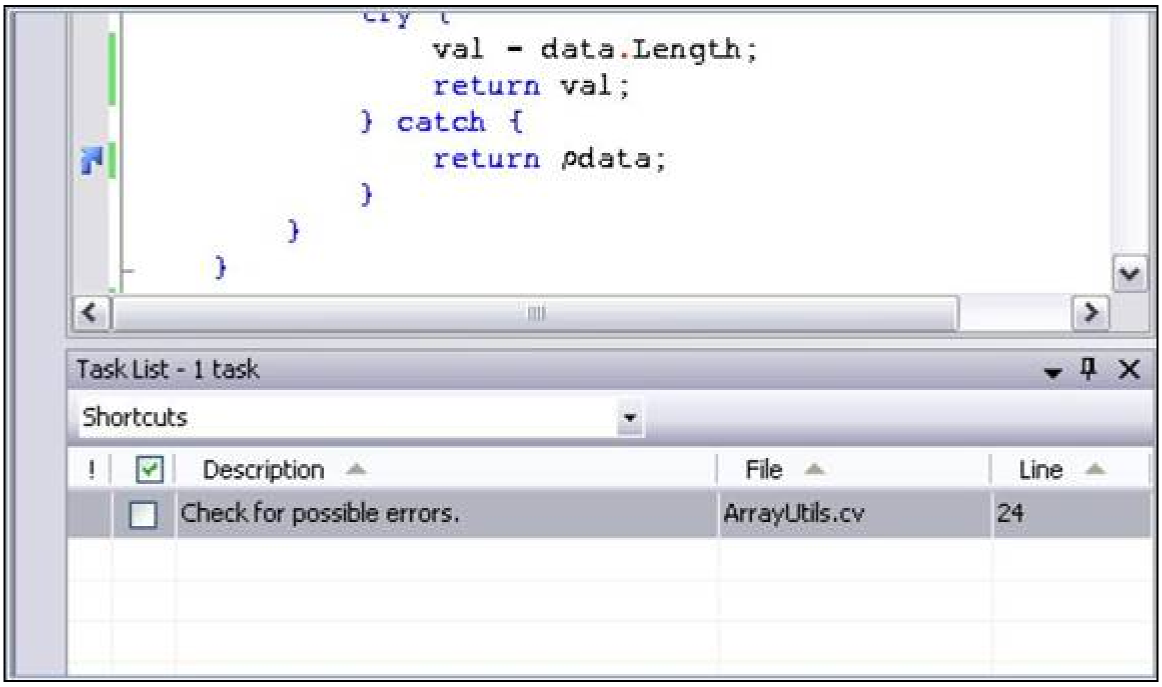
Press Ctrl-K, Ctrl-H.

This marks the current line with a shortcut icon and inserts a clickable shortcut icon in the Task List.

To remove the shortcut:

Press Ctrl-K, Ctrl-H.

These shortcuts survive IDE restarts.



```
        try {
            val - data.Length;
            return val;
        } catch {
            return pdata;
        }
    }
}
```

Task List - 1 task

Shortcuts

!	✓	Description	File	Line
	☐	Check for possible errors.	ArrayUtils.cv	24

Figure 1. Comment Tokens.

1.1 - Commenting Code Blocks (Ctrl-K, Ctrl-C)

One-line comments are extremely useful in explaining seldom used code and assisting in navigation and definition of development projects. To inserted a comment for a code block or segment:

Press the "//" token for Visual APL.

Additionally, Visual APL allows you to comment entire paragraphs and segments. To place a comment in a paragraph or segment:

Select "/#" (and corresponding "#/") tag around the comment.

To quickly comment entire paragraphs:

Select the text.

Click the Comment button (see Figure 2) or

Press Ctrl-K, Ctrl-C.

This comments an entire selection.

To uncomment any selection:

Click the Uncomment button or

Press Ctrl-K, Ctrl-U.

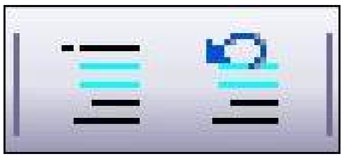


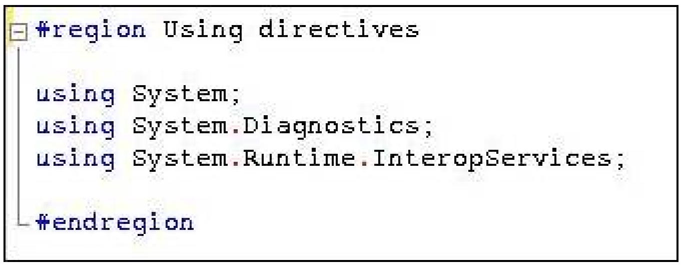
Figure 5 - Comment and Uncomment buttons

1.2 -Creating Regions

The more code you generate, the more difficult it can become to navigate. In addition to selecting classes and their methods from the drop-down lists above the main editor, you can also group your code into logical regions. Regions are extremely helpful for dividing code in logical ways and even commenting it. Regions allow you to collapse code to a single line defining the region and still easily see what is inside it once it is collapsed. They can even be nested. Automatically generated code in VS.NET usually uses this feature, so you may already be familiar with it.

To specify a region:

Insert a #region keyword and a description at the beginning of your segment and a corresponding #endregion keyword at the end of your segment (see Figure 3).



```
#region Using directives

using System;
using System.Diagnostics;
using System.Runtime.InteropServices;

#endregion
```

Figure 3 - Creating regions

The Outlining menu displays various collapse and expand options. To expand and collapse the current region you are in:

Press Ctrl-M, Ctrl-M.

To expand or collapse all regions at once:

Right-click the gray bar to the left of the main editor window.

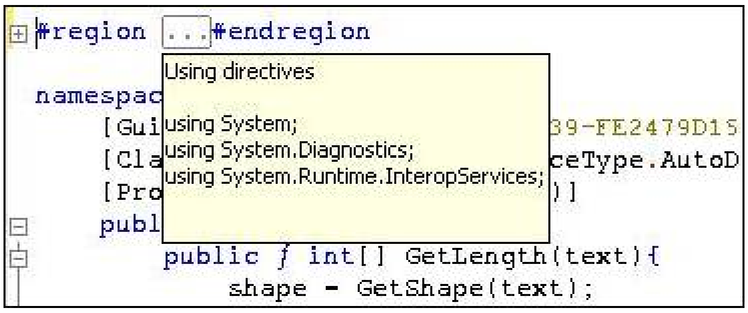
To collapse an individual region:

Click the plus sign next to the #region keyword.

This collapses the code into a single line that shows the region description.

To display the inside of a collapsed region:

Move the mouse over the gray description area (see Figure 4).



```
#region ...#endregion
                Using directives

 namespac       using System;
      [Gui      using System.Diagnostics;           39-FE2479D15
      [Cla      using System.Runtime.InteropServices; ceType.AutoD
      [Pro                                          )]
      publ
         public ƒ int[] GetLength(text){
             shape ← GetShape(text);
```

Figure 4 - Mouse over a region to see its content

You can even drag and drop collapsed regions inside your code. When you paste a collapsed region into a different location, the pasted text is automatically expanded.

1.3 -Hiding Selection by Using Temporary Regions (Ctrl-M,Ctrl-H)

Regions are created automatically for methods, comments, and sections encompassed by the #region compiler directive. In Visual APL and in regular text files, you have the option to create temporary regions around any section without the need for "#region". This is useful when you want to create a region that will not be preserved once your project is closed. In this case, you can temporarily define a region using the following method:

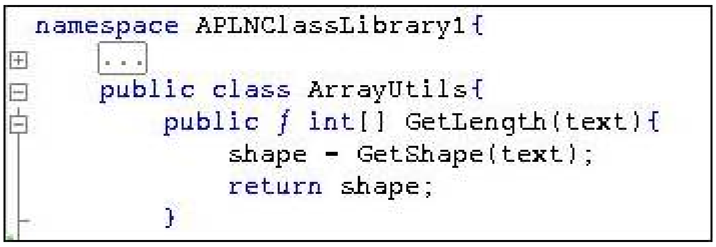
Highlight the section you want to hide and press Ctrl-M, Ctrl-H.

This hides the current selection in a temporary collapsed region (see Figure 5).

To expand a temporary collapsed region:

Press Ctrl-M, Ctrl-M.

Temporary regions are lost after you close a project.



```
namespace APLNClassLibrary1{
    ...
    public class ArrayUtils{
        public ƒ int[] GetLength(text){
            shape ← GetShape(text);
            return shape;
        }
```

Figure 5 - Hiding portions of any text file

For creating regions which are preserved past the closing of your project, see "Creating Regions"

1.4 -Selecting a Single Word (Ctrl+W)

To select a single word when editing code:

Double-click anywhere in the word or just press Ctrl-W.

Double-clicking in a word is a common method used by many word processing and publishing programs to quickly select a word.

1.5 -Placing Code into the Toolbox (Ctrl-Alt-X)

When creating a project, you may want to use certain pieces of code or text again and again. You may have a standard copyright header that you place at the top of each file or a certain line of code to perform a common task. To simplify this repetitive task, you can place it into your Toolbox. The Toolbox is the window that lists all windows or web controls. To place your item into the Toolbox, use the following method:

1. Pull up the Toolbox:

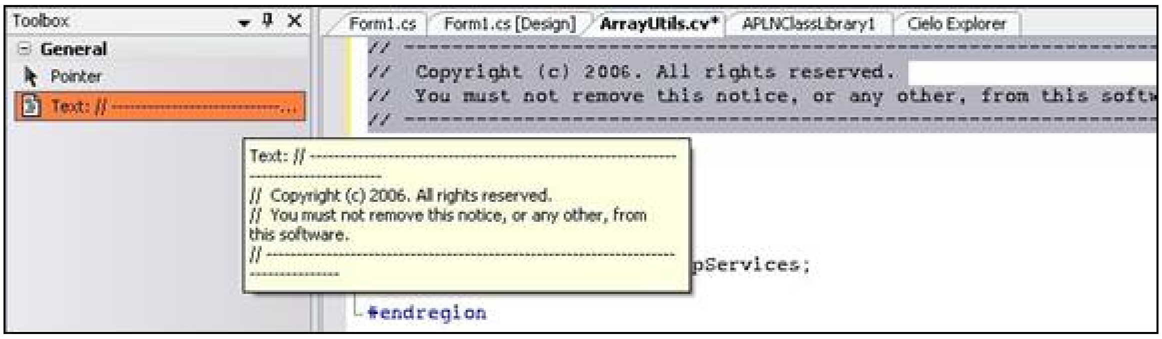
   Press Ctrl-Alt-X.

2. Move the frequently used text or code into the Toolbox:

   Highlight your item and drag the selected text onto the General tab in your Toolbox (see Figure 10).

3. Rename the produced text item in your Toolbox

   Right-click it and choose Rename Item from the pop-up menu.



```
// ------------------------------------------------------------------------
//  Copyright (c) 2006. All rights reserved.
//  You must not remove this notice, or any other, from this softw
// ------------------------------------------------------------------------
```

```
Text: // ------------------------------------------------------------
----------------------
// Copyright (c) 2006. All rights reserved.
// You must not remove this notice, or any other, from
this software.
// ---------------------------------------------------------------
--------------
```

```
pServices;
#endregion
```

Figure 10. Adding text to the Toolbox

To insert an item into your text or code from the Toolbox:

Select the item in your Toolbox, then either:

Drag it into your code window or

Place your cursor in your text where you want to insert the item,
Double-click the entry

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

1.6 - Using the Clipboard Ring (Ctrl-Shift-V)

The Clipboard Ring works like a historical file of the last used text selections that you placed on the Clipboard. Because it preserves many levels of selections, it is useful when you accidentally overwrite the current Clipboard content or when you find yourself needing several different items concurrently.

To use the Clipboard Ring you can either:

Double-click one of the remembered Clipboard items to paste it at the cursor's current location or

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.

To progress one item at a time through the Clipboard Ring:

Select Edit > Cycle Clipboard Ring (Ctrl-Shift-V)

Doing this 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. This method makes it easy to paste specific content in the code editor as it becomes visible during the cycling. Continue cycling through the Clipboard's contents until you find your desired item.

1.7 - Transposing a Single Character or Word (Ctrl-T or Ctrl-Shift-T)

To switch the position of the current characters or words on either side of the cursor you need to use Transpose. This procedure switches the characters or words, then moves the cursor to the right. Transpose is useful if you mistype a word or write a sentence or code segment with words in the incorrect order.

To transpose a single character:

Press 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 one character at a time.

To transpose a single word:

Press Ctrl-Shift-T.

Note: This does more than just swap two adjacent words. VS.NET knows 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.

1.8 - Cutting, Copying, Deleting, and Transposing a Single Line

If you need to cut, copy, delete or transpose an entire line, you can do this easily with one keyboard sequence.

To **copy** the complete, current line to the Clipboard:

Press Ctrl-C (or click the Copy icon) without any text selected.

To **cut** an entire line:

Press Ctrl-X (or click the Cut icon) without any text selected.

This will cut the entire current line and place it in the Clipboard.

To **delete** a single line:

Press Ctrl-L without any text selected.

To **transpose**, or **swap** the current line with the one below it:

Press Alt-Shift-T without any text selected.

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.

1.9 - Formatting Entire Blocks (Ctrl-K, Ctrl-F or Ctrl-K,Ctrl-D)

To apply formatting to an entire selection, there are several useful functions you can use. Uppercasing, lowercasing, or deleting horizontal white spaces are just a few examples.

To access these features:

Select Edit > Advanced

One of the most useful features here is the Format Selection function.

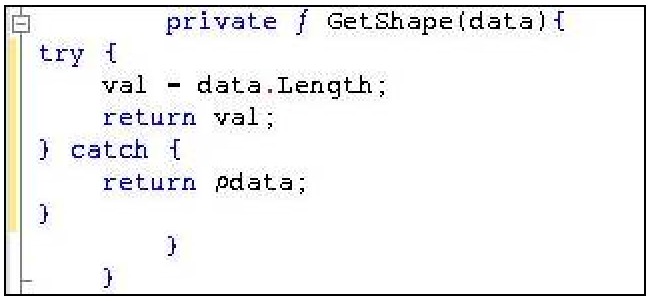
To access the Format Selection Function:

Press Ctrl-K, Ctrl-F,

This feature 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 Visual APL) but Format Selection forces this automatic format (see Figures 11 and 12).

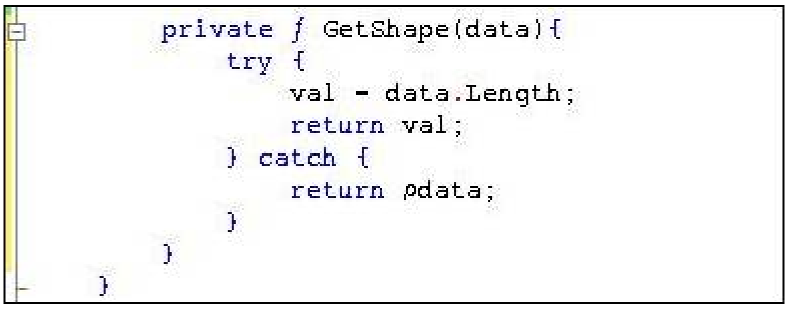
You can also format the entire document. To do this:

Press Ctrl-K, Ctrl-D.



```
        private ƒ GetShape(data){
try {
    val ← data.Length;
    return val;
} catch {
    return ⍴data;
}
        }
    }
```

Figure 11. Before formatting block



```
        private ƒ GetShape(data){
            try {
                val ← data.Length;
                return val;
            } catch {
                return ⍴data;
            }
        }
    }
```

Figure 12. After formatting block

1.10 - Toggling Word-Wrapping (Ctrl-R,Ctrl-R)

To turn word wrapping on and off for the current view:

Select Edit > Advanced

Or use the keyboard shortcut:

(Ctrl-R, Ctrl-R)

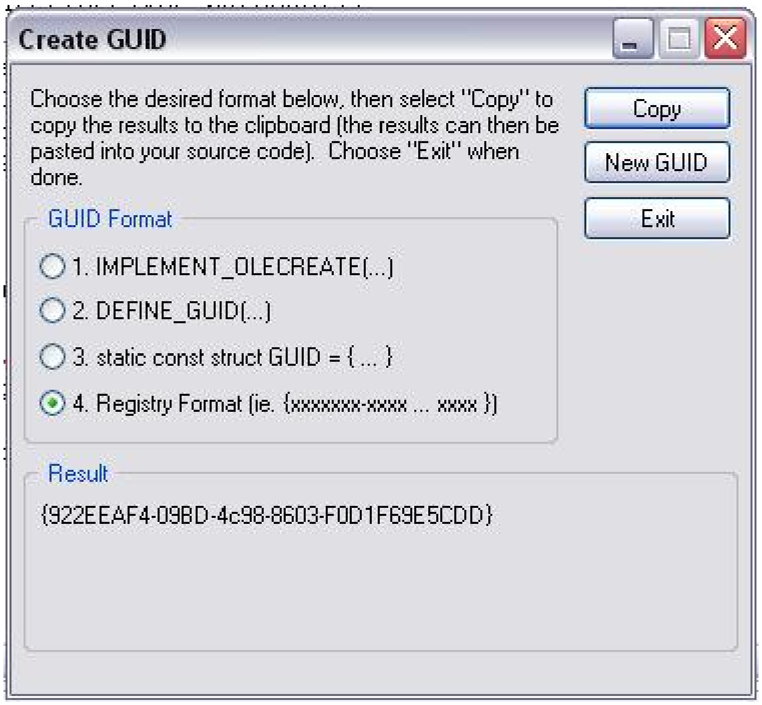
1.11 - Creating GUIDs

As you develop new classes and components, you often need to create Global Unique Identifiers (GUIDs). These are 128-bit values often represented by 32 hexadecimals. In the past, component developers used GUIDs to assign their components with unique names to reduce the likelihood of two components sharing the same GUID. Developers now use GUIDs for anything that requires a unique identifier. GUIDs can be created manually by randomly selecting 32 hexadecimals, but this is somewhat tedious. VS.NET comes with a utility that creates GUIDs for you whenever you need one.

To create a GUID, open the Create GUID dialog box:

Select Tools > Create GUID (see Figure 13).

Here you can generate identifiers in various formats, including common code items often used in COM development.



Choose the desired format below, then select "Copy" to copy the results to the clipboard (the results can then be pasted into your source code). Choose "Exit" when done.

- 1. IMPLEMENT_OLECREATE(...)
- 2. DEFINE_GUID(...)
- 3. static const struct GUID = { ... }
- [✓ 4. Registry Format (ie. {xxxxxxxx-xxxx ... xxxx })]

{922EEAF4-09BD-4c98-8603-F0D1F69E5CDD}

Figure 13. Create GUID tool

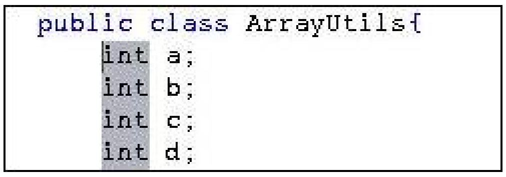
1.12 - Creating Rectangular Selections

To make a rectangular selection of text or code, there is a technique which allows you to do this without including the intervening lines (see Figure 14).

To select a rectangular area:

Press the Alt key while dragging the mouse to select the area.

Manipulating the selection by copying, cutting, or pasting rectangular blocks can be done very easily this way.



```
public class ArrayUtils{
    int a;
    int b;
    int c;
    int d;
```

Figure 14. Rectangular selection

1.13 - Switching Between Views (F7)

For Windows forms, you can easily switch between both views. To toggle between designer and code views:

Press F7 (designer and code)

1.14 - Going to a Line Number (Ctrl-G)

For quick and easy navigation inside your code or text file, you can jump to a particular line.

To go to a specific line number you need to access the go to dialog box. To do this, either:

press Ctrl-G or

double-click the line number status bar at the bottom.

A small dialog box will appears. To jump to a line number:

enter a line number

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 number which exceeds the number of lines in the file will place you at the end of the file. A number that is too low will jump you to the beginning of the file.

1.15 - Searching for a Word

There are several methods for searching for a word inside a file. It is helpful to know all the methods for ease in moving around your file. The following are common methods for finding a word.

1. Access the Find dialog box

   Select Edit > Find

   Enter a term in the Find dialog box.

2. Access the Combo box in the main toolbar next to the configuration drop-down list. To open the combo box and invoke the search function:

   Press Ctrl-D

   Enter or paste a word into this list and press Enter

   Repeat pressing Enter in that drop-down list to find the next match.

3. Select the entire word, or place the cursor somewhere inside the word:

   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 specified in the Find dialog box. Set the options you desire in the Find dialog box first to search correctly (for example, enabling Search Hidden Text to include all collapsed regions in the search area).

1.16 - Performing an Incremental Search (Ctrl-I)

An incremental search 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.

To initiate an incremental search:

Press Ctrl-I

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.

To return to normal editing mode:

Press Escape

In the Cielo Explorer you have to single click with the mouse.

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.

1.17 - Searching or Replacing with Regular Expressions or Wildcards

Regular expressions can look extremely intimidating, but they are extremely powerful tools to find complicated search keys and patterns. Regular expressions is a built in feature that allows you to describe a searchable pattern in terms of wildcards, characters, and groups.

This feature in VS.NET is often overlooked by many developers. To access this feature, bring up the Search or the Replace dialog box:

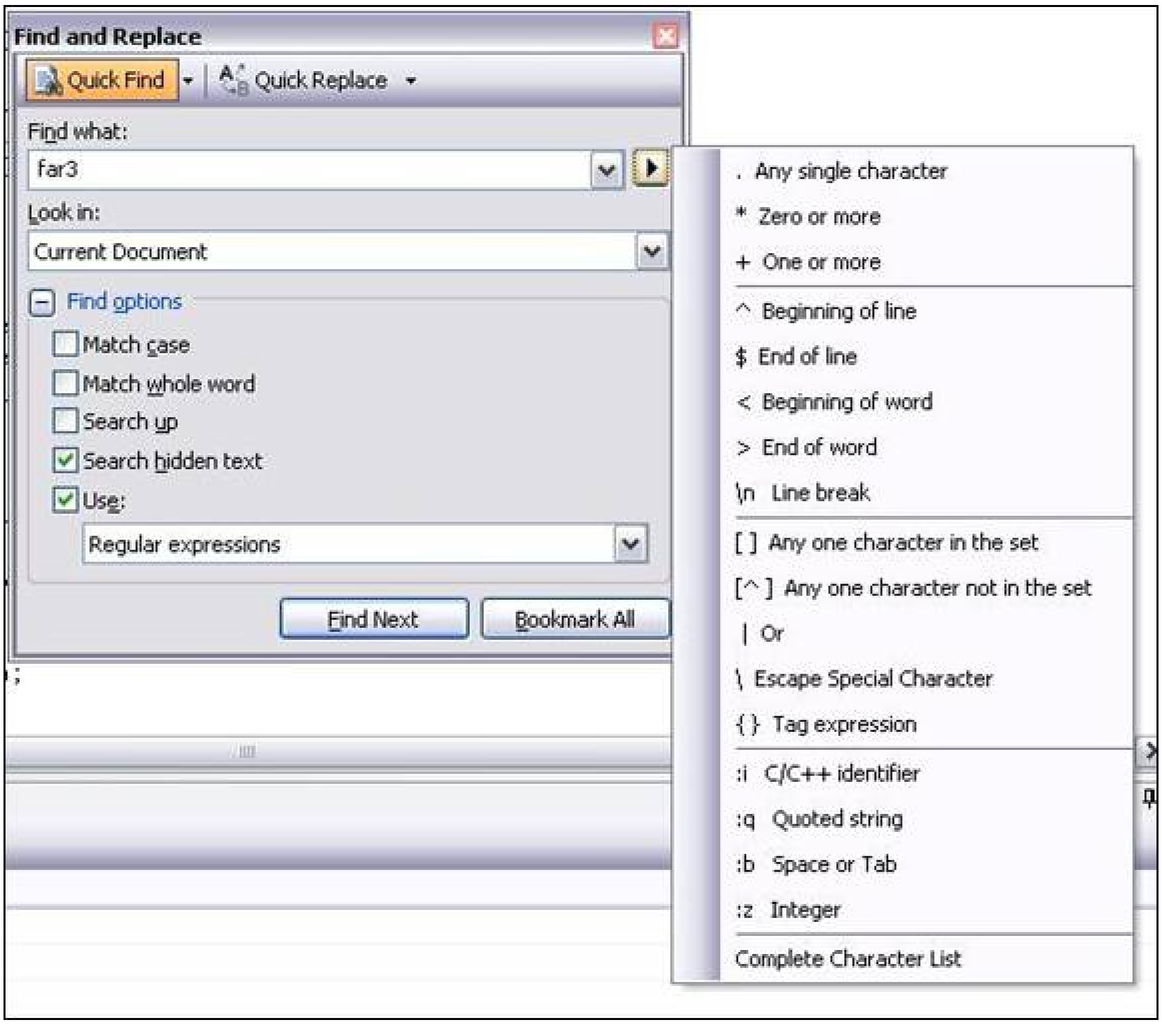
Press either Ctrl-F or Ctrl-H, respectively

Note: Besides the regular options to refine your search, the last check box allows you to define your search based on regular expressions or wildcards.

You can use either of two modes. To use Regular Expression mode, specify the expression using a similar notation you are accustomed to with the System.Text.RegularExpressions namespace.

To see a list of possible constructs that you can insert into your regular expression:

Click the arrow button next to the Find What field (see Figure 15).


Find and Replace

Quick Find | Quick Replace

Find what: far3

Look in: Current Document

Find options
- [ ] Match case
- [ ] Match whole word
- [ ] Search up
- [✓] Search hidden text
- [✓] Use: Regular expressions

Find Next | Bookmark All

.	Any single character
*	Zero or more
+	One or more
^	Beginning of line
$	End of line
<	Beginning of word
>	End of word
\n	Line break
[ ]	Any one character in the set
[^]	Any one character not in the set
|	Or
\	Escape Special Character
{ }	Tag expression
:i	C/C++ identifier
:q	Quoted string
:b	Space or Tab
:z	Integer

Complete Character List

Figure 15. Use regular expressions.

To use 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 when developing programs based on regular expressions.

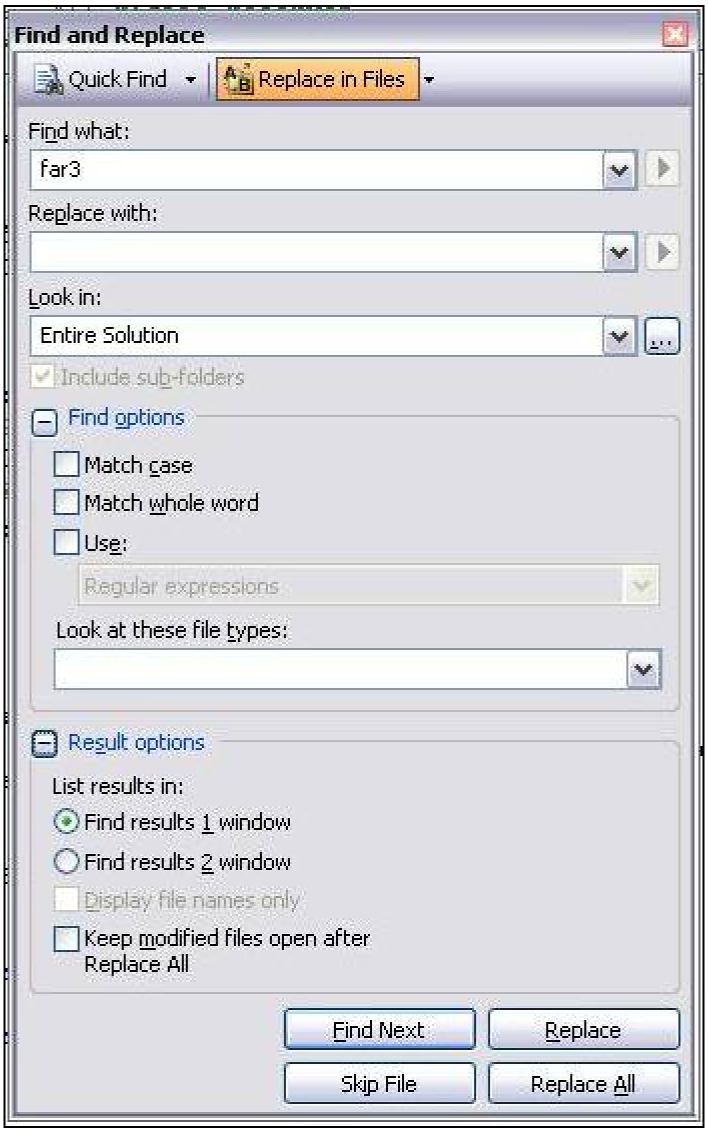
1.18 - Global Search or Replace (Ctrl-Shift-F or Ctrl-Shift-H)

The global search and replace feature in VS.NET spans entire projects and solutions. This is similar to normal search and replace dialog boxes, except that you can specify the scope of the search or replace action over multiple files.

To bring up the global search or global replace dialog box:

Press Ctrl-Shift-F or Ctrl-Shift-H, respectively

This feature allows you to perform a global search and replace in just the current document, the current project, the entire solution, or any open documents (see Figure 16). You can also filter which files you want to search based on wildcards.



Find what: far3

Look in: Entire Solution

[✓ Find results 1 window]

Figure 16. Global search and replace.

Once the search or replace action is started, VS.NET searches all specified documents and modifies them if required. The global replace, will also prompt you 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, global replace will automatically save the modified files and make this a permanent action.

To immediately stop a global search or replace anytime:

Press Ctrl-Break.

Once a search or replace action is completed, a list of occurrences that have been found will be displayed in the “Find Result” dialog box.

To iterate over the Find Result list:

Press F8 or navigate to an occurrence by double-clicking it.

If that occurrence is currently located in a collapsed region, you can expand it. To automatically expand the region:

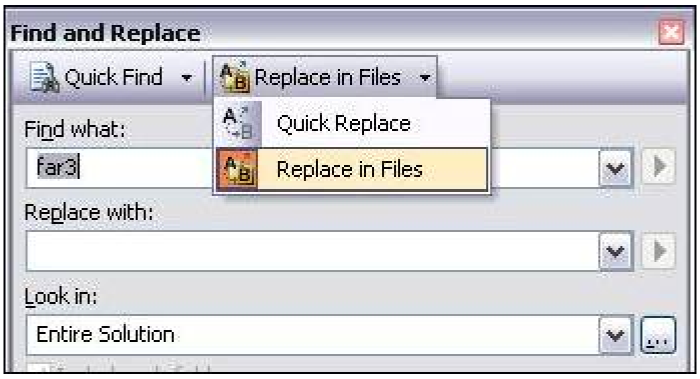
Double-click the same find result in the list again.

On initiating a new find or replace action, VS.NET clears this window to fill the list with the new results. If you want to keep the results of the previous search and output the result in a second window:

Check the Display in Find 2 option in the search dialog box

You can then tab between both result sets.

All find/replace functionalities are included in a single dialog box (see Figure 17), you can also access the global find/replace functionalities using the drop-down list at the top. All shortcuts remain the same.


Find and Replace

Quick Find | Replace in Files (menu: Quick Replace, Replace in Files)

Find what: far3

Replace with:

Look in: Entire Solution

Figure 17. Global replace in files

1.19 - Using Bookmarks

Bookmarks enable you to return quickly to a given page or section of your code or file. When you determine critical sections of your programming that you want to return to frequently, instead of scrolling to these places, bookmark those lines.

To place a bookmark, first, make the bookmark toolbar visible:

Right-click any existing toolbar and select Text Editor from the pop-up menu.

Click on the blue flag icon in Text Editor toolbar.

Another method which can be used to place a bookmark:

Press Ctrl-K, Ctrl-K.

This second method not only makes a bookmark visible on the left side of the code, but you can now jump quickly among other bookmarks. To jump to the other bookmarks you can either:

Click the appropriate flag buttons on the toolbar (see Figure 18) or

Press Ctrl-K, Ctrl-P (for the previous bookmark) or Ctrl-K, Ctrl-N (for the next bookmark).

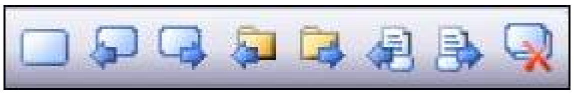


Figure 18. Bookmark toolbar

To clear all bookmarks:

Press the Clear Flag icon or

Press Ctrl-K, Ctrl-L.

The Find dialog box in VS.NET allows you to bookmark all matching occurrences as follows:

Click the Mark All button.

As part of VS.NET 2005 considerable support for bookmarks, you can also have the option of moving to the next or previous bookmark within the same file as follows:

Press the appropriate buttons on the bookmark toolbar (see Figure 28).

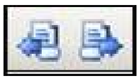


Figure 28 - Move to bookmarks in the same file in VS.NET 2005

You can also name your bookmarks by first opening a new Bookmarks window:

Press Ctrl-K, Ctrl-W or

Select View > Other Windows >Bookmark Window.

This displays all the bookmarks that you have created (see Figure 19).

To jump to a bookmark's location:

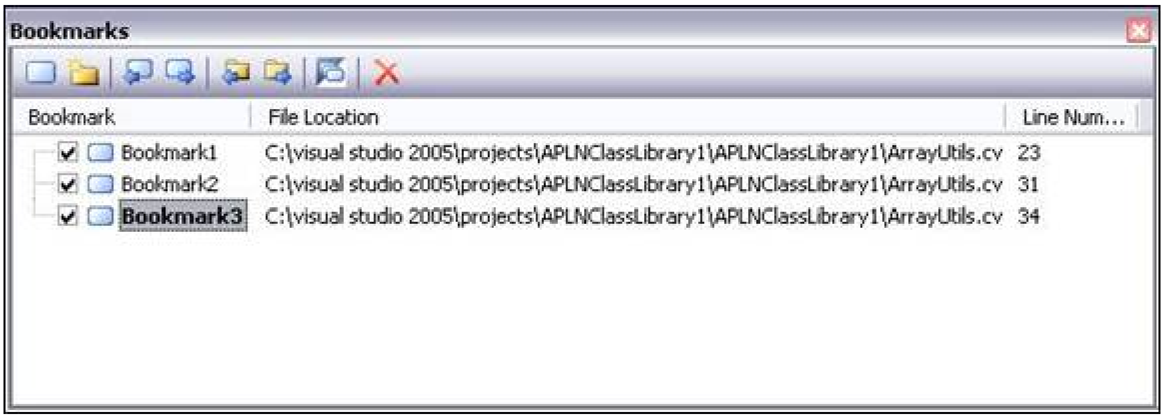
Double-click the bookmark.

To rename a bookmark:

Press F2 or

Right-click the bookmark and use the Rename context menu item.

You can categorize your bookmarks and organize them into folders. You can also, jump to the next or previous bookmark within the same folder. To perform any of these functions, simply select the appropriate icon on the toolbar.



Bookmark	File Location	Line Num...
✓ Bookmark1	C:\visual studio 2005\projects\APLNClassLibrary1\APLNClassLibrary1\ArrayUtils.cv	23
✓ Bookmark2	C:\visual studio 2005\projects\APLNClassLibrary1\APLNClassLibrary1\ArrayUtils.cv	31
✓ Bookmark3	C:\visual studio 2005\projects\APLNClassLibrary1\APLNClassLibrary1\ArrayUtils.cv	34

Figure 19. Manage your bookmarks in the Bookmarks Window.

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

1.20 - Using Browser-Like Navigation (Ctrl -, Ctrl Shift -)

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 located to the right of the Undo and Redo buttons (see top left of Figure 20). You can also access them in the View menu.

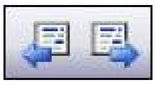


Figure 20. Navigate buttons

Similar to a web browser, VS.NET keeps a history of your recently accessed locations. 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 easily return back to the last edit location as follows:

Click the Navigate-Backward button.

These Navigate buttons have pre-assigned shortcuts. To Navigate back:

Press Ctrl-Hyphen

To Navigate forward:

Press Ctrl-Shift-Hyphen.

1.21 - Inserting External Text File

A common method of inserting code fragments involves opening a file in Notepad and copying the code from there. To bypass this step of opening and closing Notepad you can:

Select Edit > Insert File as Text from within the code editor.

Chapter 2: Exploring the IDE

Visual Studio .NET is an easily customizable feature-rich Integrated Development Environment (IDE). It allows a developer quick access to commonly used commands and activities which enable you to control and modify your project and solutions. This chapter covers a range of topics such as: the Solution Explorer; window positioning; managing macros; modifying menu items and other tips and tricks useful for in navigating inside the IDE.

2.0 - Setting Project Dependencies

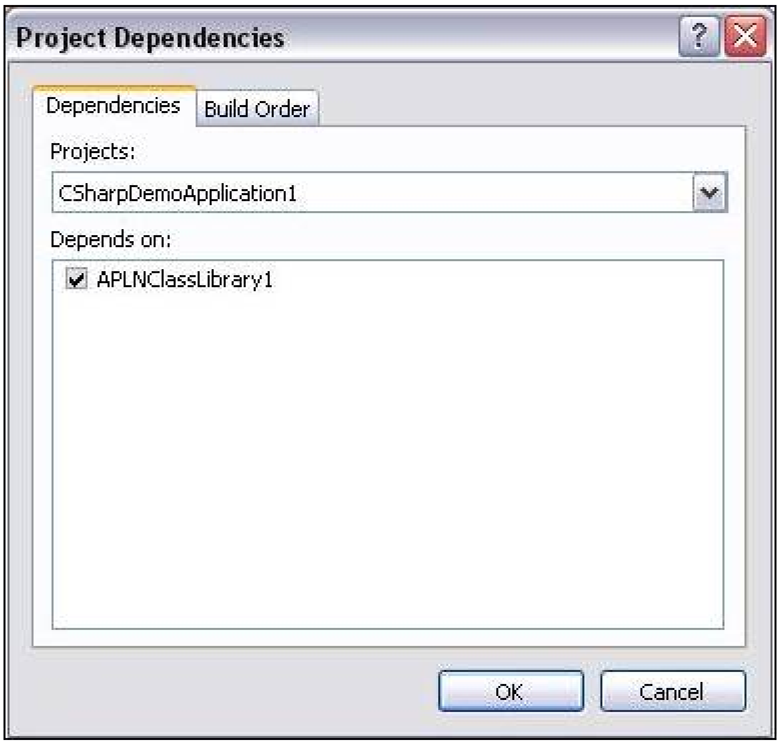
In a large solution with multiple projects and custom build events, it is often necessary to control the build order for your projects. VS.NET has the capability to figure out which project needs to be built first by analyzing the references of each one. The first project built is normally the one referenced first. This algorithm is based on your set project references for your projects.

VS.NET also allows you to compile a certain project before another one without having project references. This is accomplished from a pop-up menu that allows you to choose Project Dependencies. To designate the order in which projects will be built:

Right-click your project that needs to be built last and choose Project Dependencies from the pop-up menu.

Set manual dependencies on other projects by check-marking them.

This will ensure that the checked projects will be built before the current project (see Figure 21). A drop-down menu for the current project allows you to switch to another project's dependencies.



Projects: CSharpDemoApplication1

Depends on: [✓] APLNClassLibrary1

Figure 21. Setting project dependencies manually

VS.NET prevents you from creating circular references or modifying dependencies that resulted from adding project references. To verify the build order at any given time:

Click the read-only Build Order tab.

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

2.1 - Embedding Files As Resources

Embedding files as resources allows you to embed any given file directly into your produced assembly. For instance to display a company logo on your Windows application, you could produce a regular Windows assembly and link to an external image that you send along with your application. You can also embed the image right into the assembly you produce. This enables you to avoid shipping the external image and, more importantly, prevents the possibility of these two files becoming separated.

To embed a file as a resource, it must first be included in your solution. You can then select the file in the Solution Explorer and change the Build Action property in the Properties window. The build action tells the compiler what 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("mylogo.bmp");
```

After this retrieves the 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. This enables you to embed any file type: sound and movie files, PDF files, or even another assembly.

2.2 - Changing the Font Size of IDE Windows for Demos

It is a common practice when demonstrating VS.NET or your code, to increase the font size of the text editor so that everyone in the audience can easily see the demonstration. The font size can be easily increased by a couple of methods:

Select Tools > Options > Environment > Font and Colors > Size.

This works great, except that the text in the Output windows, Solution Explorer, Class view, Macro Explorer, or in the file tab titles can still be very hard to read.

Control the size of the text in these elements as follows:

In the same settings window, the first drop-down list reads Show Settings For. Change it to read Dialogs and Tools Windows.

Set the font and the size here in this window.

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.

To increase the font of the Output window:

Change Show Settings For to Text Output Tools Windows.

To reset any of these settings to their default installation values:

Click the Use Defaults button.

Note: This button applies only to the currently selected item in the Show Settings For drop-down list, so repeat this step for every setting that you want reset back to the default settings.

2.3 - Dragging Files to Obtain a Full Path

A useful feature of VS.NET is the ability to 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.

2.4 - 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. To move any window in VS.NET:

Drag 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), an outline appears, allowing you to preview the result before dropping the window.

To dock and undock windows:

Double-click 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, the preview outlines are too confusing to make this an intuitive interface. If you have moved the windows positions and would like them reset, you can always reset all windows positions to their installation defaults:

Select Tools > Options > Environment > General > Reset Windows Layout.

With VS.NET 2005, you can also reset the windows positions. To do this:

Select Window > Reset Windows Layout.

One aspect of moving windows around is the ability to create a split screen. Use the following steps to split the editor into two vertical screens complete with their own set of file tabs (see Figure 23):

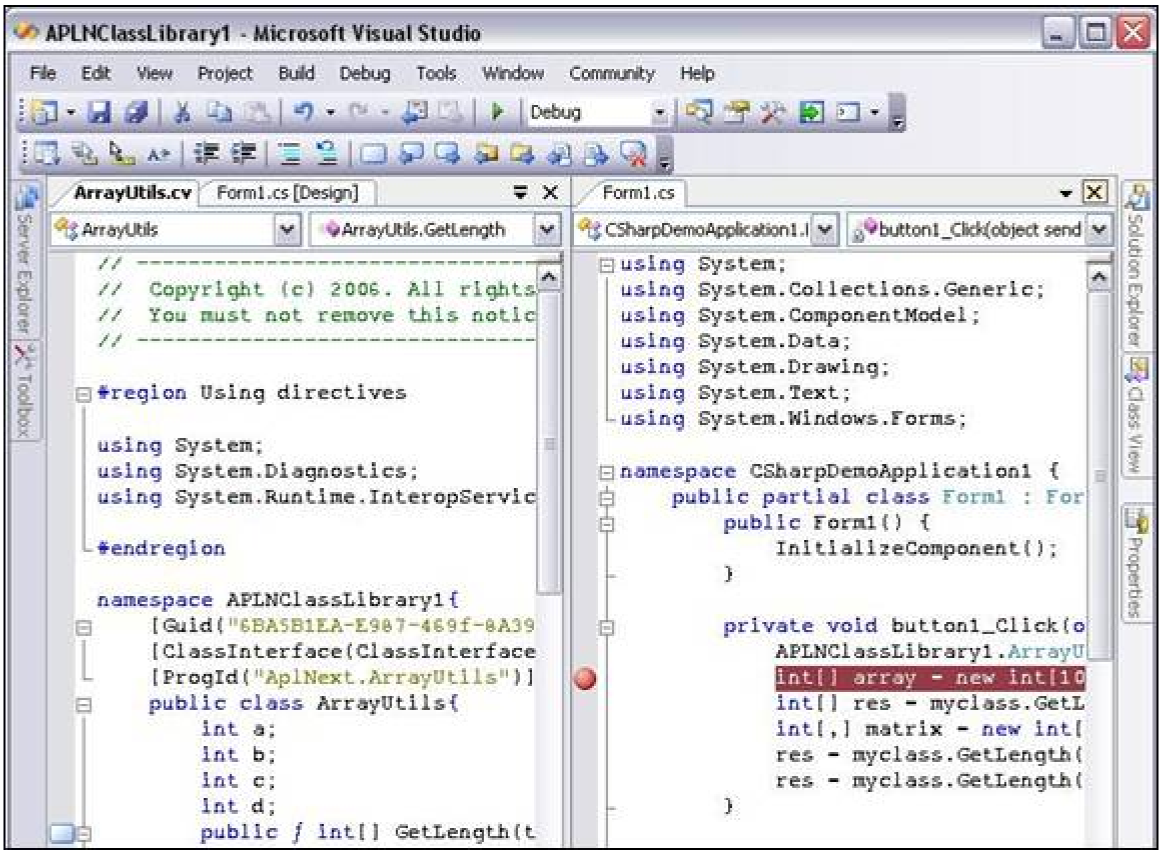
Drag the tab of any open file and move it to the right of your editor (to the left of where the Solution Explorer usually resides).

This docks your selected file to the right and splits the editor into two vertical screens.

To close vertical split mode either:

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.



```
// ---------------------------------
//  Copyright (c) 2006. All rights
//  You must not remove this notic
// ---------------------------------

#region Using directives

using System;
using System.Diagnostics;
using System.Runtime.InteropServic

#endregion

namespace APLNClassLibrary1{
    [Guid("6BA5B1EA-E987-469f-8A39
    [ClassInterface(ClassInterface
    [ProgId("AplNext.ArrayUtils")]
    public class ArrayUtils{
        int a;
        int b;
        int c;
        int d;
        public ƒ int[] GetLength(t
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace CSharpDemoApplication1 {
    public partial class Form1 : For
        public Form1() {
            InitializeComponent();
        }

        private void button1_Click(o
            APLNClassLibrary1.ArrayU
            int[] array = new int[10
            int[] res = myclass.GetL
            int[,] matrix = new int[
            res = myclass.GetLength(
            res = myclass.GetLength(
        }
```

Figure 22. Vertical split

To create a horizontal split screen:

Drag a file tab to the bottom of your editor.

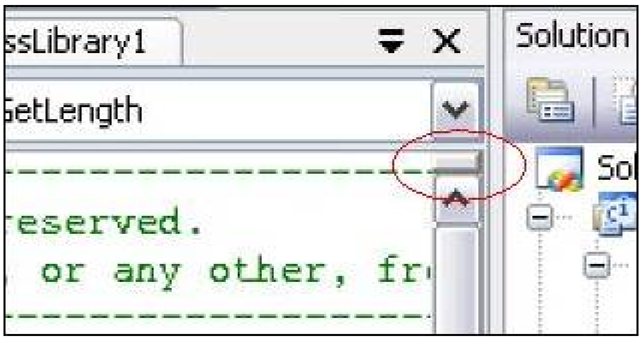
2.5 - Creating Split Screens in the Same File

The “Moving Any Window Around” trick described in “Moving Any Window Around” 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? To do this:

> select Window > Split

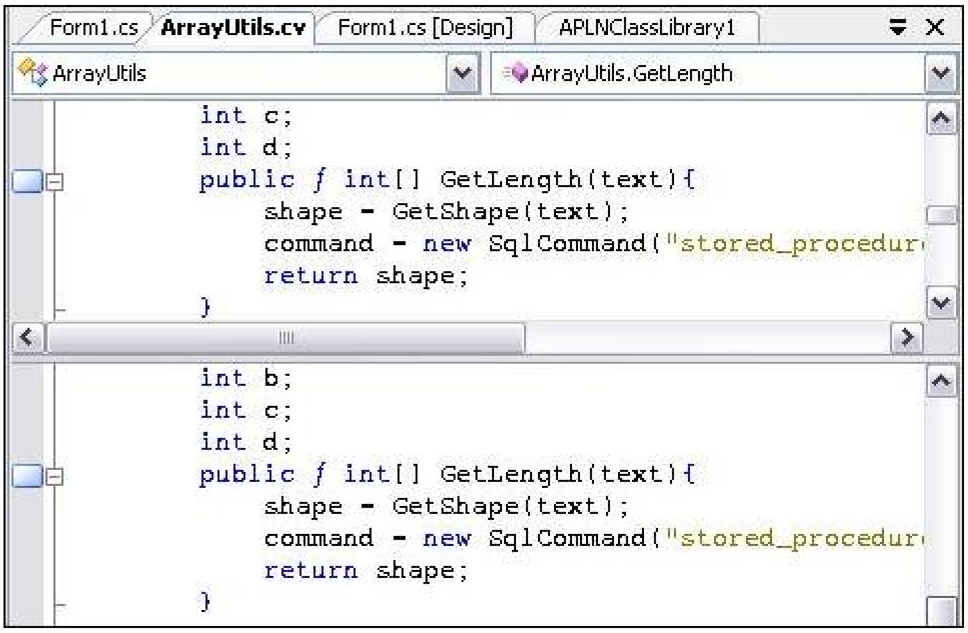
The horizontal divider can also be generated using a faster method:

> Move your cursor right above the vertical scrollbar of the main editor. There is a very thin, short, rectangular-shaped divider (see Figure 23).
>
> Place your mouse over that divider, 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 24).



```
-------------------
eserved.
 or any other, fr
-------------------
```

Figure 23. Horizontal split divider



```
        int c;
        int d;
        public ƒ int[] GetLength(text){
            shape = GetShape(text);
            command = new SqlCommand("stored_procedur
            return shape;
        }
```

```
        int b;
        int c;
        int d;
        public ƒ int[] GetLength(text){
            shape = GetShape(text);
            command = new SqlCommand("stored_procedur
            return shape;
        }
```

Figure 24. Split document.

To move the divider back to the top of your editor window:

> Select the divider bar and slide back into its original position.

2.6 - Customizing the VS.NET Menu and Toolbars

The VS.NET menu can be customized in a variety of ways. You can add and remove commands as well as reorder them. To customize the menu and toolbars:

> 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 and will show 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. To copy icons from other menu items:

Right-click a menu item with the desired icon.

Choose Copy Button Image from the pop-up menu

Right-click the menu item you want to modify.

Choose Paste Button Image from the pop-up menu.

To add other commands to a menu:

Drag a command from the Command tab directly into the VS.NET menu.

Note: 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 25).

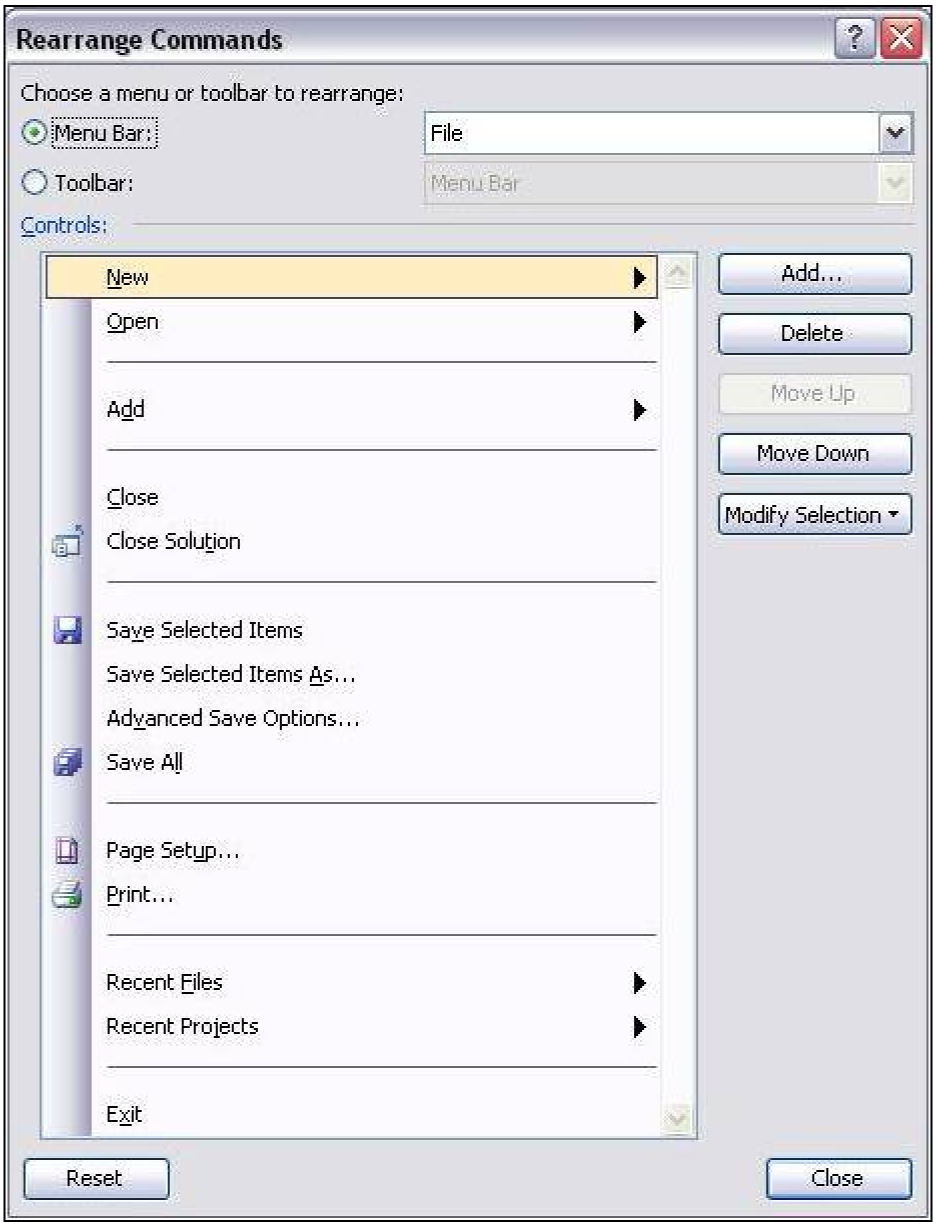


Figure 25. Customize menus using the rearrange commands

2.7 - Dragging Files from Windows Explorer into VS.NET

Visual Studio .NET completely supports file drag (and drop) actions. It allows you to 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.

To drag files from Windows Explorer into VS.NET if you don't have enough screen space:

Drag the file into the Windows taskbar at the bottom of your screen

Pause for a few seconds over the taskbar for VS.NET. The pause brings VS.NET into focus.

Drop the file into the appropriate location.

2.8 - Using Full-Screen Mode (Ctrl – Shift – Enter)

Full-screen mode allows you to hide virtually everything except the main editor, where the entire screen shows the main view. To enter full-screen mode:

Select View > Full Screen or

Press Ctrl-Shift-Enter.

The main menu is still visible at the top, and a floating button that closes full-screen mode is also available. To hide the Close Full Screen mode button—you need to 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.

2.9 - Copying the Fully Qualified Name of a Class

The Class view is a hierarchical view of all your classes and namespaces in your solution. To display this view:

Select View > Class View or press Ctrl-Shift-C.

To go to any class and its members and navigate to the member definitions:

Double-click on the desired item.

Another useful feature allows you to extract the full namespace of any class or member:

Highlight the class or the class member.

Press Ctrl-C.

This copies the complete namespace of the selected item to the Clipboard. This feature comes in handy when you have a complex or deep namespace structure.

To paste the namespace into the VS.NET code editor, there is no need to copy it to the Clipboard first. Use the following method:

Drag a class or member of a class from the Class view directly into your code

Watch VS.NET paste the complete namespace and member name there.

2.10 - Changing Properties of Several Controls

When designing your Windows forms, you can 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. To select a series of controls either:

Hold down Ctrl or Shift when selecting controls or

Draw 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 is useful 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.

Change it back to an empty string by pressing Delete.

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.

2.11 - Locking Controls

When laying out windows controls on Windows forms, you can easily move the controls around or create event handlers by simple dragging and double-clicking. However, this simplicity has its drawbacks as you can move things around accidentally very easily. This can cause problems if you have already finished designing your Windows forms. In order to prevent this from happening, you can lock your form.

To lock the position of your controls on your form:

While in the Designer view, right-click anywhere on your form

Choose Lock Controls from the pop-up menu (see Figure 26).

You still have the ability to add event handlers and modify a control's appearance, but you can no longer accidentally move or resize a control. To indicate that it is locked and unmovable, a thin, black outline appears around each selected control.

To return to the Designer view:

Right-click your form and choose Lock Controls from the pop-up menu again.

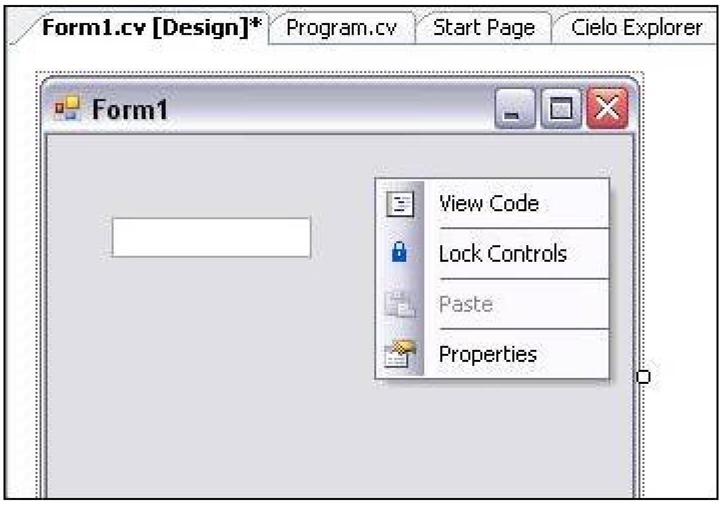


Figure 26. Lock controls in a form

2.12 - Toggling the Description in the Properties Window

The Properties window not only displays all properties of a selected control, but 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. To turn off the Description box panel:

Right-click the property name.

Choose Description from the pop-up menu.

To turn it back on use the same method.

2.13 - 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 selection. For instance, the FormBorderStyle property of a Windows form only accepts None, FixedSingle, Fixed3D, FixedDialog, Sizable, FixedToolWindow, and SizableToolWindow. To select the appropriate item:

Open the drop-down list.

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).

This trick can be extremely useful when switching Boolean values because a double-click changes the value quickly from True to False, or vice versa.

2.14 - Adding and Removing Event Handlers Through the IDE

Adding default handlers through the IDE is quite easy. In most cases, you only need to double-click a control which creates the necessary code for the default event handler.

Adding and removing non-default events handlers is still easy, but, in Visual APL, it 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 Visual APL is to use the Properties window:

Select the control

Click the Events button in the Properties window (the yellow 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.

In addition, the event handler fields are clickable (see Figure 27).

To create an event handler:

Double-click an empty field.

Choose which event you want to subscribe to.

To hook an event handler which is already written, to an event:

Use the drop-down button next to the selected field that automatically lists all matching event handlers.

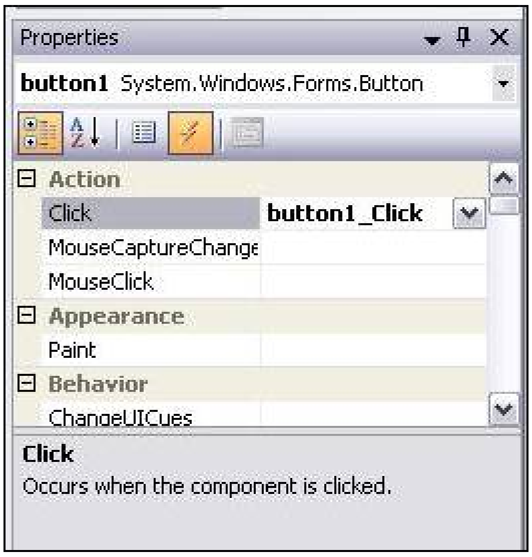


Figure 27. Setting events

To delete an event handler:

Delete the value in the event field.

This also removes the event handler subscription you have in the InitializeComponents() method.

2.15 - Selecting Control Through a Drop-Down List

When there are many controls on a Windows form, it can become a challenge to find a specific control, and 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.

To select a specific control:

Select the drop-down list that appears right above the Properties window.

Select the desired control.

Note: 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.

Chapter 3: Compiling, Debugging, and Deploying

Not only is VS.NET a great editor, it is also a powerful compiler, debugger, and profiler. It allows you to precisely control your compilation procedure and provides the features which are absolutely essential in to locating and fixing a bug: analyzing your code, attaching to running processes that you want to debug, and changing code and variables at runtime. This chapter covers topics that you need to know when it comes to compiling and debugging your programs.

3.0 - Setting the Default Namespace and Assembly Name

Following 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. It places it, by default, in the top-level namespace, which usually means the name of your assembly. To set the default namespace when you create new projects:

Select Project > Properties > Application.

Specify the default namespace in the Default Namespace field.

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.

3.1 - Generating Compiler Warnings Through the Obsolete Attribute

A commonly used way to display warnings in VS.NET at compile-time is to set an Obsolete Attribute to a method. Throughout the product development cycle, occasionally certain methods become obsolete. Sometimes the old method is not useful anymore. It may have become inefficient, or has been replaced by another method. If you can't modify those methods, will need to write another implementation of the method using a slightly different name or signature. To maintain 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()...
```

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 affect the compilation behavior in any way. You also must activate the Task List to see these warnings. Unlike warning compiler directives, the warning only appears if there is code that tries to invoke the obsolete method. These warnings will never appear if you don't refer to these methods anywhere in your code.

3.2 - 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.
These are the default settings. To specify another directory where you want to place the produced assemblies and external files:

Select Project > Properties > Build for Visual APL 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, if you want, you can 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.

3.3 - Setting the .NET Framework Version for Your Assembly

A great side-by-side installation feature of the .NET Framework is the ability to have multiple versions of the .NET Framework installed on a given computer, without any of them interfering. By default, all non-web applications use the .NET Framework with which they were compiled (if available), 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). What you need to do is:

Insert the appropriate Configuration/startup/supportedRuntime and Configuration/startup/requiredRuntime XML tags in the configuration file

Set its version attribute to the specific .NET Framework version.

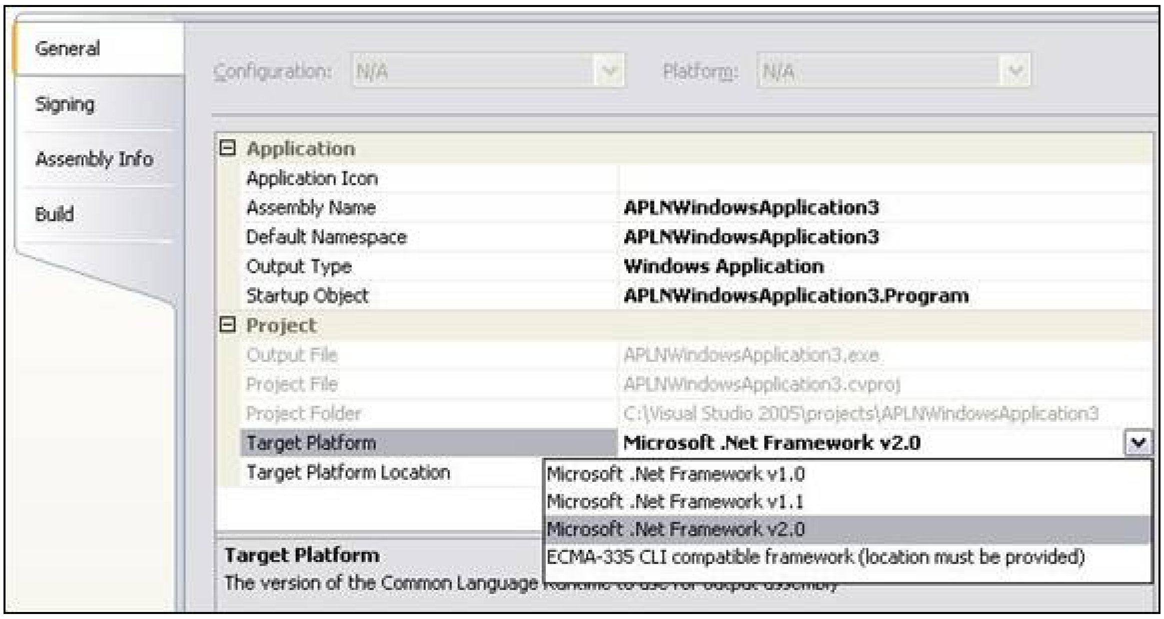
This enables you to force a Windows application to use an older version of the .NET Framework.

This configuration modification is easy in VS.NET. To set this for Visual APL:

Select Project > Properties > General > Target Platform.

Set the supported and required runtime versions for your assembly (see Figure 28).

To verify that your assembly is picking up the correct version, check the Version property in the .NET Framework class System.Environment.



Configuration: N/A   Platform: N/A

Application	
Application Icon	
Assembly Name	APLNWindowsApplication3
Default Namespace	APLNWindowsApplication3
Output Type	Windows Application
Startup Object	APLNWindowsApplication3.Program
Project	
Output File	APLNWindowsApplication3.exe
Project File	APLNWindowsApplication3.cvproj
Project Folder	C:\Visual Studio 2005\projects\APLNWindowsApplication3
Target Platform	Microsoft .Net Framework v2.0
Target Platform Location	

Microsoft .Net Framework v1.0
Microsoft .Net Framework v1.1
Microsoft .Net Framework v2.0
ECMA-335 CLI compatible framework (location must be provided)

Target Platform
The version of the Common Language Runtime to use for output assembly

Figure 28. Choosing the target runtime.

Note: Supporting the 1.0 Framework, or even version 1.1 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 2.0–specific features.

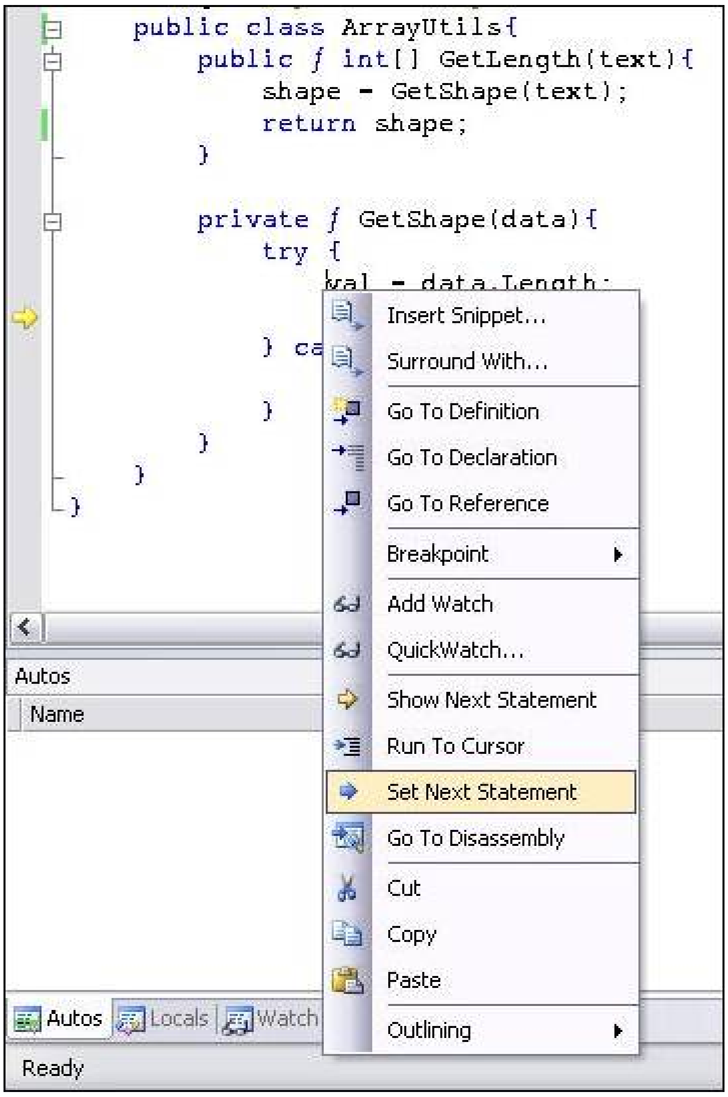
3.4 - Moving the Next Statement During Debugging

When stepping through your program one line at a time, you may need to jump a few lines back. To do this:

Right-click an arbitrary line

Choose Set Next Statement from the pop-up menu (see Figure 29).

This forces the debugger to jump to that line and continue debugging "normally" from there.



```
    public class ArrayUtils{
        public ƒ int[] GetLength(text){
            shape = GetShape(text);
            return shape;
        }

        private ƒ GetShape(data){
            try {
                val = data.Length;

            } ca

            }
        }
    }
}
```

Figure 29. Set Next Statement

To jump back, and also jump forward in and out of control statements:

Drag the yellow arrow to any line.

Note: You cannot jump out of the current stack frame, so you are limited to moving inside your current method.

In addition, moving the current execution line can bring your program into states that under normal execution could not occur. Still, it's an extremely useful feature to rerun certain code lines without restarting your debugging session.

3.5 - Changing Variable Values in the Watch Window

In addition to moving the next-statement pointer, you can change variable values at debug-time. In the process of debugging your application, you may have moved your variables of interest into the Watch window (probably 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 accomplished by entering the new value.

Note: You need to change the internal tick value of DateTime variables.

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. You can only change a reference variable to another reference variable of the same type (or its derived types).

Note: This can bring your program into states that under normal conditions would never be encountered.

3.6 - 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. In addition to 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. To view, edit, and modify stored procedures:

Right-click any stored procedure.

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 quite useful for its designed purpose of viewing, editing, and modifying stored procedures.

To execute stored procedures at design-time:

Right-click a stored procedure.

Choose Run Stored Procedure from the pop-up menu.

VS.NET inspects your stored procedure's parameter list. If necessary, the Run Stored Procedure dialog box is displayed:

Enter each parameter's value.

Execute your stored procedure and see the results.

3.7 - 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. While debugging your program, you step into methods and methods within methods. The stack trace keeps track of all these different levels.

To see the current stack trace:

Select Debug > Windows > Call Stack or

Press Ctrl-Alt-C,

Each method invocation is displayed on its own line, including the line-number and argument values. Each new method invocation is known as a stack frame.

The stack trace has been around in Visual Studio for a long time and is a widely known tool. The advantage of the stack trace window is that it allows you to identify how you get to the current execution point and also inspect the arguments that have been passed to the methods.

To make VS.NET immediately jump to the method invocation on a particular level of your program:

Double-click any line in the stack trace.

A relatively unknown aspect of the stack trace is that you can customize the Call Stack window. To do this:

Right-click the call stack.

Customize what appears there (see Figure 30) according to your requirements.

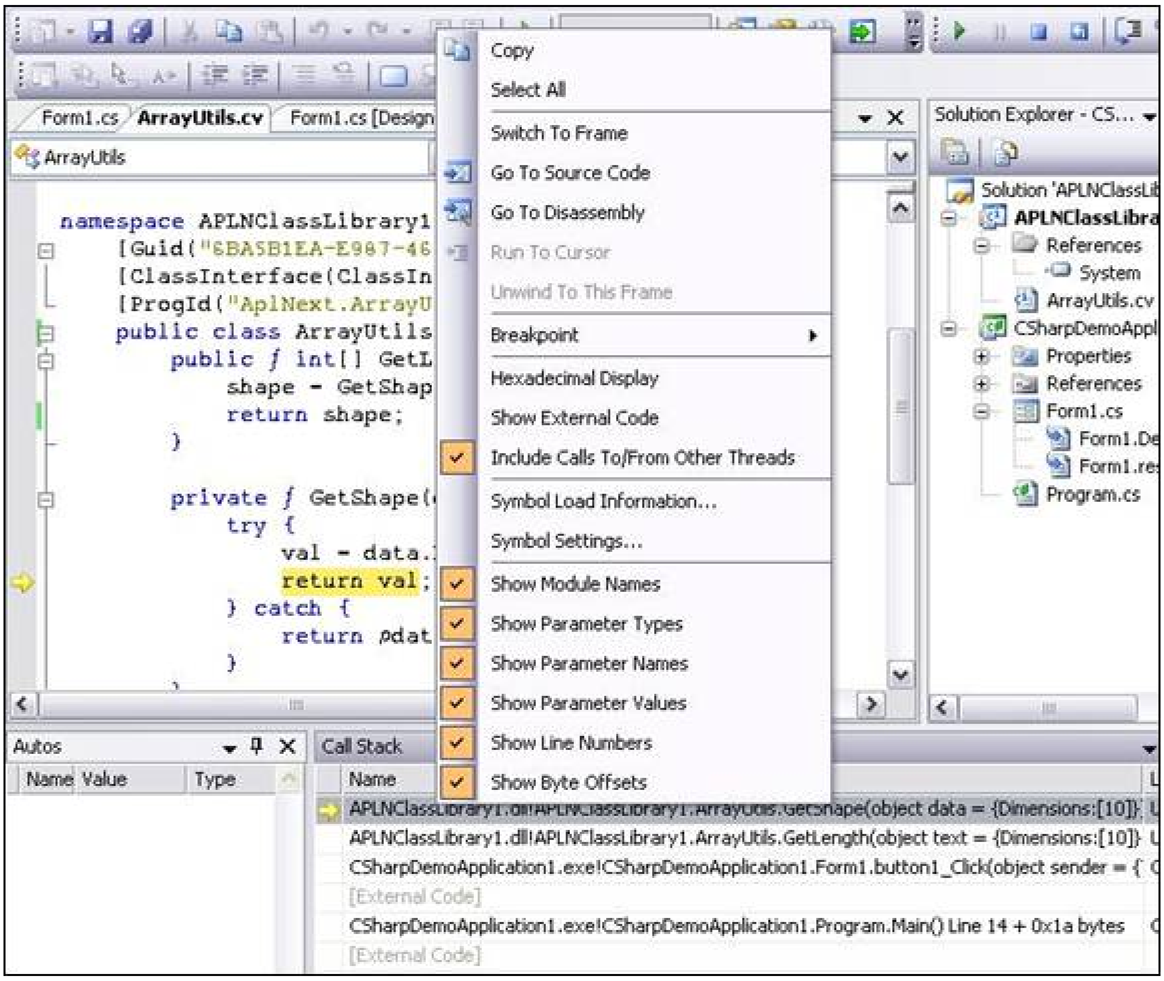
In addition, you can send the information regarding a single method invocation to a coworker:

Copy a stack frame to the Clipboard by pressing Ctrl-C.

To send your coworker the entire call stack:

Press Ctrl-A first, or

Before copying the selection to the Clipboard, Choose Select All from the context menu that appears after you right-click.



```
namespace APLNClassLibrary1
    [Guid("6BA5B1EA-E987-46
    [ClassInterface(ClassIn
    [ProgId("AplNext.ArrayU
    public class ArrayUtils
        public ƒ int[] GetL
            shape = GetShap
            return shape;
        }

        private ƒ GetShape(
            try {
                val = data.
                return val;
            } catch {
                return ρdat
            }
```

```
APLNClassLibrary1.dll!APLNClassLibrary1.ArrayUtils.GetShape(object data = {Dimensions:[10]}
APLNClassLibrary1.dll!APLNClassLibrary1.ArrayUtils.GetLength(object text = {Dimensions:[10]}
CSharpDemoApplication1.exe!CSharpDemoApplication1.Form1.button1_Click(object sender = {
[External Code]
CSharpDemoApplication1.exe!CSharpDemoApplication1.Program.Main() Line 14 + 0x1a bytes
[External Code]
```

Figure 30. Customize the CallStack

3.8 - Attaching VS.NET to an Already Running Process

To instruct VS.NET to debug your program, you first are telling it to build your project (if necessary) then start 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, assuming that the project was built with the debug release. To debug your program:

Press F5

There some cases, where you need, or want, to debug an already running process that has not been started with VS.NET you must:

Open the project for the program that is already running.

Select Debug > Attach to Process

A list of all active processes on your machine is displayed.

From the Processes dialog box, select the process you are interested in debugging and click Attach.

3.9 - Debugging Several Projects Inside the Solution

In a multi-project solution, VS.NET will start the project that you have marked as the "startup project." That project is indicated 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.

It is possible to debug those programs anyway, using the instructions in, "Attaching VS.NET to an Already Running Process."

To instruct VS.NET to start a project and attachs itself to a specific program:

Right-click your project

Select Debug > Start New Instance from the pop-up menu.

You can repeat these steps several times to start multiple instances of your program and still debug them all. This is useful in debugging multi-threaded client-server scenarios.

Tell VS.NET which projects you want to start on each new debug session (see Figure 31):

Right-click your solution

Choose Set Startup Projects from the pop-up menu.

By default, VS.NET uses the Single Startup project, where only one project is started.

To start more than one project:

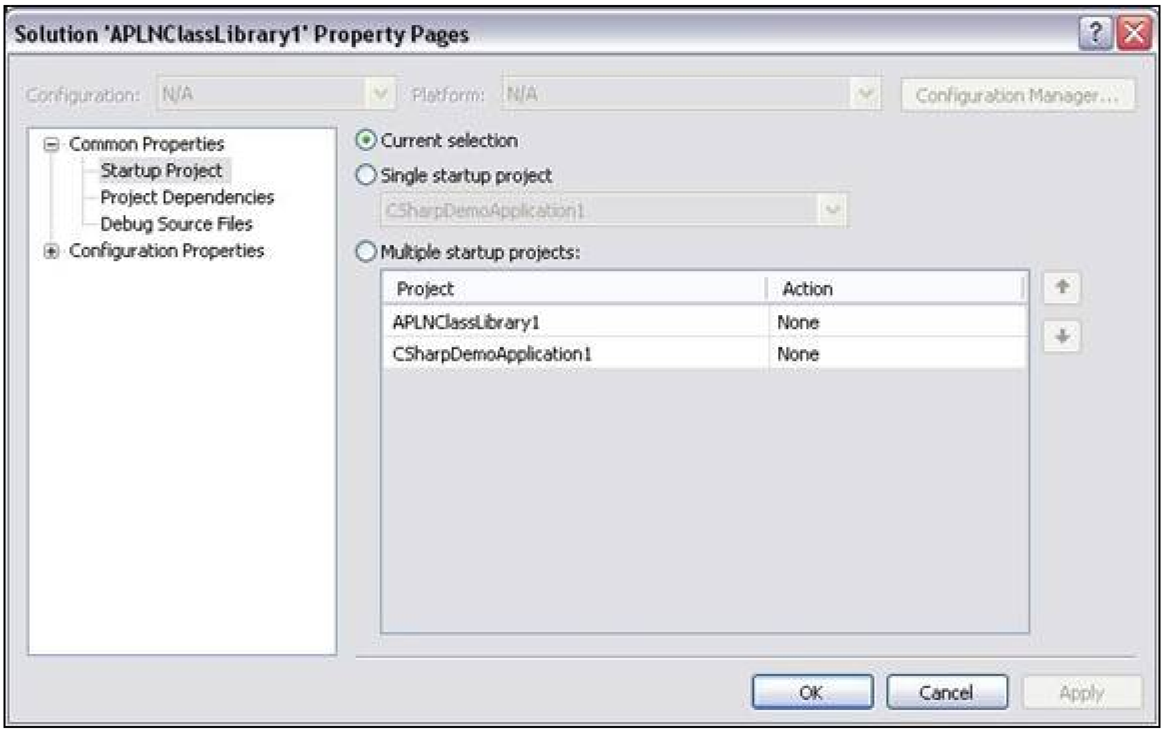
Switch to Multiple Startup Projects

Modify the Action value for each property: None, Start, or Start Without Debugging.

To control the order by which these multiple projects start:

Click 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.


Solution 'APLNClassLibrary1' Property Pages

Configuration: N/A   Platform: N/A   Configuration Manager...

- Common Properties
  - Startup Project
  - Project Dependencies
  - Debug Source Files
- Configuration Properties

(●) Current selection
( ) Single startup project: CSharpDemoApplication1
( ) Multiple startup projects:

Project	Action
APLNClassLibrary1	None
CSharpDemoApplication1	None

Figure 31. Multiple startup projects

3.10 - 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.

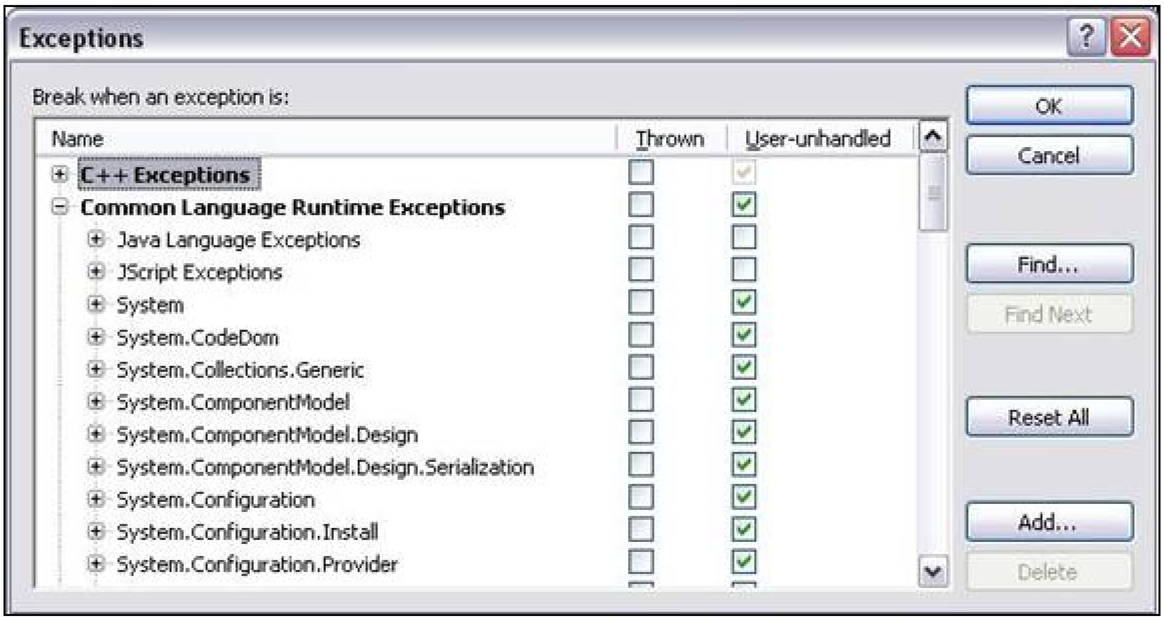
To specify the exceptions that developers are interested in defining, there is a setting in VS.NET. To utilize this setting:

Select Debug > Exceptions, or

Ctrl-Alt-E.

A tree view–style list of all possible exceptions that VS.NET can hook into (see Figure 32) will be displayed.

In addition to the many Common Language Runtime exceptions, you can hook into C++, Native Run-Time checks, and Win32 exceptions.



Break when an exception is:

Name	Thrown	User-unhandled
⊞ C++ Exceptions	☐	✓
⊟ Common Language Runtime Exceptions	☐	✓
⊞ Java Language Exceptions	☐	☐
⊞ JScript Exceptions	☐	☐
⊞ System	☐	✓
⊞ System.CodeDom	☐	✓
⊞ System.Collections.Generic	☐	✓
⊞ System.ComponentModel	☐	✓
⊞ System.ComponentModel.Design	☐	✓
⊞ System.ComponentModel.Design.Serialization	☐	✓
⊞ System.Configuration	☐	✓
⊞ System.Configuration.Install	☐	✓
⊞ System.Configuration.Provider	☐	✓

Figure 32. Break on specific exceptions

From this list you are able to:

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. In the predefined .NET exceptions, you can hook into your own .NET exceptions.

To specify the complete, fully qualified string that defines your .NET exception, for example, "MyCompany.MyProduct.MyBusinessException":

Click the Add button in the Exceptions dialog box.

3.11 - Breaking Only When Certain Conditions Apply (Ctrl – Alt – B)

A heavily used method to add or remove exceptions is 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.
To access these conditions:

Set your breakpoint using your normal method.

Right-click your breakpoint.

From the context menu, choose Condition (Figure 33) to get to the Breakpoints window (Figure 34).

Two buttons stand out at the bottom of the Breakpoints window. To specify a condition under which a breakpoint becomes active:

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 in the following way:

Sset your breakpoint as you would normally do.

Right-click your breakpoint.

From the context menu, choose Condition to get to the same screen (see Figure 62).

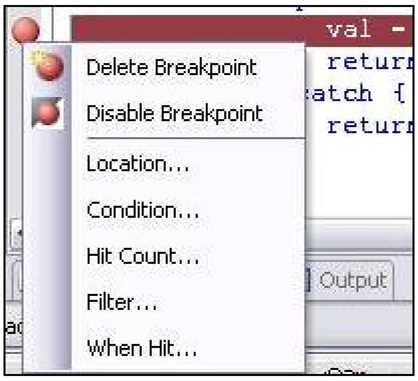
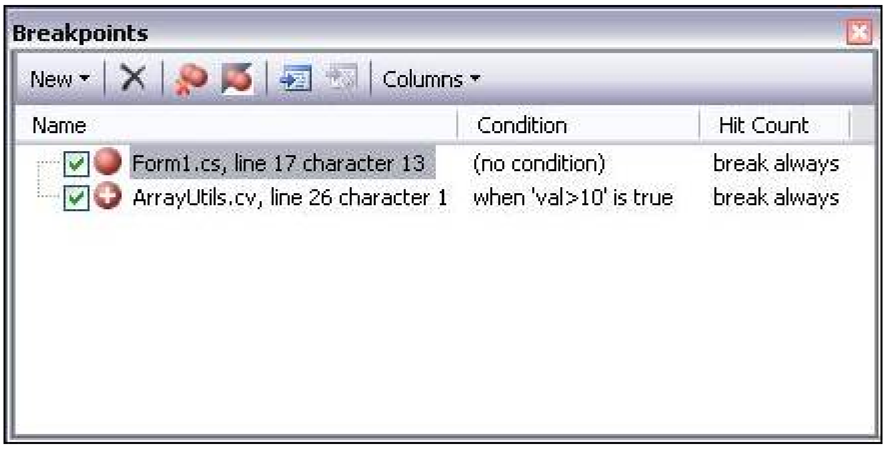


Figure 33. Set breakpoint condition.



Name	Condition	Hit Count
Form1.cs, line 17 character 13	(no condition)	break always
ArrayUtils.cv, line 26 character 1	when 'val>10' is true	break always

Figure 34. Breakpoints Window

To see and modify the condition in the Breakpoints window:

Open that window by selecting Debug > Windows > Breakpoints or

Pressing Ctrl-Alt-B.

A list of all breakpoints that you have set, along with their conditions will be displayed.

Note: 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.

3.12 - 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.

To save all these trace logs:

Press Ctrl-S to save the entire output to a file.

To search through the Output window:

Press Ctrl-F

You can even apply some of the other editor tips and tricks 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).

3.13 - Aligning UI Elements Automatically

If you are positioning UI elements in a Windows form, you have probably noticed various colored lines that appear on the form as you move or resize elements (see Figure 35). 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.

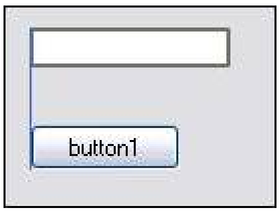


Figure 35. Align lines

To position UI elements without snapping to these colored lines:

Press Alt to turn off automatic alignment temporarily.

To switch back to the grid where all UI elements are aligned to a predefined grid:

Select Tools > Options > Windows Forms Designer > General and change LayoutMode back to SnapToGrid.

Note: 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.

3.14 - Adding a Standard Menu Strip

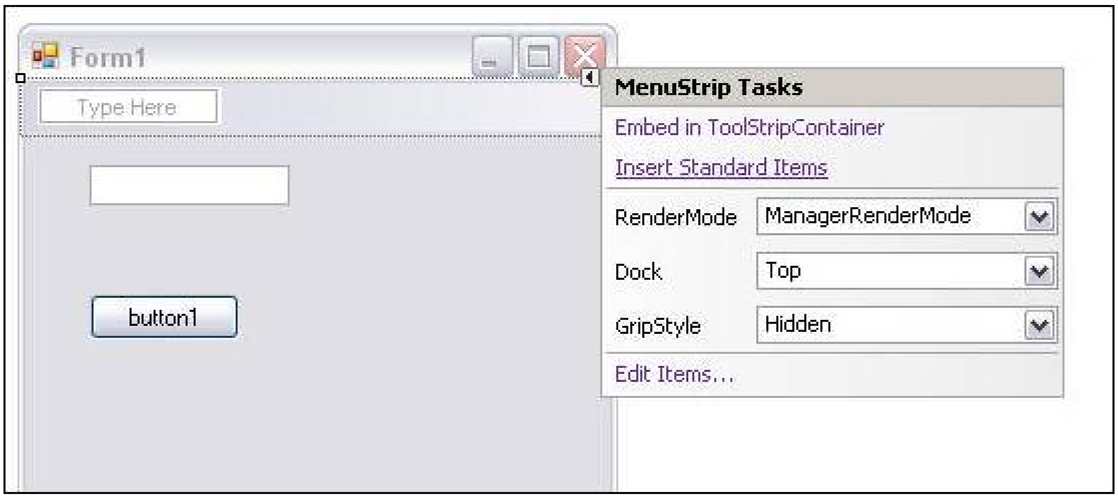
Standard Windows applications use a common set of top-level menu items. In most cases, they are File, Edit, Tools, and Help. VS.NET 2005 allows you to add these default menu items to your own Windows forms applications.   To add your own menu strip:

Drag a MenuStrip to your Windows form.

With the MenuStrip selected, the description panel below the Properties window shows an Insert Standard Items link (see Figure 74):

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.



RenderMode: ManagerRenderMode
Dock: Top
GripStyle: Hidden

Figure 36. Menu strip standard items

3.15 - 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.

Figure 37 - Tab Order button on the left of the Layout bar

VS.NET 2005 introduces a new way to set the tab order: the Tab Order button on the Layout bar (see Figure 37).

Click the Tab Order button to display the tab index for all UI elements on the form.

You now see all the tab indices.

Click repeatedly on each UI element to set the tab order in linear fashion.

The first element you select is given 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

Press the Escape key.

3.16 - Importing and Exporting IDE Settings

VS.NET is an extremely powerful tool with many things in the IDE that you can customize to suit your specifications. Because you will become accustomed to your particular settings, moving from one machine to another can cause problems if you are not able to move your IDE settings along with you.

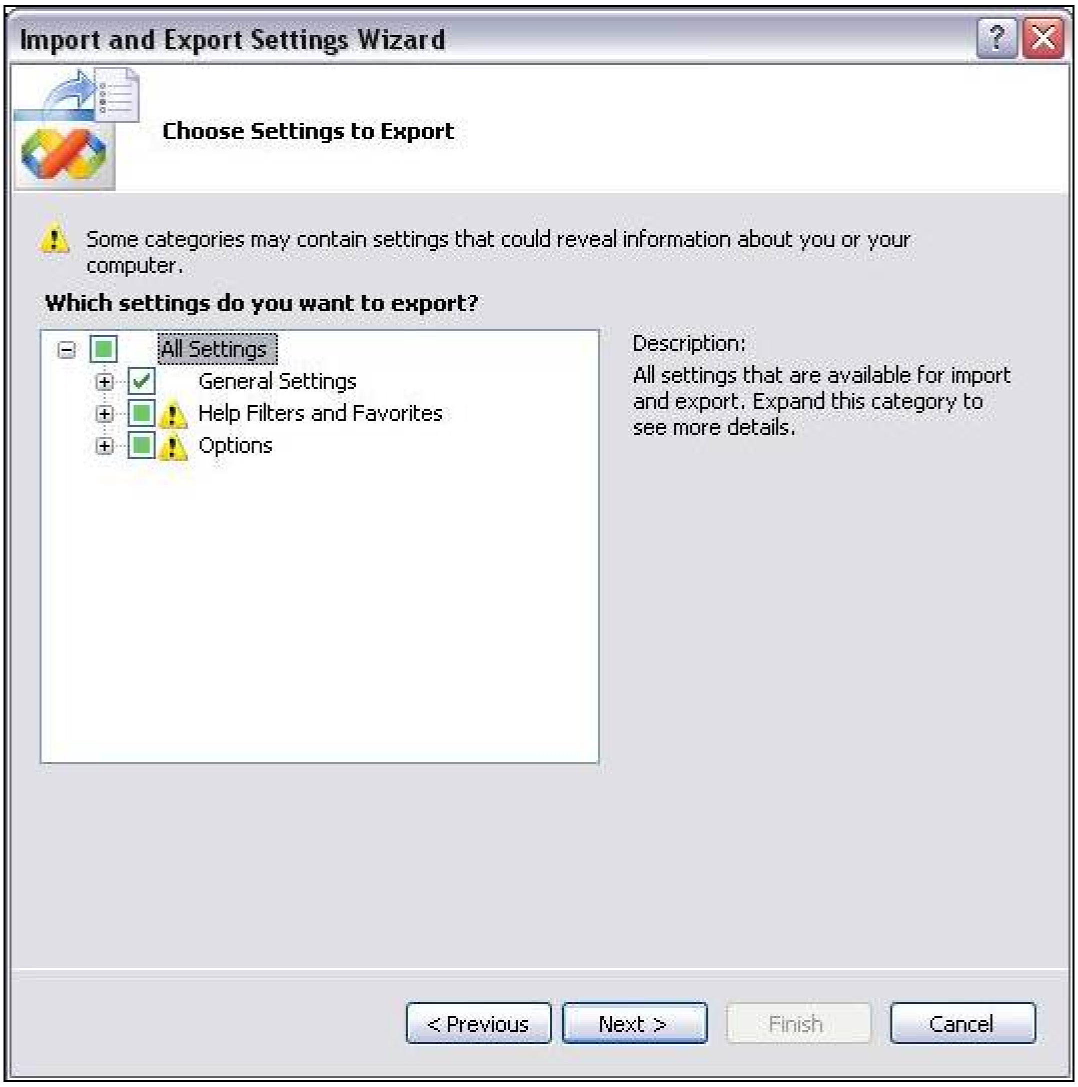
VS.NET 2005 allows you to export your IDE settings to an XML file (the extension is actually ".vssettings"). To import it into another instance of VS.NET on another computer:

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 38).

Check the options you want to be part of your profile.

Export them to the .vssettings file.


Import and Export Settings Wizard

Choose Settings to Export

Some categories may contain settings that could reveal information about you or your computer.

Which settings do you want to export?

- All Settings
  - ✓ General Settings
  - Help Filters and Favorites
  - Options

Description:
All settings that are available for import and export. Expand this category to see more details.

Figure 38. Export VS Settings

Import the .vssettings file to another VS.NET IDE.

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).

To create a master .vssettings file for coordination between co-workers, e-mail it to all your team members so that they can import it individually. You can also create the single .vssettings file and place it on a well-known network share on the intranet. To obtain these settings have the members of your team:

Select 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. One advantage of this feature is 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.

3.17 - Closing All Other Windows

It's very common to have a lot of files open at the same time when developing your program. After working for a while, you might have several dozen files open and want to close all of them except the one on which you are currently working.

To close all the open files:

Right-click one of the file tabs.

Choose Close All But This from the pop-up menu.

This option does exactly what it says (see Figure 39).

Other menu options new to VS.NET 2005:

1. Open Containing Folder
   -starts up Windows Explorer and opens the folder in which your file is located.

2. Copy Full Path
   -copies the full file path of the selected file into the Clipboard.

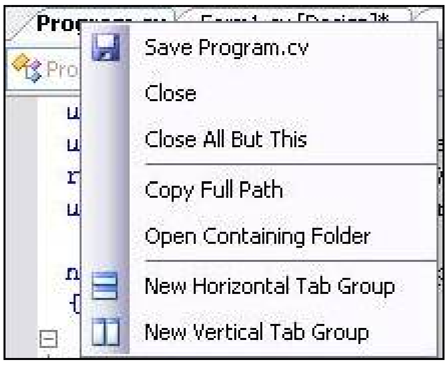


Figure 39. File tabs options

3.18 - Showing Shortcuts for All Buttons

Using and memorizing shortcuts whenever available, gives you a strong advantage when developing. It naturally increases your speed and therefore your efficiency. Keyboard shortcuts prove to be faster than manipulating the mouse. Many VS.NET menu and submenu items have these shortcuts which are seen every time you click the menu item.

This reminder is also available for the toolbar buttons. To see this reminder:

Select Tools > Customize

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.

Introduction

Visual Studio .NET comes complete with many features and functions that dramatically increase our efficiency as developers.   As a powerful code editor, compiler, and debugger, it contains features to stress-test, analyze, and optimize your code, and allows easy integration with code documentation, reporting, or smart-device programming, such as the Pocket PC.

Because of the sheer number of features that Visual Studio .NET contains, it is a challenge for .NET developers to become familiar with all of its features, shortcuts, and functionalities. The beginner developer will find a virtual treasure trove of features with which to start, while advanced Visual Studio .NET users will appreciate the many new features and improvements the new Visual Studio .NET 2005 brings.

Most of these features and functionalities are documented, and are accessible through the VS.NET main menu or context menus. But, because of the vast number of features with which VS.NET is equipped, developers don't always know them or use them. This guide should help familiarize developers with the tips and tricks that are at their disposal in this powerful tool and their specific application to the Visual APL language.

Chapter 1: Editing Code

While you create code, there are many techniques and shortcuts that allow you to write and navigate through your code quickly and easily. This chapter introduces many of the tips and tricks you will need for such tasks as code navigation, performing complex find-and-replace searches, and generating code.

1.1 - Commenting Code Blocks (Ctrl-K, Ctrl-C)

One-line comments are extremely useful in explaining seldom used code and assisting in navigation and definition of development projects. To inserted a comment for a code block or segment:

Press the “//” token for Visual APL.

Additionally, Visual APL allows you to comment entire paragraphs and segments. To place a comment in a paragraph or segment:

Select “/#” (and corresponding “#/”) tag around the comment.

To quickly comment entire paragraphs:

Select the text.

Click the Comment button (see Figure 2) or

Press Ctrl-K, Ctrl-C.

This comments an entire selection.

To uncomment any selection:

Click the Uncomment button or

Press Ctrl-K, Ctrl-U.

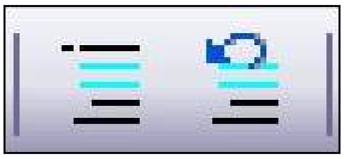


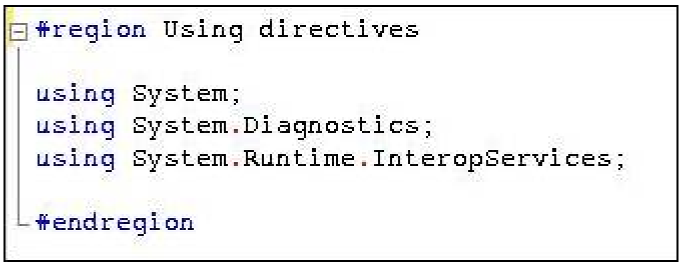
Figure 5 - Comment and Uncomment buttons

1.2 -Creating Regions

The more code you generate, the more difficult it can become to navigate. In addition to selecting classes and their methods from the drop-down lists above the main editor, you can also group your code into logical regions. Regions are extremely helpful for dividing code in logical ways and even commenting it. Regions allow you to collapse code to a single line defining the region and still easily see what is inside it once it is collapsed. They can even be nested. Automatically generated code in VS.NET usually uses this feature, so you may already be familiar with it.

To specify a region:

> Insert a #region keyword and a description at the beginning of your segment and a corresponding #endregion keyword at the end of your segment (see Figure 3).



```
#region Using directives

using System;
using System.Diagnostics;
using System.Runtime.InteropServices;

#endregion
```

Figure 3 - Creating regions

The Outlining menu displays various collapse and expand options. To expand and collapse the current region you are in:

> Press Ctrl-M, Ctrl-M.

To expand or collapse all regions at once:

> Right-click the gray bar to the left of the main editor window.

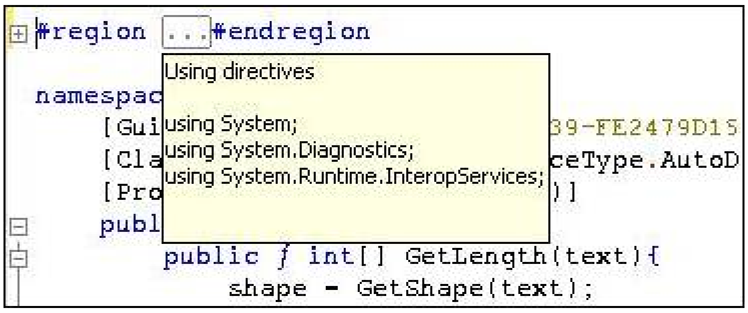
To collapse an individual region:

> Click the plus sign next to the #region keyword.

This collapses the code into a single line that shows the region description.

To display the inside of a collapsed region:

> Move the mouse over the gray description area (see Figure 4).



```
#region ...#endregion

namespac
    [Gui                                      39-FE2479D15
    [Cla                                      ceType.AutoD
    [Pro                                      )]
    publ
        public ƒ int[] GetLength(text){
            shape = GetShape(text);
```

```
Using directives

using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
```

Figure 4 - Mouse over a region to see its content

You can even drag and drop collapsed regions inside your code. When you paste a collapsed region into a different location, the pasted text is automatically expanded.

1.3 -Hiding Selection by Using Temporary Regions (Ctrl-M,Ctrl-H)

Regions are created automatically for methods, comments, and sections encompassed by the #region compiler directive. In Visual APL and in regular text files, you have the option to create temporary regions around any section without the need for "#region". This is useful when you want to create a region that will not be preserved once your project is closed. In this case, you can temporarily define a region using the following method:

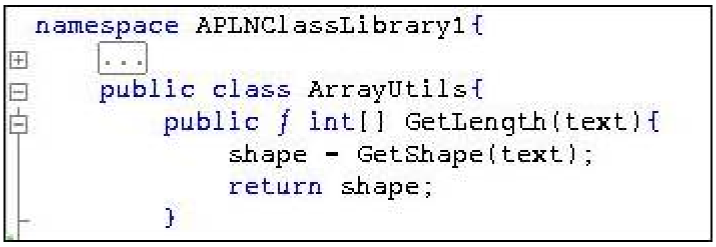
Highlight the section you want to hide and press Ctrl-M, Ctrl-H.

This hides the current selection in a temporary collapsed region (see Figure 5).

To expand a temporary collapsed region:

Press Ctrl-M, Ctrl-M.

Temporary regions are lost after you close a project.



```
namespace APLNClassLibrary1{
    ...
    public class ArrayUtils{
        public ƒ int[] GetLength(text){
            shape ← GetShape(text);
            return shape;
        }
```

Figure 5 - Hiding portions of any text file

For creating regions which are preserved past the closing of your project, see "Creating Regions"

1.4 -Selecting a Single Word (Ctrl+W)

To select a single word when editing code:

> Double-click anywhere in the word or just press Ctrl-W.

Double-clicking in a word is a common method used by many word processing and publishing programs to quickly select a word.

1.5 -Placing Code into the Toolbox (Ctrl-Alt-X)

When creating a project, you may want to use certain pieces of code or text again and again. You may have a standard copyright header that you place at the top of each file or a certain line of code to perform a common task. To simplify this repetitive task, you can place it into your Toolbox. The Toolbox is the window that lists all windows or web controls. To place your item into the Toolbox, use the following method:

1. Pull up the Toolbox:

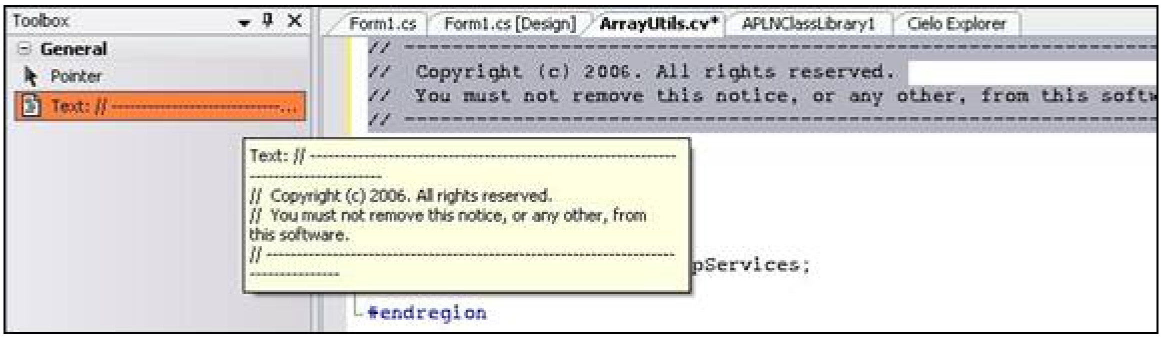
   Press Ctrl-Alt-X.

2. Move the frequently used text or code into the Toolbox:

   Highlight your item and drag the selected text onto the General tab in your Toolbox (see Figure 10).

3. Rename the produced text item in your Toolbox

   Right-click it and choose Rename Item from the pop-up menu.



```
// ------------------------------------------------------------------------
//  Copyright (c) 2006. All rights reserved.
//  You must not remove this notice, or any other, from this softw
// ------------------------------------------------------------------------
```

```
Text: // -----------------------------------------------------------------
-------------------------
// Copyright (c) 2006. All rights reserved.
// You must not remove this notice, or any other, from
this software.
// ----------------------------------------------------------------------
---------------
```

```
pServices;

#endregion
```

Figure 10. Adding text to the Toolbox

To insert an item into your text or code from the Toolbox:

Select the item in your Toolbox, then either:

Drag it into your code window or

Place your cursor in your text where you want to insert the item,
Double-click the entry

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

1.6 - Using the Clipboard Ring (Ctrl-Shift-V)

The Clipboard Ring works like a historical file of the last used text selections that you placed on the Clipboard. Because it preserves many levels of selections, it is useful when you accidentally overwrite the current Clipboard content or when you find yourself needing several different items concurrently.

To use the Clipboard Ring you can either:

Double-click one of the remembered Clipboard items to paste it at the cursor's current location or

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.

To progress one item at a time through the Clipboard Ring:

Select Edit > Cycle Clipboard Ring (Ctrl-Shift-V)

Doing this 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. This method makes it easy to paste specific content in the code editor as it becomes visible during the cycling. Continue cycling through the Clipboard's contents until you find your desired item.

1.7 - Transposing a Single Character or Word (Ctrl-T or Ctrl-Shift-T)

To switch the position of the current characters or words on either side of the cursor you need to use Transpose. This procedure switches the characters or words, then moves the cursor to the right. Transpose is useful if you mistype a word or write a sentence or code segment with words in the incorrect order.

To transpose a single character:

Press 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 one character at a time.

To transpose a single word:

Press Ctrl-Shift-T.

Note: This does more than just swap two adjacent words. VS.NET knows 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.

1.8 - Cutting, Copying, Deleting, and Transposing a Single Line

If you need to cut, copy, delete or transpose an entire line, you can do this easily with one keyboard sequence.

To **copy** the complete, current line to the Clipboard:

> Press Ctrl-C (or click the Copy icon) without any text selected.

To **cut** an entire line:

> Press Ctrl-X (or click the Cut icon) without any text selected.

This will cut the entire current line and place it in the Clipboard.

To **delete** a single line:

> Press Ctrl-L without any text selected.

To **transpose**, or **swap** the current line with the one below it:

> Press Alt-Shift-T without any text selected.

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.

1.9 - Formatting Entire Blocks (Ctrl-K, Ctrl-F or Ctrl-K,Ctrl-D)

To apply formatting to an entire selection, there are several useful functions you can use. Uppercasing, lowercasing, or deleting horizontal white spaces are just a few examples.

To access these features:

Select Edit > Advanced

One of the most useful features here is the Format Selection function.

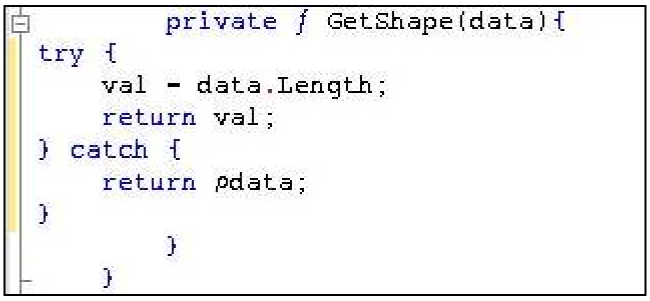
To access the Format Selection Function:

Press Ctrl-K, Ctrl-F,

This feature 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 Visual APL) but Format Selection forces this automatic format (see Figures 11 and 12).

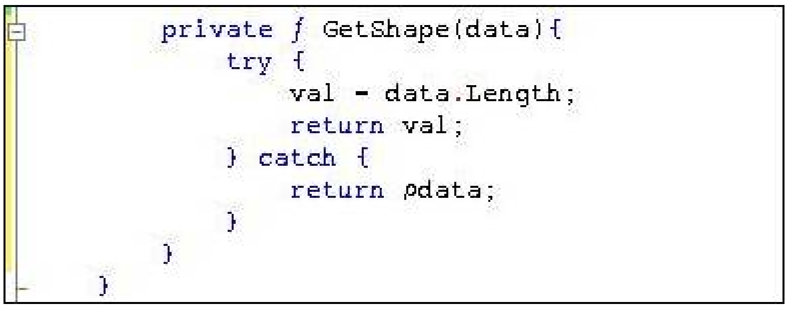
You can also format the entire document. To do this:

Press Ctrl-K, Ctrl-D.



```
        private ƒ GetShape(data){
try {
    val ← data.Length;
    return val;
} catch {
    return ⍴data;
}
        }
    }
```

Figure 11. Before formatting block



```
        private ƒ GetShape(data){
            try {
                val ← data.Length;
                return val;
            } catch {
                return ⍴data;
            }
        }
    }
```

Figure 12. After formatting block

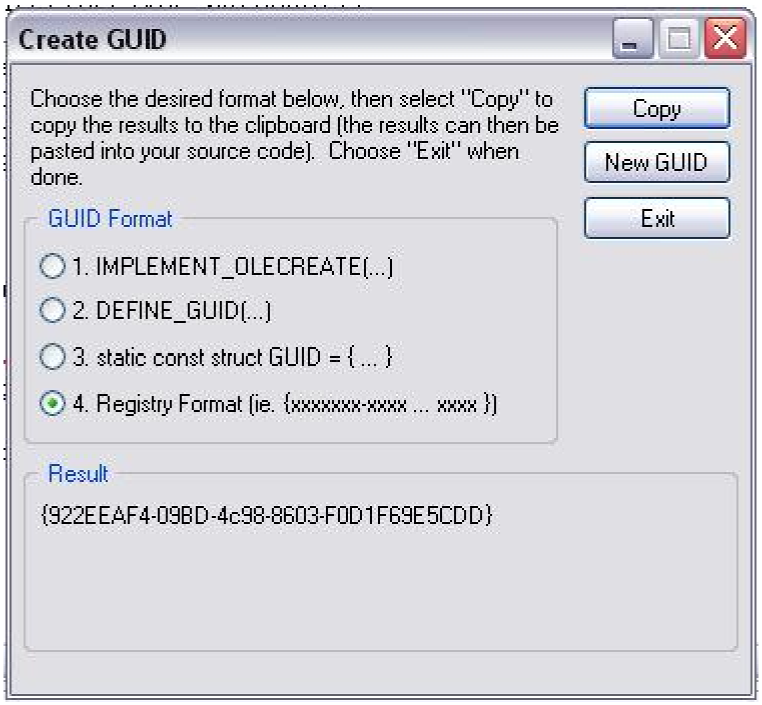
1.11 - Creating GUIDs

As you develop new classes and components, you often need to create Global Unique Identifiers (GUIDs). These are 128-bit values often represented by 32 hexadecimals. In the past, component developers used GUIDs to assign their components with unique names to reduce the likelihood of two components sharing the same GUID. Developers now use GUIDs for anything that requires a unique identifier. GUIDs can be created manually by randomly selecting 32 hexadecimals, but this is somewhat tedious. VS.NET comes with a utility that creates GUIDs for you whenever you need one.

To create a GUID, open the Create GUID dialog box:

Select Tools > Create GUID (see Figure 13).

Here you can generate identifiers in various formats, including common code items often used in COM development.


Create GUID

Choose the desired format below, then select "Copy" to copy the results to the clipboard (the results can then be pasted into your source code). Choose "Exit" when done.

GUID Format

1. IMPLEMENT_OLECREATE(...)
2. DEFINE_GUID(...)
3. static const struct GUID = { ... }
[✓ 4. Registry Format (ie. {xxxxxxx-xxxx ... xxxx })]

Result

{922EEAF4-09BD-4c98-8603-F0D1F69E5CDD}

Figure 13. Create GUID tool

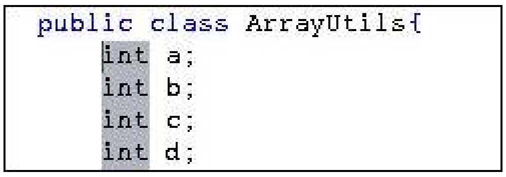
1.12 - Creating Rectangular Selections

To make a rectangular selection of text or code, there is a technique which allows you to do this without including the intervening lines (see Figure 14).

To select a rectangular area:

> Press the Alt key while dragging the mouse to select the area.

Manipulating the selection by copying, cutting, or pasting rectangular blocks can be done very easily this way.



```
public class ArrayUtils{
    int a;
    int b;
    int c;
    int d;
```

Figure 14. Rectangular selection

1.13 - Switching Between Views (F7)

For Windows forms, you can easily switch between both views. To toggle between designer and code views:

Press F7 (designer and code)

1.14 - Going to a Line Number (Ctrl-G)

For quick and easy navigation inside your code or text file, you can jump to a particular line.

To go to a specific line number you need to access the go to dialog box. To do this, either:

press Ctrl-G or

double-click the line number status bar at the bottom.

A small dialog box will appears. To jump to a line number:

enter a line number

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 number which exceeds the number of lines in the file will place you at the end of the file. A number that is too low will jump you to the beginning of the file.

1.15 - Searching for a Word

There are several methods for searching for a word inside a file. It is helpful to know all the methods for ease in moving around your file. The following are common methods for finding a word.

1. Access the Find dialog box

   Select Edit > Find

   Enter a term in the Find dialog box.

2. Access the Combo box in the main toolbar next to the configuration drop-down list. To open the combo box and invoke the search function:

   Press Ctrl-D

   Enter or paste a word into this list and press Enter

   Repeat pressing Enter in that drop-down list to find the next match.

3. Select the entire word, or place the cursor somewhere inside the word:

   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 specified in the Find dialog box. Set the options you desire in the Find dialog box first to search correctly (for example, enabling Search Hidden Text to include all collapsed regions in the search area).

1.16 - Performing an Incremental Search (Ctrl-I)

An incremental search 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.

To initiate an incremental search:

Press Ctrl-I

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.

To return to normal editing mode:

Press Escape

In the Cielo Explorer you have to single click with the mouse.

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.

1.17 - Searching or Replacing with Regular Expressions or Wildcards

Regular expressions can look extremely intimidating, but they are extremely powerful tools to find complicated search keys and patterns. Regular expressions is a built in feature that allows you to describe a searchable pattern in terms of wildcards, characters, and groups.

This feature in VS.NET is often overlooked by many developers. To access this feature, bring up the Search or the Replace dialog box:

Press either Ctrl-F or Ctrl-H, respectively

Note: Besides the regular options to refine your search, the last check box allows you to define your search based on regular expressions or wildcards.

You can use either of two modes. To use Regular Expression mode, specify the expression using a similar notation you are accustomed to with the System.Text.RegularExpressions namespace.

To see a list of possible constructs that you can insert into your regular expression:

Click the arrow button next to the Find What field (see Figure 15).

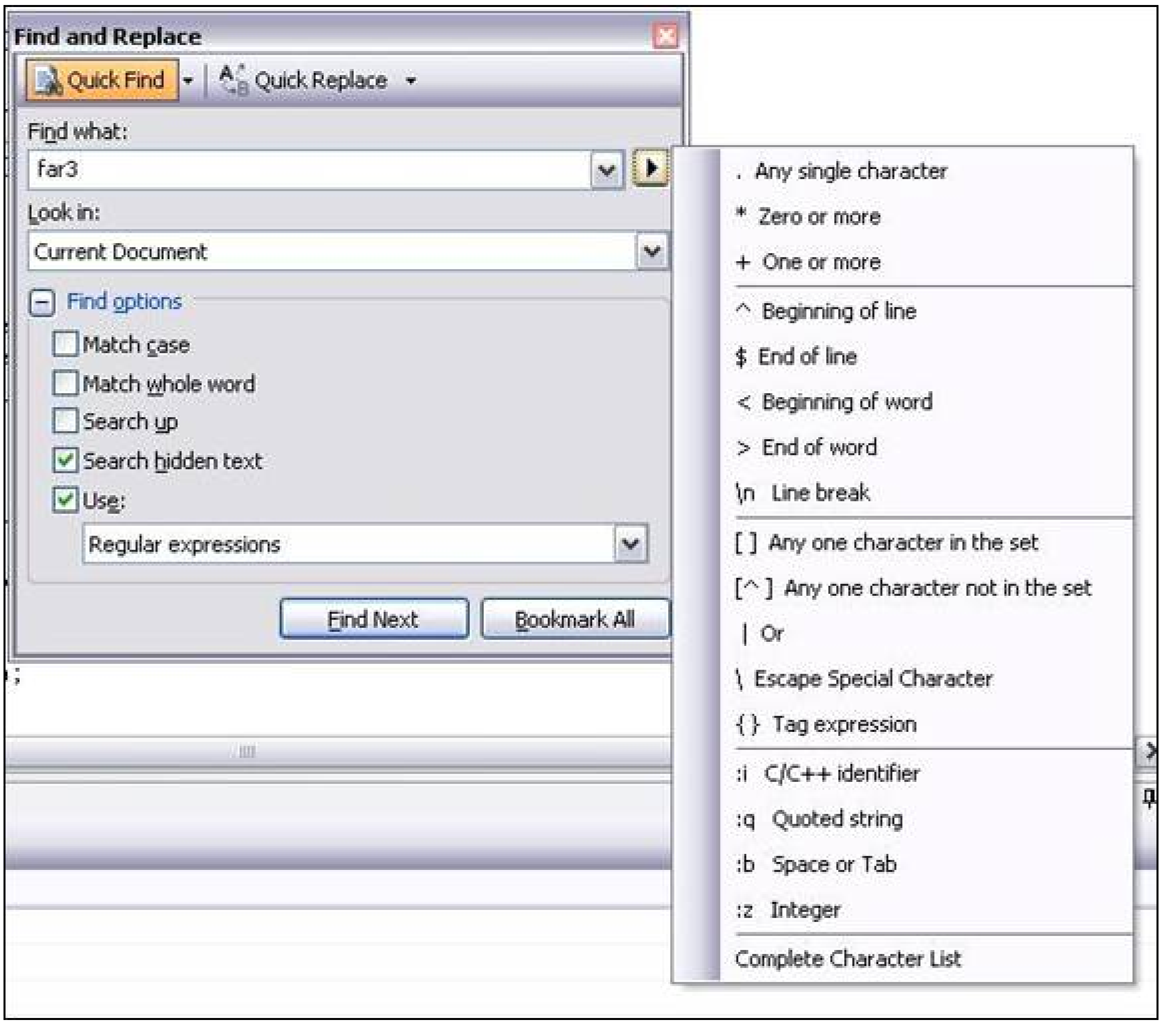


Figure 15. Use regular expressions.

To use 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 when developing programs based on regular expressions.

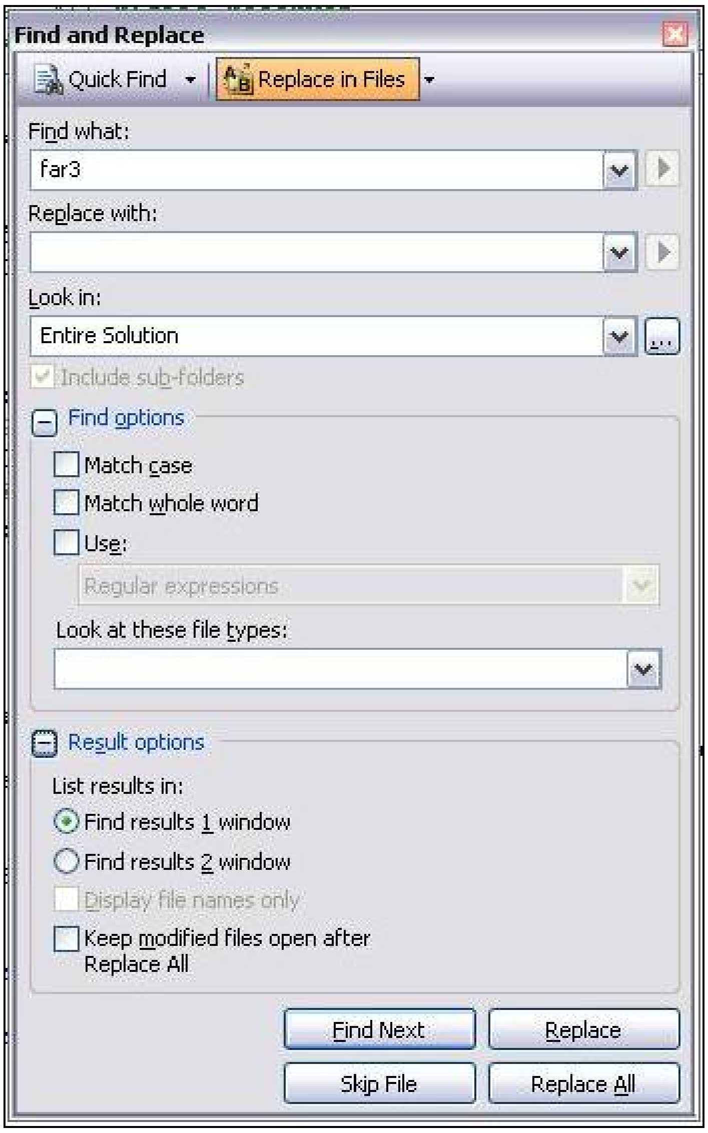
1.18 - Global Search or Replace (Ctrl-Shift-F or Ctrl-Shift-H)

The global search and replace feature in VS.NET spans entire projects and solutions. This is similar to normal search and replace dialog boxes, except that you can specify the scope of the search or replace action over multiple files.

To bring up the global search or global replace dialog box:

Press Ctrl-Shift-F or Ctrl-Shift-H, respectively

This feature allows you to perform a global search and replace in just the current document, the current project, the entire solution, or any open documents (see Figure 16). You can also filter which files you want to search based on wildcards.



Find and Replace

Quick Find | Replace in Files

Find what: far3

Replace with:

Look in: Entire Solution

☑ Include sub-folders

Find options

☐ Match case

☐ Match whole word

☐ Use: Regular expressions

Look at these file types:

Result options

List results in:

● Find results 1 window

○ Find results 2 window

☐ Display file names only

☐ Keep modified files open after Replace All

Figure 16. Global search and replace.

Once the search or replace action is started, VS.NET searches all specified documents and modifies them if required. The global replace, will also prompt you 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, global replace will automatically save the modified files and make this a permanent action.

To immediately stop a global search or replace anytime:

Press Ctrl-Break.

Once a search or replace action is completed, a list of occurrences that have been found will be displayed in the “Find Result” dialog box.

To iterate over the Find Result list:

Press F8 or navigate to an occurrence by double-clicking it.

If that occurrence is currently located in a collapsed region, you can expand it. To automatically expand the region:

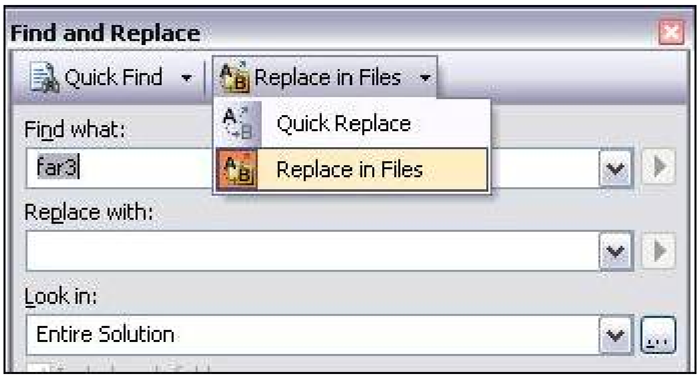
Double-click the same find result in the list again.

On initiating a new find or replace action, VS.NET clears this window to fill the list with the new results. If you want to keep the results of the previous search and output the result in a second window:

Check the Display in Find 2 option in the search dialog box

You can then tab between both result sets.

All find/replace functionalities are included in a single dialog box (see Figure 17), you can also access the global find/replace functionalities using the drop-down list at the top. All shortcuts remain the same.


Find and Replace

Find what: far3

Replace with:

Look in: Entire Solution

Figure 17. Global replace in files

1.19 - Using Bookmarks

Bookmarks enable you to return quickly to a given page or section of your code or file. When you determine critical sections of your programming that you want to return to frequently, instead of scrolling to these places, bookmark those lines.

To place a bookmark, first, make the bookmark toolbar visible:

Right-click any existing toolbar and select Text Editor from the pop-up menu.

Click on the blue flag icon in Text Editor toolbar.

Another method which can be used to place a bookmark:

Press Ctrl-K, Ctrl-K.

This second method not only makes a bookmark visible on the left side of the code, but you can now jump quickly among other bookmarks. To jump to the other bookmarks you can either:

Click the appropriate flag buttons on the toolbar (see Figure 18) or

Press Ctrl-K, Ctrl-P (for the previous bookmark) or Ctrl-K, Ctrl-N (for the next bookmark).

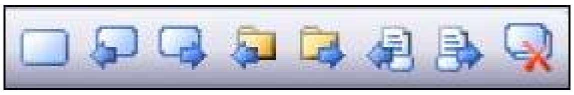


Figure 18. Bookmark toolbar

To clear all bookmarks:

Press the Clear Flag icon or

Press Ctrl-K, Ctrl-L.

The Find dialog box in VS.NET allows you to bookmark all matching occurrences as follows:

Click the Mark All button.

As part of VS.NET 2005 considerable support for bookmarks, you can also have the option of moving to the next or previous bookmark within the same file as follows:

Press the appropriate buttons on the bookmark toolbar (see Figure 28).

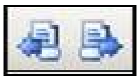


Figure 28 - Move to bookmarks in the same file in VS.NET 2005

You can also name your bookmarks by first opening a new Bookmarks window:

Press Ctrl-K, Ctrl-W or

Select View > Other Windows >Bookmark Window.

This displays all the bookmarks that you have created (see Figure 19).

To jump to a bookmark's location:

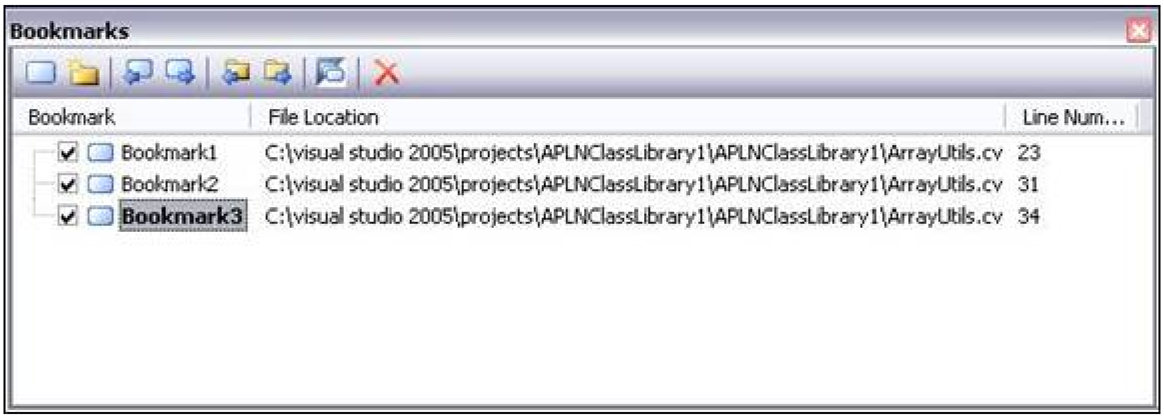
Double-click the bookmark.

To rename a bookmark:

Press F2 or

Right-click the bookmark and use the Rename context menu item.

You can categorize your bookmarks and organize them into folders. You can also, jump to the next or previous bookmark within the same folder. To perform any of these functions, simply select the appropriate icon on the toolbar.


Bookmarks

Bookmark	File Location	Line Num...
✓ Bookmark1	C:\visual studio 2005\projects\APLNClassLibrary1\APLNClassLibrary1\ArrayUtils.cv	23
✓ Bookmark2	C:\visual studio 2005\projects\APLNClassLibrary1\APLNClassLibrary1\ArrayUtils.cv	31
✓ Bookmark3	C:\visual studio 2005\projects\APLNClassLibrary1\APLNClassLibrary1\ArrayUtils.cv	34

Figure 19. Manage your bookmarks in the Bookmarks Window.

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

1.20 - Using Browser-Like Navigation (Ctrl -, Ctrl Shift -)

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 located to the right of the Undo and Redo buttons (see top left of Figure 20). You can also access them in the View menu.

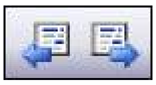


Figure 20. Navigate buttons

Similar to a web browser, VS.NET keeps a history of your recently accessed locations. 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 easily return back to the last edit location as follows:

Click the Navigate-Backward button.

These Navigate buttons have pre-assigned shortcuts. To Navigate back:

Press Ctrl-Hyphen

To Navigate forward:

Press Ctrl-Shift-Hyphen.

1.21 - Inserting External Text File

A common method of inserting code fragments involves opening a file in Notepad and copying the code from there. To bypass this step of opening and closing Notepad you can:

Select Edit > Insert File as Text from within the code editor.

Chapter 2: Exploring the IDE

Visual Studio .NET is an easily customizable feature-rich Integrated Development Environment (IDE). It allows a developer quick access to commonly used commands and activities which enable you to control and modify your project and solutions. This chapter covers a range of topics such as: the Solution Explorer; window positioning; managing macros; modifying menu items and other tips and tricks useful for in navigating inside the IDE.

2.0 - Setting Project Dependencies

In a large solution with multiple projects and custom build events, it is often necessary to control the build order for your projects. VS.NET has the capability to figure out which project needs to be built first by analyzing the references of each one. The first project built is normally the one referenced first. This algorithm is based on your set project references for your projects.

VS.NET also allows you to compile a certain project before another one without having project references. This is accomplished from a pop-up menu that allows you to choose Project Dependencies. To designate the order in which projects will be built:

Right-click your project that needs to be built last and choose Project Dependencies from the pop-up menu.

Set manual dependencies on other projects by check-marking them.

This will ensure that the checked projects will be built before the current project (see Figure 21). A drop-down menu for the current project allows you to switch to another project's dependencies.

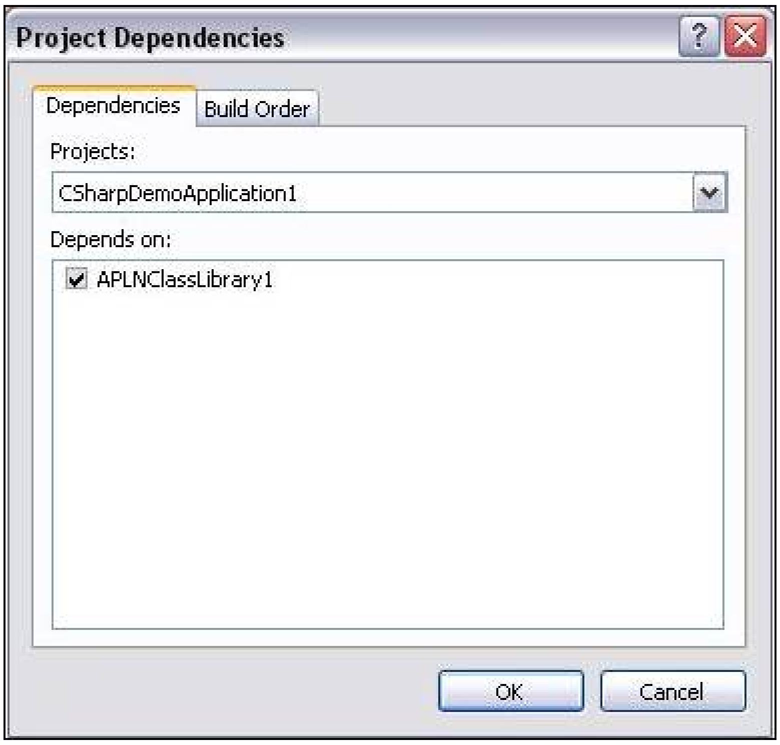


Figure 21. Setting project dependencies manually

VS.NET prevents you from creating circular references or modifying dependencies that resulted from adding project references. To verify the build order at any given time:

Click the read-only Build Order tab.

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

2.1 - Embedding Files As Resources

Embedding files as resources allows you to embed any given file directly into your produced assembly. For instance to display a company logo on your Windows application, you could produce a regular Windows assembly and link to an external image that you send along with your application. You can also embed the image right into the assembly you produce. This enables you to avoid shipping the external image and, more importantly, prevents the possibility of these two files becoming separated.

To embed a file as a resource, it must first be included in your solution. You can then select the file in the Solution Explorer and change the Build Action property in the Properties window. The build action tells the compiler what 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("mylogo.bmp");
```

After this retrieves the 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. This enables you to embed any file type: sound and movie files, PDF files, or even another assembly.

2.2 - Changing the Font Size of IDE Windows for Demos

It is a common practice when demonstrating VS.NET or your code, to increase the font size of the text editor so that everyone in the audience can easily see the demonstration. The font size can be easily increased by a couple of methods:

> Select Tools > Options > Environment > Font and Colors > Size.

This works great, except that the text in the Output windows, Solution Explorer, Class view, Macro Explorer, or in the file tab titles can still be very hard to read.

Control the size of the text in these elements as follows:

> In the same settings window, the first drop-down list reads Show Settings For. Change it to read Dialogs and Tools Windows.
>
> Set the font and the size here in this window.

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.

To increase the font of the Output window:

> Change Show Settings For to Text Output Tools Windows.

To reset any of these settings to their default installation values:

> Click the Use Defaults button.

Note: This button applies only to the currently selected item in the Show Settings For drop-down list, so repeat this step for every setting that you want reset back to the default settings.

2.3 - Dragging Files to Obtain a Full Path

A useful feature of VS.NET is the ability to 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.

2.4 - 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. To move any window in VS.NET:

Drag 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), an outline appears, allowing you to preview the result before dropping the window.

To dock and undock windows:

Double-click 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, the preview outlines are too confusing to make this an intuitive interface. If you have moved the windows positions and would like them reset, you can always reset all windows positions to their installation defaults:

Select Tools > Options > Environment > General > Reset Windows Layout.

With VS.NET 2005, you can also reset the windows positions. To do this:

Select Window > Reset Windows Layout.

One aspect of moving windows around is the ability to create a split screen. Use the following steps to split the editor into two vertical screens complete with their own set of file tabs (see Figure 23):

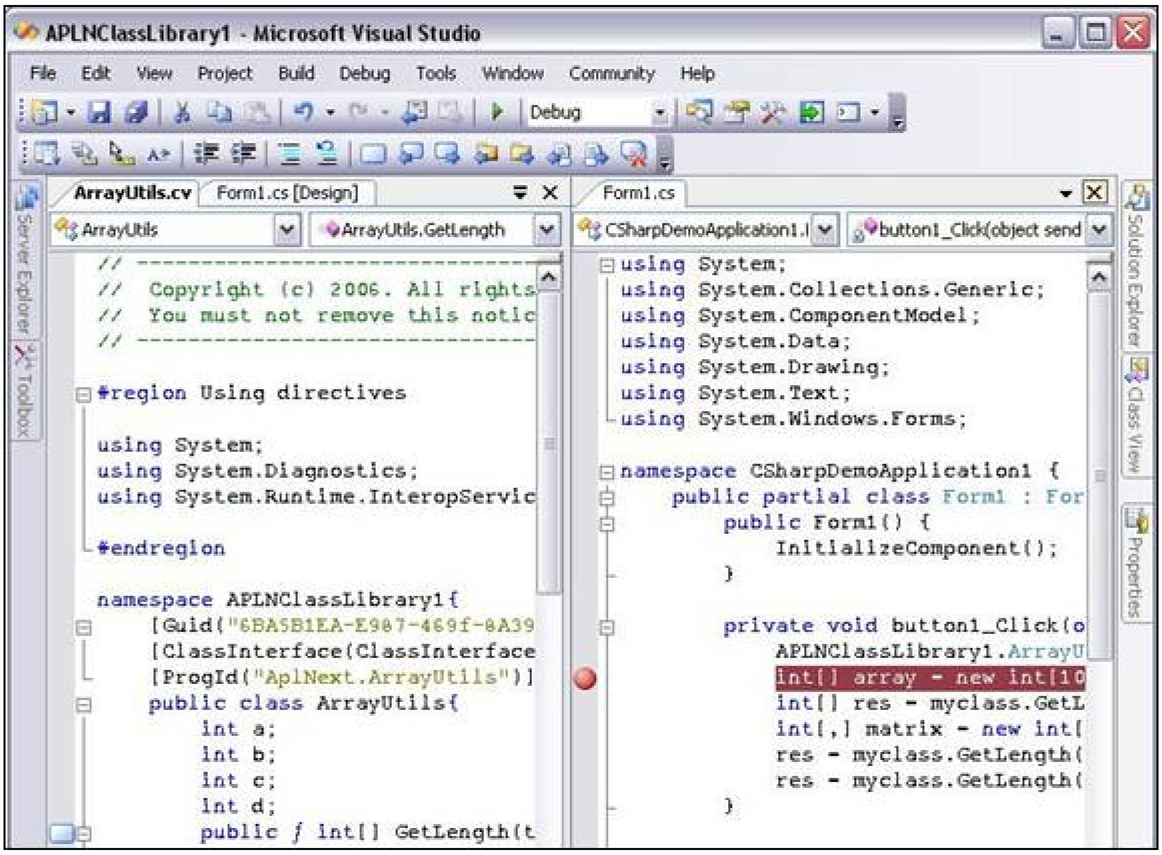
Drag the tab of any open file and move it to the right of your editor (to the left of where the Solution Explorer usually resides).

This docks your selected file to the right and splits the editor into two vertical screens.

To close vertical split mode either:

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.



```
//  ------------------------------------
//   Copyright (c) 2006. All rights
//   You must not remove this notic
//  ------------------------------------

#region Using directives

using System;
using System.Diagnostics;
using System.Runtime.InteropServic

#endregion

namespace APLNClassLibrary1{
    [Guid("6BA5B1EA-E987-469f-8A39
    [ClassInterface(ClassInterface
    [ProgId("AplNext.ArrayUtils")]
    public class ArrayUtils{
        int a;
        int b;
        int c;
        int d;
        public ƒ int[] GetLength(t
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace CSharpDemoApplication1 {
    public partial class Form1 : For
        public Form1() {
            InitializeComponent();
        }

        private void button1_Click(o
            APLNClassLibrary1.ArrayU
            int[] array = new int[10
            int[] res = myclass.GetL
            int[,] matrix = new int[
            res = myclass.GetLength(
            res = myclass.GetLength(
        }
```

Figure 22. Vertical split

To create a horizontal split screen:

Drag a file tab to the bottom of your editor.

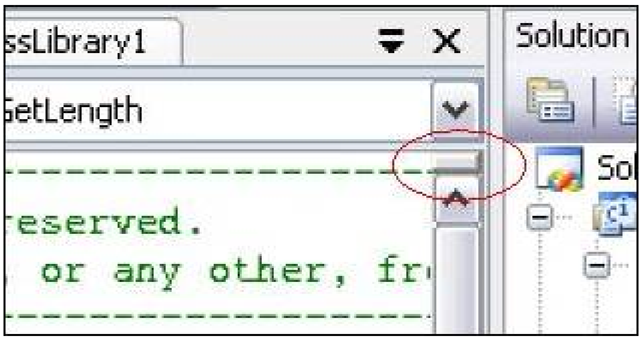
2.5 - Creating Split Screens in the Same File

The “Moving Any Window Around” trick described in “Moving Any Window Around” 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? To do this:

> select Window > Split

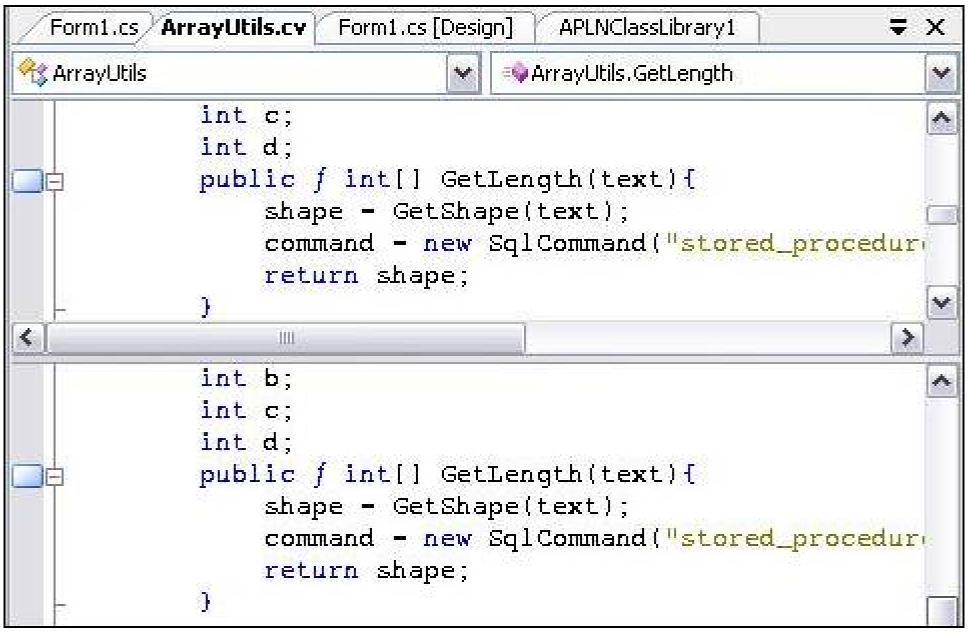
The horizontal divider can also be generated using a faster method:

> Move your cursor right above the vertical scrollbar of the main editor. There is a very thin, short, rectangular-shaped divider (see Figure 23).
>
> Place your mouse over that divider, 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 24).



```
-------------------
eserved.
, or any other, fr
-------------------
```

Figure 23. Horizontal split divider



```
        int c;
        int d;
        public ƒ int[] GetLength(text){
            shape = GetShape(text);
            command = new SqlCommand("stored_procedur
            return shape;
        }
```

```
        int b;
        int c;
        int d;
        public ƒ int[] GetLength(text){
            shape = GetShape(text);
            command = new SqlCommand("stored_procedur
            return shape;
        }
```

Figure 24. Split document.

To move the divider back to the top of your editor window:

> Select the divider bar and slide back into its original position.

2.6 - Customizing the VS.NET Menu and Toolbars

The VS.NET menu can be customized in a variety of ways. You can add and remove commands as well as reorder them. To customize the menu and toolbars:

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 and will show 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. To copy icons from other menu items:

Right-click a menu item with the desired icon.

Choose Copy Button Image from the pop-up menu

Right-click the menu item you want to modify.

Choose Paste Button Image from the pop-up menu.

To add other commands to a menu:

Drag a command from the Command tab directly into the VS.NET menu.

Note: 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 25).

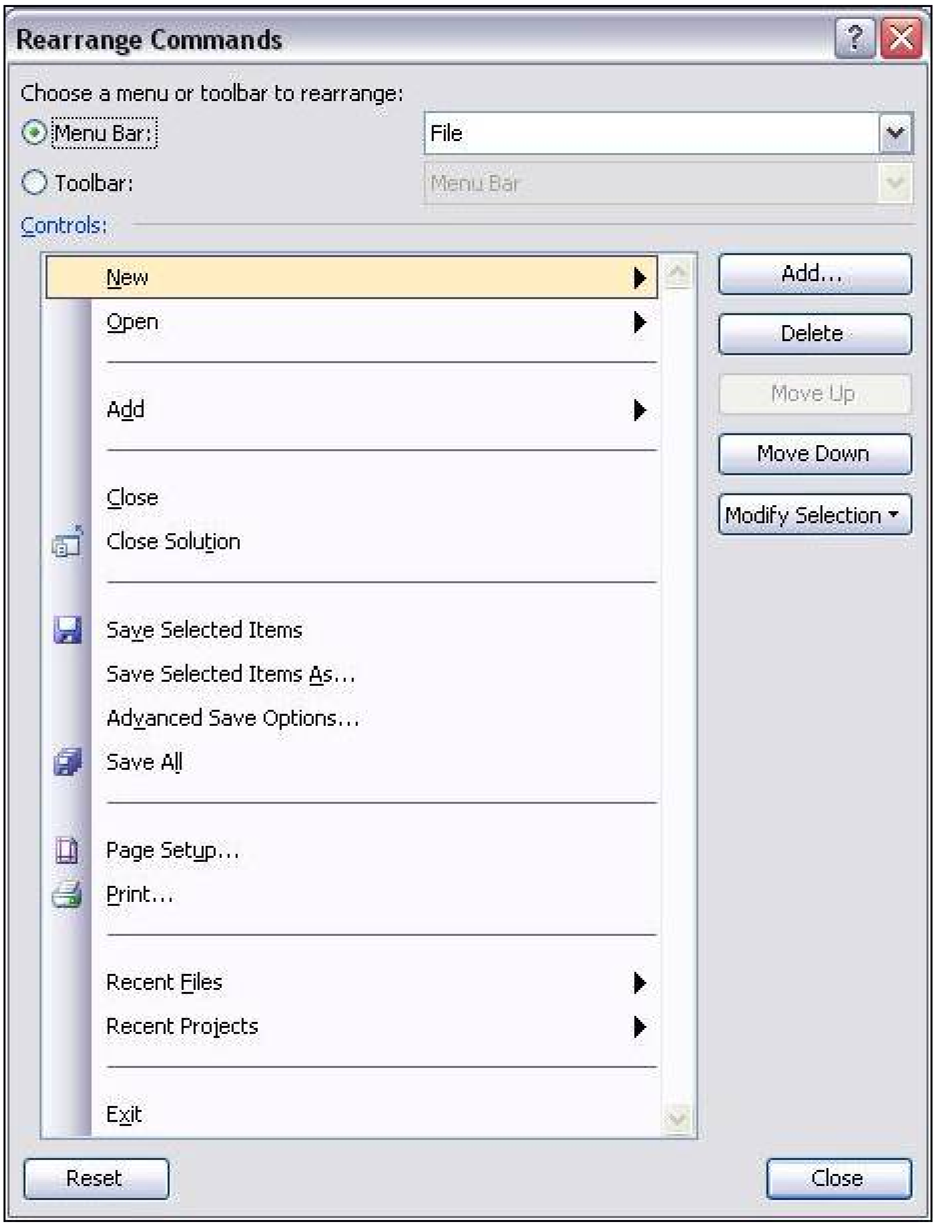


Figure 25. Customize menus using the rearrange commands

2.7 - Dragging Files from Windows Explorer into VS.NET

Visual Studio .NET completely supports file drag (and drop) actions. It allows you to 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.

To drag files from Windows Explorer into VS.NET if you don't have enough screen space:

Drag the file into the Windows taskbar at the bottom of your screen

Pause for a few seconds over the taskbar for VS.NET. The pause brings VS.NET into focus.

Drop the file into the appropriate location.

2.8 - Using Full-Screen Mode (Ctrl – Shift – Enter)

Full-screen mode allows you to hide virtually everything except the main editor, where the entire screen shows the main view. To enter full-screen mode:

Select View > Full Screen or

Press Ctrl-Shift-Enter.

The main menu is still visible at the top, and a floating button that closes full-screen mode is also available. To hide the Close Full Screen mode button—you need to 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.

2.9 - Copying the Fully Qualified Name of a Class

The Class view is a hierarchical view of all your classes and namespaces in your solution. To display this view:

Select View > Class View or press Ctrl-Shift-C.

To go to any class and its members and navigate to the member definitions:

Double-click on the desired item.

Another useful feature allows you to extract the full namespace of any class or member:

Highlight the class or the class member.

Press Ctrl-C.

This copies the complete namespace of the selected item to the Clipboard. This feature comes in handy when you have a complex or deep namespace structure.

To paste the namespace into the VS.NET code editor, there is no need to copy it to the Clipboard first. Use the following method:

Drag a class or member of a class from the Class view directly into your code

Watch VS.NET paste the complete namespace and member name there.

2.10 - Changing Properties of Several Controls

When designing your Windows forms, you can 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. To select a series of controls either:

Hold down Ctrl or Shift when selecting controls or

Draw 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 is useful 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.

Change it back to an empty string by pressing Delete.

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.

2.11 - Locking Controls

When laying out windows controls on Windows forms, you can easily move the controls around or create event handlers by simple dragging and double-clicking. However, this simplicity has its drawbacks as you can move things around accidentally very easily. This can cause problems if you have already finished designing your Windows forms. In order to prevent this from happening, you can lock your form.

To lock the position of your controls on your form:

While in the Designer view, right-click anywhere on your form

Choose Lock Controls from the pop-up menu (see Figure 26).

You still have the ability to add event handlers and modify a control's appearance, but you can no longer accidentally move or resize a control. To indicate that it is locked and unmovable, a thin, black outline appears around each selected control.

To return to the Designer view:

Right-click your form and choose Lock Controls from the pop-up menu again.

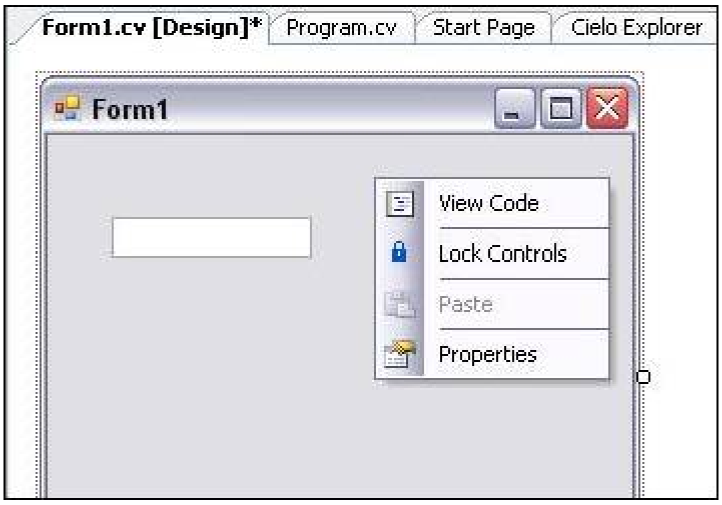


Figure 26. Lock controls in a form

2.12 - Toggling the Description in the Properties Window

The Properties window not only displays all properties of a selected control, but 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. To turn off the Description box panel:

Right-click the property name.

Choose Description from the pop-up menu.

To turn it back on use the same method.

2.13 - 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 selection. For instance, the FormBorderStyle property of a Windows form only accepts None, FixedSingle, Fixed3D, FixedDialog, Sizable, FixedToolWindow, and SizableToolWindow. To select the appropriate item:

Open the drop-down list.

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).

This trick can be extremely useful when switching Boolean values because a double-click changes the value quickly from True to False, or vice versa.

2.14 - Adding and Removing Event Handlers Through the IDE

Adding default handlers through the IDE is quite easy. In most cases, you only need to double-click a control which creates the necessary code for the default event handler.

Adding and removing non-default events handlers is still easy, but, in Visual APL, it 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 Visual APL is to use the Properties window:

Select the control

Click the Events button in the Properties window (the yellow 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.

In addition, the event handler fields are clickable (see Figure 27).

To create an event handler:

Double-click an empty field.

Choose which event you want to subscribe to.

To hook an event handler which is already written, to an event:

Use the drop-down button next to the selected field that automatically lists all matching event handlers.

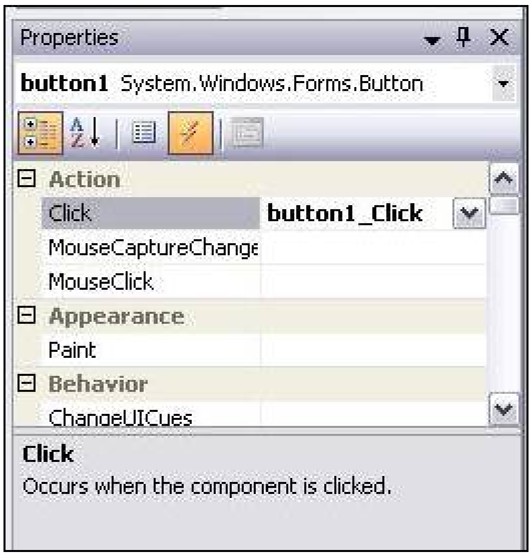


Figure 27. Setting events

To delete an event handler:

Delete the value in the event field.

This also removes the event handler subscription you have in the InitializeComponents() method.

2.15 - Selecting Control Through a Drop-Down List

When there are many controls on a Windows form, it can become a challenge to find a specific control, and 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.

To select a specific control:

Select the drop-down list that appears right above the Properties window.

Select the desired control.

Note: 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.

Chapter 3: Compiling, Debugging, and Deploying

Not only is VS.NET a great editor, it is also a powerful compiler, debugger, and profiler. It allows you to precisely control your compilation procedure and provides the features which are absolutely essential in to locating and fixing a bug: analyzing your code, attaching to running processes that you want to debug, and changing code and variables at runtime. This chapter covers topics that you need to know when it comes to compiling and debugging your programs.

3.0 - Setting the Default Namespace and Assembly Name

Following 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. It places it, by default, in the top-level namespace, which usually means the name of your assembly. To set the default namespace when you create new projects:

Select Project > Properties > Application.

Specify the default namespace in the Default Namespace field.

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.

3.1 - Generating Compiler Warnings Through the Obsolete Attribute

A commonly used way to display warnings in VS.NET at compile-time is to set an Obsolete Attribute to a method. Throughout the product development cycle, occasionally certain methods become obsolete. Sometimes the old method is not useful anymore. It may have become inefficient, or has been replaced by another method. If you can't modify those methods, will need to write another implementation of the method using a slightly different name or signature. To maintain 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()...
```

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 affect the compilation behavior in any way. You also must activate the Task List to see these warnings. Unlike warning compiler directives, the warning only appears if there is code that tries to invoke the obsolete method. These warnings will never appear if you don't refer to these methods anywhere in your code.

3.2 - 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.
These are the default settings. To specify another directory where you want to place the produced assemblies and external files:

Select Project > Properties > Build for Visual APL 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, if you want, you can 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.

3.3 - Setting the .NET Framework Version for Your Assembly

A great side-by-side installation feature of the .NET Framework is the ability to have multiple versions of the .NET Framework installed on a given computer, without any of them interfering. By default, all non-web applications use the .NET Framework with which they were compiled (if available), 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). What you need to do is:

Insert the appropriate Configuration/startup/supportedRuntime and Configuration/startup/requiredRuntime XML tags in the configuration file

Set its version attribute to the specific .NET Framework version.

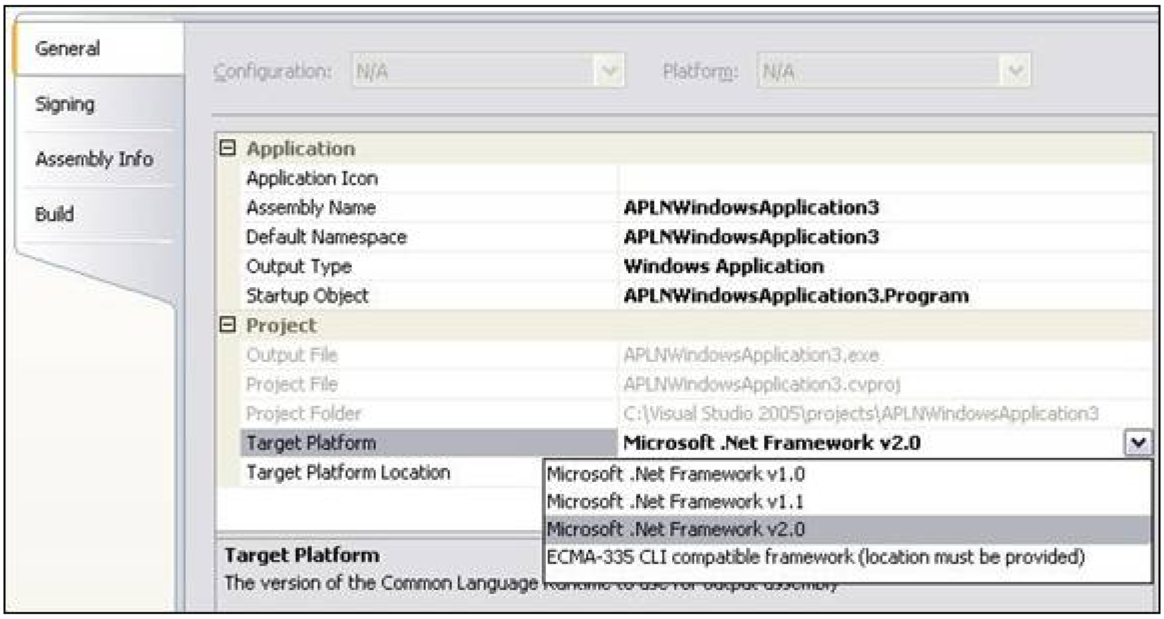
This enables you to force a Windows application to use an older version of the .NET Framework.

This configuration modification is easy in VS.NET. To set this for Visual APL:

Select Project > Properties > General > Target Platform.

Set the supported and required runtime versions for your assembly (see Figure 28).

To verify that your assembly is picking up the correct version, check the Version property in the .NET Framework class System.Environment.



Application
Application Icon
Assembly Name APLNWindowsApplication3
Default Namespace APLNWindowsApplication3
Output Type Windows Application
Startup Object APLNWindowsApplication3.Program
Project
Output File APLNWindowsApplication3.exe
Project File APLNWindowsApplication3.cvproj
Project Folder C:\Visual Studio 2005\projects\APLNWindowsApplication3
Target Platform Microsoft .Net Framework v2.0
Target Platform Location

Microsoft .Net Framework v1.0
Microsoft .Net Framework v1.1
Microsoft .Net Framework v2.0
ECMA-335 CLI compatible framework (location must be provided)

Target Platform
The version of the Common Language Runtime to use for output assembly

Figure 28. Choosing the target runtime.

Note: Supporting the 1.0 Framework, or even version 1.1 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 2.0–specific features.

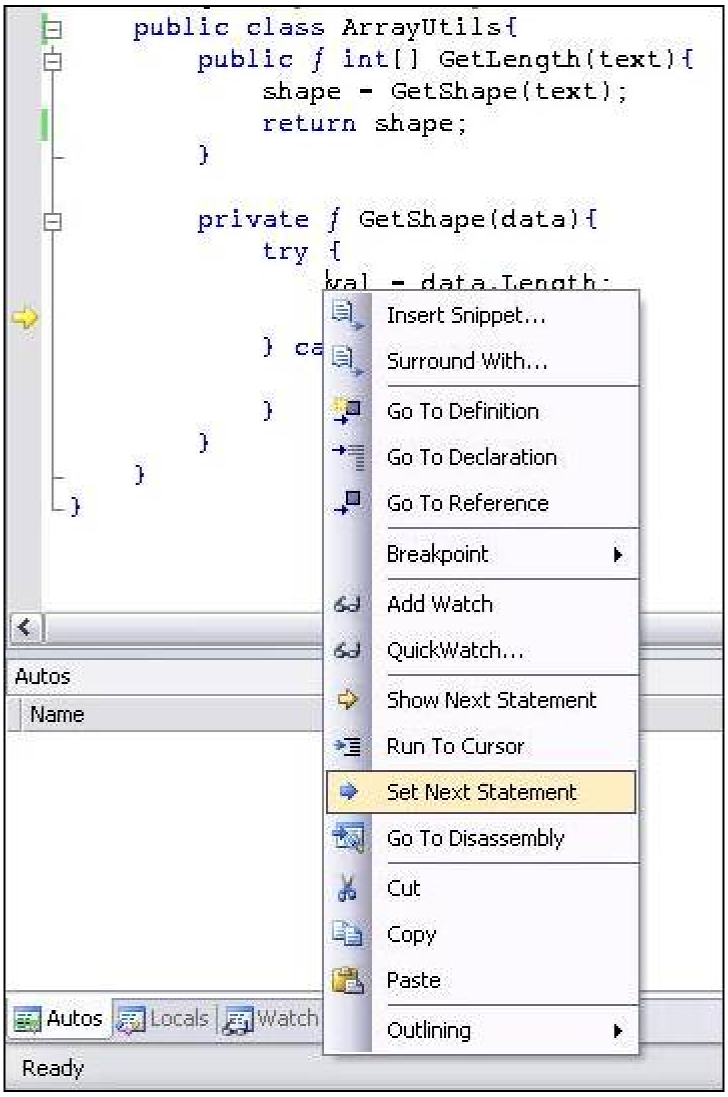
3.4 - Moving the Next Statement During Debugging

When stepping through your program one line at a time, you may need to jump a few lines back. To do this:

Right-click an arbitrary line

Choose Set Next Statement from the pop-up menu (see Figure 29).

This forces the debugger to jump to that line and continue debugging "normally" from there.



```
public class ArrayUtils{
    public ƒ int[] GetLength(text){
        shape = GetShape(text);
        return shape;
    }

    private ƒ GetShape(data){
        try {
            val = data.Length;

        } ca

        }
    }
}
}
```

Figure 29. Set Next Statement

To jump back, and also jump forward in and out of control statements:

Drag the yellow arrow to any line.

Note: You cannot jump out of the current stack frame, so you are limited to moving inside your current method.

In addition, moving the current execution line can bring your program into states that under normal execution could not occur. Still, it's an extremely useful feature to rerun certain code lines without restarting your debugging session.

3.5 - Changing Variable Values in the Watch Window

In addition to moving the next-statement pointer, you can change variable values at debug-time. In the process of debugging your application, you may have moved your variables of interest into the Watch window (probably 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 accomplished by entering the new value.

Note: You need to change the internal tick value of DateTime variables.

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. You can only change a reference variable to another reference variable of the same type (or its derived types).

Note: This can bring your program into states that under normal conditions would never be encountered.

3.6 - 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. In addition to 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. To view, edit, and modify stored procedures:

Right-click any stored procedure.

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 quite useful for its designed purpose of viewing, editing, and modifying stored procedures.

To execute stored procedures at design-time:

Right-click a stored procedure.

Choose Run Stored Procedure from the pop-up menu.

VS.NET inspects your stored procedure’s parameter list. If necessary, the Run Stored Procedure dialog box is displayed:

Enter each parameter’s value.

Execute your stored procedure and see the results.

3.7 - 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. While debugging your program, you step into methods and methods within methods. The stack trace keeps track of all these different levels.

To see the current stack trace:

Select Debug > Windows > Call Stack or

Press Ctrl-Alt-C,

Each method invocation is displayed on its own line, including the line-number and argument values. Each new method invocation is known as a stack frame.

The stack trace has been around in Visual Studio for a long time and is a widely known tool. The advantage of the stack trace window is that it allows you to identify how you get to the current execution point and also inspect the arguments that have been passed to the methods.

To make VS.NET immediately jump to the method invocation on a particular level of your program:

Double-click any line in the stack trace.

A relatively unknown aspect of the stack trace is that you can customize the Call Stack window. To do this:

Right-click the call stack.

Customize what appears there (see Figure 30) according to your requirements.

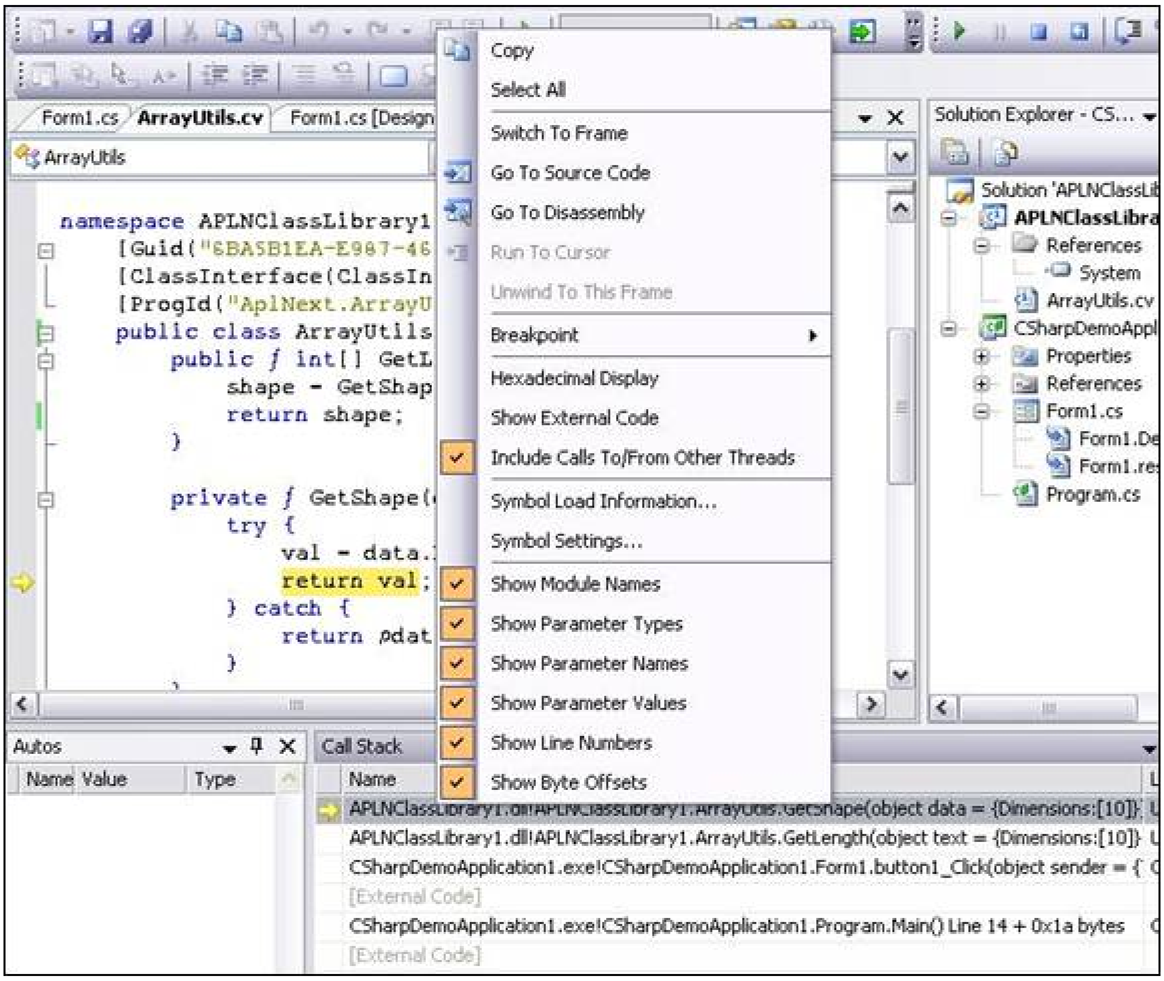
In addition, you can send the information regarding a single method invocation to a coworker:

Copy a stack frame to the Clipboard by pressing Ctrl-C.

To send your coworker the entire call stack:

Press Ctrl-A first, or

Before copying the selection to the Clipboard, Choose Select All from the context menu that appears after you right-click.



```
namespace APLNClassLibrary1
    [Guid("6BA5B1EA-E987-46
    [ClassInterface(ClassIn
    [ProgId("AplNext.ArrayU
    public class ArrayUtils
        public ƒ int[] GetL
            shape ← GetShap
            return shape;
        }

        private ƒ GetShape(
            try {
                val ← data.
                return val;
            } catch {
                return ⍴dat
            }
```

Copy
Select All
Switch To Frame
Go To Source Code
Go To Disassembly
Run To Cursor
Unwind To This Frame
Breakpoint ▸
Hexadecimal Display
Show External Code
✓ Include Calls To/From Other Threads
Symbol Load Information...
Symbol Settings...
✓ Show Module Names
✓ Show Parameter Types
✓ Show Parameter Names
✓ Show Parameter Values
✓ Show Line Numbers
✓ Show Byte Offsets

```
APLNClassLibrary1.dll!APLNClassLibrary1.ArrayUtils.GetShape(object data = {Dimensions:[10]}
APLNClassLibrary1.dll!APLNClassLibrary1.ArrayUtils.GetLength(object text = {Dimensions:[10]}
CSharpDemoApplication1.exe!CSharpDemoApplication1.Form1.button1_Click(object sender = {
[External Code]
CSharpDemoApplication1.exe!CSharpDemoApplication1.Program.Main() Line 14 + 0x1a bytes
[External Code]
```

Figure 30. Customize the CallStack

3.8 - Attaching VS.NET to an Already Running Process

To instruct VS.NET to debug your program, you first are telling it to build your project (if necessary) then start 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, assuming that the project was built with the debug release. To debug your program:

Press F5

There some cases, where you need, or want, to debug an already running process that has not been started with VS.NET you must:

Open the project for the program that is already running.

Select Debug > Attach to Process

A list of all active processes on your machine is displayed.

From the Processes dialog box, select the process you are interested in debugging and click Attach.

3.9 - Debugging Several Projects Inside the Solution

In a multi-project solution, VS.NET will start the project that you have marked as the "startup project." That project is indicated 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.

It is possible to debug those programs anyway, using the instructions in, "Attaching VS.NET to an Already Running Process."

To instruct VS.NET to start a project and attachs itself to a specific program:

Right-click your project

Select Debug > Start New Instance from the pop-up menu.

You can repeat these steps several times to start multiple instances of your program and still debug them all. This is useful in debugging multi-threaded client-server scenarios.

Tell VS.NET which projects you want to start on each new debug session (see Figure 31):

Right-click your solution

Choose Set Startup Projects from the pop-up menu.

By default, VS.NET uses the Single Startup project, where only one project is started.

To start more than one project:

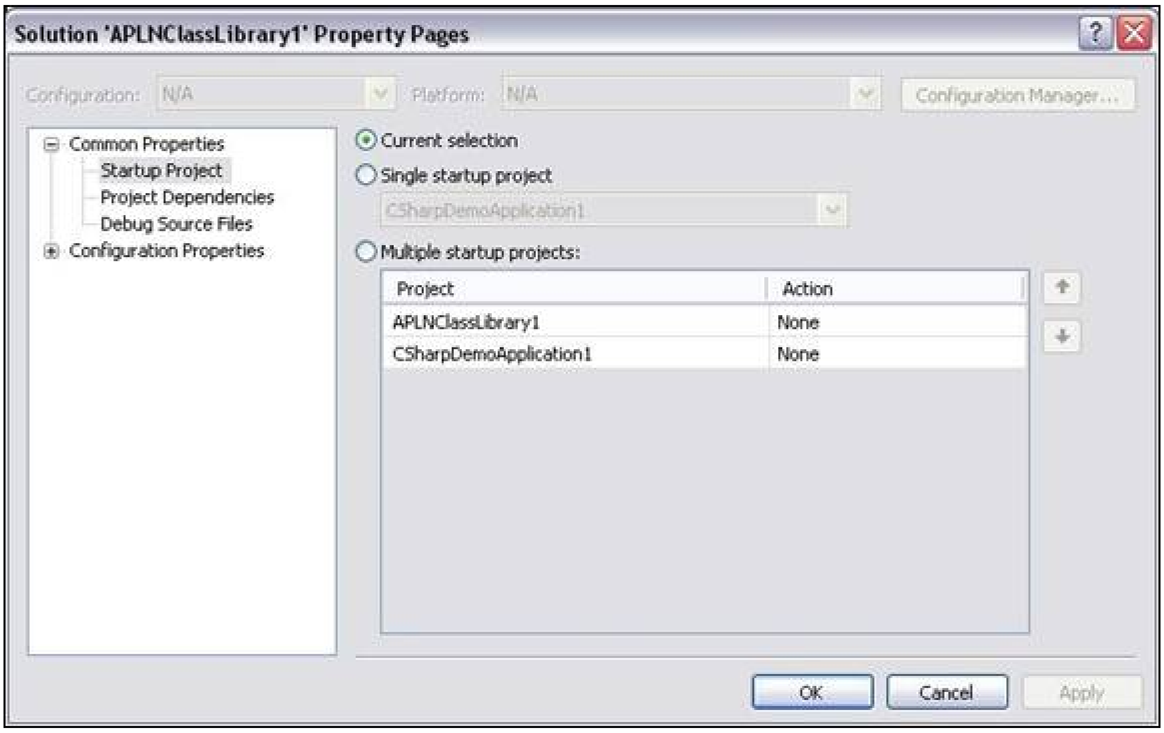
Switch to Multiple Startup Projects

Modify the Action value for each property: None, Start, or Start Without Debugging.

To control the order by which these multiple projects start:

Click 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.


Solution 'APLNClassLibrary1' Property Pages

Configuration: N/A Platform: N/A

Common Properties
- Startup Project
- Project Dependencies
- Debug Source Files

Configuration Properties

[✓ Current selection]
Single startup project: CSharpDemoApplication1
Multiple startup projects:

Project	Action
APLNClassLibrary1	None
CSharpDemoApplication1	None

Figure 31. Multiple startup projects

3.10 - 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.

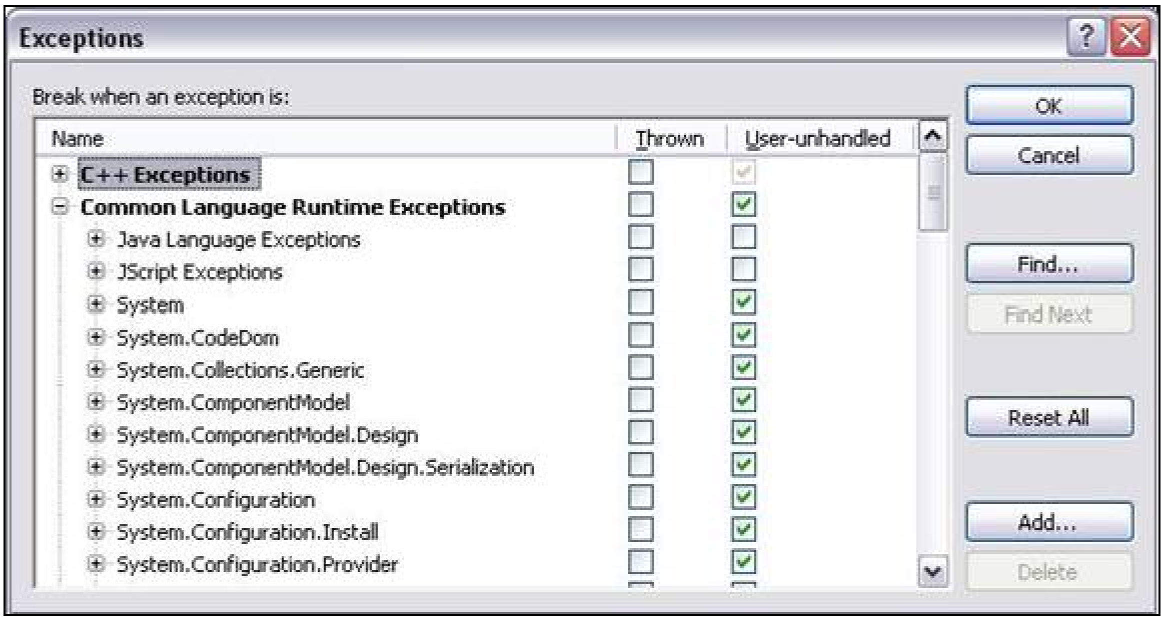
To specify the exceptions that developers are interested in defining, there is a setting in VS.NET. To utilize this setting:

Select Debug > Exceptions, or

Ctrl-Alt-E.

A tree view–style list of all possible exceptions that VS.NET can hook into (see Figure 32) will be displayed.

In addition to the many Common Language Runtime exceptions, you can hook into C++, Native Run-Time checks, and Win32 exceptions.



Break when an exception is:

Name	Thrown	User-unhandled
⊞ C++ Exceptions	☐	✓
⊟ Common Language Runtime Exceptions	☐	✓
⊞ Java Language Exceptions	☐	☐
⊞ JScript Exceptions	☐	☐
⊞ System	☐	✓
⊞ System.CodeDom	☐	✓
⊞ System.Collections.Generic	☐	✓
⊞ System.ComponentModel	☐	✓
⊞ System.ComponentModel.Design	☐	✓
⊞ System.ComponentModel.Design.Serialization	☐	✓
⊞ System.Configuration	☐	✓
⊞ System.Configuration.Install	☐	✓
⊞ System.Configuration.Provider	☐	✓

Figure 32. Break on specific exceptions

From this list you are able to:

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. In the predefined .NET exceptions, you can hook into your own .NET exceptions.

To specify the complete, fully qualified string that defines your .NET exception, for example, "MyCompany.MyProduct.MyBusinessException":

Click the Add button in the Exceptions dialog box.

3.11 - Breaking Only When Certain Conditions Apply (Ctrl – Alt – B)

A heavily used method to add or remove exceptions is 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.
To access these conditions:

Set your breakpoint using your normal method.

Right-click your breakpoint.

From the context menu, choose Condition (Figure 33) to get to the Breakpoints window (Figure 34).

Two buttons stand out at the bottom of the Breakpoints window. To specify a condition under which a breakpoint becomes active:

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 in the following way:

Sset your breakpoint as you would normally do.

Right-click your breakpoint.

From the context menu, choose Condition to get to the same screen (see Figure 62).

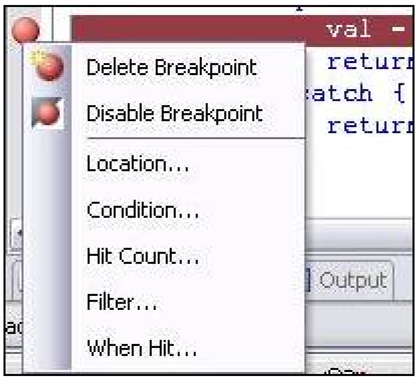
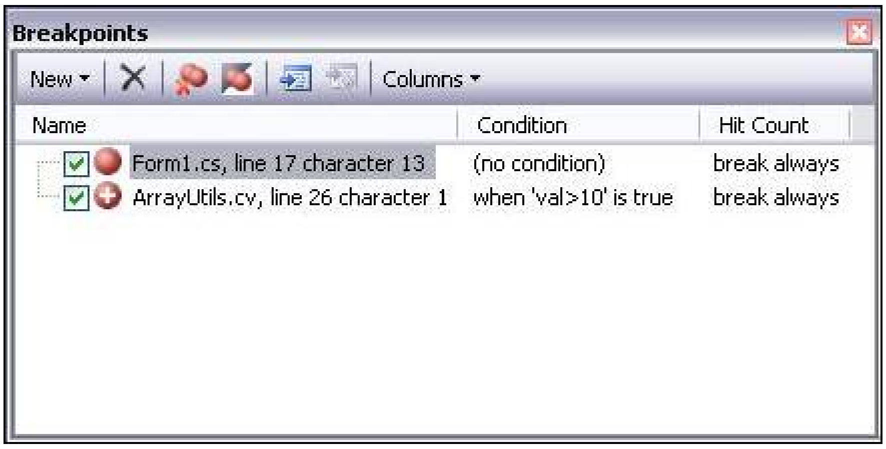


Figure 33. Set breakpoint condition.



Name	Condition	Hit Count
Form1.cs, line 17 character 13	(no condition)	break always
ArrayUtils.cv, line 26 character 1	when 'val>10' is true	break always

Figure 34. Breakpoints Window

To see and modify the condition in the Breakpoints window:

Open that window by selecting Debug > Windows > Breakpoints or

Pressing Ctrl-Alt-B.

A list of all breakpoints that you have set, along with their conditions will be displayed.

Note: 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.

3.12 - 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.

To save all these trace logs:

> Press Ctrl-S to save the entire output to a file.

To search through the Output window:

> Press Ctrl-F

You can even apply some of the other editor tips and tricks 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).

3.13 - Aligning UI Elements Automatically

If you are positioning UI elements in a Windows form, you have probably noticed various colored lines that appear on the form as you move or resize elements (see Figure 35). 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.

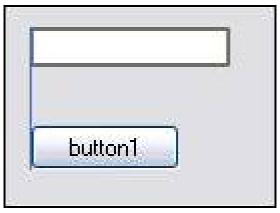


Figure 35. Align lines

To position UI elements without snapping to these colored lines:

Press Alt to turn off automatic alignment temporarily.

To switch back to the grid where all UI elements are aligned to a predefined grid:

Select Tools > Options > Windows Forms Designer > General and change LayoutMode back to SnapToGrid.

Note: 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.

3.14 - Adding a Standard Menu Strip

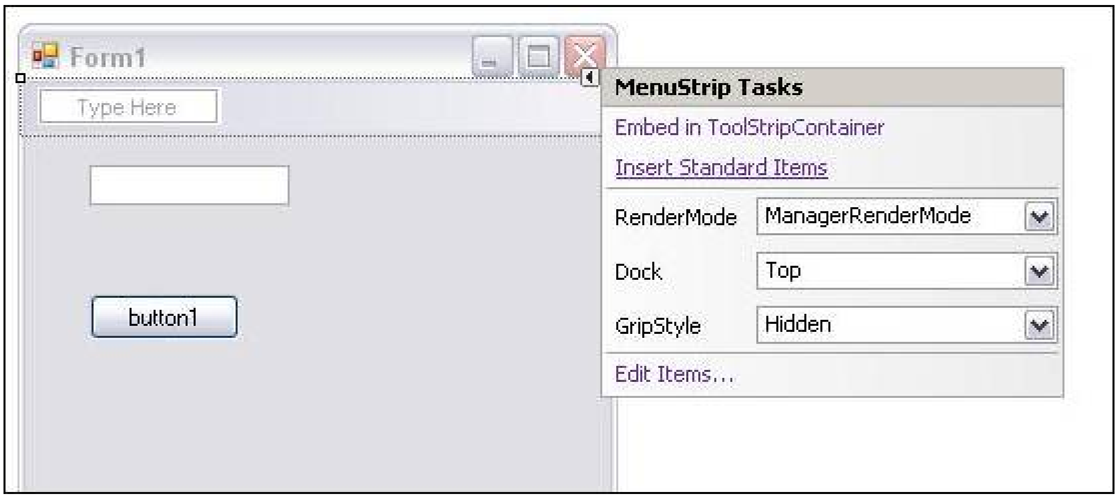
Standard Windows applications use a common set of top-level menu items. In most cases, they are File, Edit, Tools, and Help. VS.NET 2005 allows you to add these default menu items to your own Windows forms applications.   To add your own menu strip:

Drag a MenuStrip to your Windows form.

With the MenuStrip selected, the description panel below the Properties window shows an Insert Standard Items link (see Figure 74):

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.



RenderMode: ManagerRenderMode
Dock: Top
GripStyle: Hidden

Figure 36. Menu strip standard items

3.15 - 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.

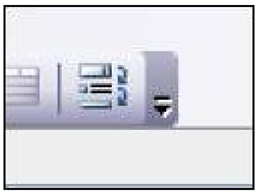


Figure 37 - Tab Order button on the left of the Layout bar

VS.NET 2005 introduces a new way to set the tab order: the Tab Order button on the Layout bar (see Figure 37).

> Click the Tab Order button to display the tab index for all UI elements on the form.

You now see all the tab indices.

> Click repeatedly on each UI element to set the tab order in linear fashion.

The first element you select is given 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

> Press the Escape key.

3.16 - Importing and Exporting IDE Settings

VS.NET is an extremely powerful tool with many things in the IDE that you can customize to suit your specifications. Because you will become accustomed to your particular settings, moving from one machine to another can cause problems if you are not able to move your IDE settings along with you.

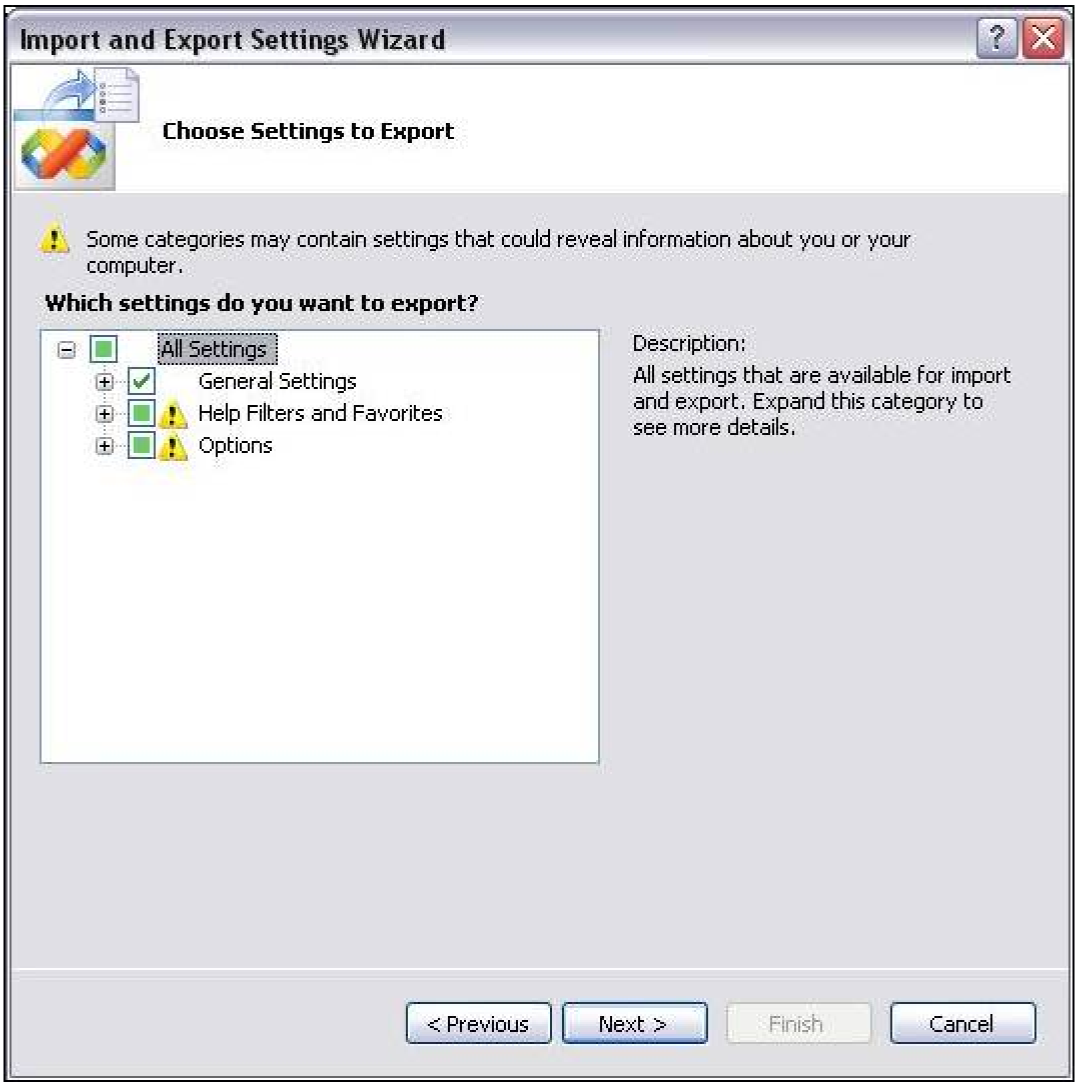
VS.NET 2005 allows you to export your IDE settings to an XML file (the extension is actually ".vssettings"). To import it into another instance of VS.NET on another computer:

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 38).

Check the options you want to be part of your profile.

Export them to the .vssettings file.



Some categories may contain settings that could reveal information about you or your computer.

Which settings do you want to export?

- All Settings
  - ✓ General Settings
  - Help Filters and Favorites
  - Options

Description:
All settings that are available for import and export. Expand this category to see more details.

Figure 38. Export VS Settings

Import the .vssettings file to another VS.NET IDE.

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).

To create a master .vssettings file for coordination between co-workers, e-mail it to all your team members so that they can import it individually. You can also create the single .vssettings file and place it on a well-known network share on the intranet. To obtain these settings have the members of your team:

Select 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. One advantage of this feature is 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.

3.17 - Closing All Other Windows

It’s very common to have a lot of files open at the same time when developing your program. After working for a while, you might have several dozen files open and want to close all of them except the one on which you are currently working.

To close all the open files:

Right-click one of the file tabs.

Choose Close All But This from the pop-up menu.

This option does exactly what it says (see Figure 39).

Other menu options new to VS.NET 2005:

1. Open Containing Folder
   -starts up Windows Explorer and opens the folder in which your file is located.

2. Copy Full Path
   -copies the full file path of the selected file into the Clipboard.

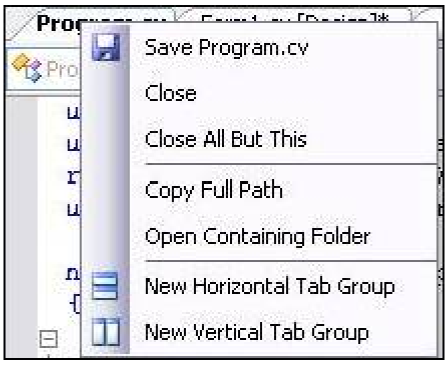


Figure 39. File tabs options

3.18 - Showing Shortcuts for All Buttons

Using and memorizing shortcuts whenever available, gives you a strong advantage when developing. It naturally increases your speed and therefore your efficiency. Keyboard shortcuts prove to be faster than manipulating the mouse. Many VS.NET menu and submenu items have these shortcuts which are seen every time you click the menu item.

This reminder is also available for the toolbar buttons. To see this reminder:

Select Tools > Customize

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.