

# **Beginning Visual C# 2005 Express Edition**

From Novice to Professional



Peter Wright

**Beginning Visual C# 2005 Express Edition: From Novice to Professional**

**Copyright © 2006 by Peter Wright**

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-549-7

ISBN-10 (pbk): 1-59059-549-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Jim Sumser

Technical Reviewer: Robert Lair

Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade

Project Manager | Production Director: Grace Wong

Copy Edit Manager: Nicole LeClerc

Copy Editor: Sharon Wilkey

Assistant Production Director: Kari Brooks-Copony

Production Editor: Laura Cheu

Compositor and Artist: Kinetic Publishing Services, LLC

Proofreader: Linda Seifert

Indexer: Broccoli Information Management

Cover Designer: Kurt Krames

Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit <http://www.springeronline.com>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail [info@apress.com](mailto:info@apress.com), or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Source Code section. You will need to answer questions pertaining to this book in order to successfully download the code.



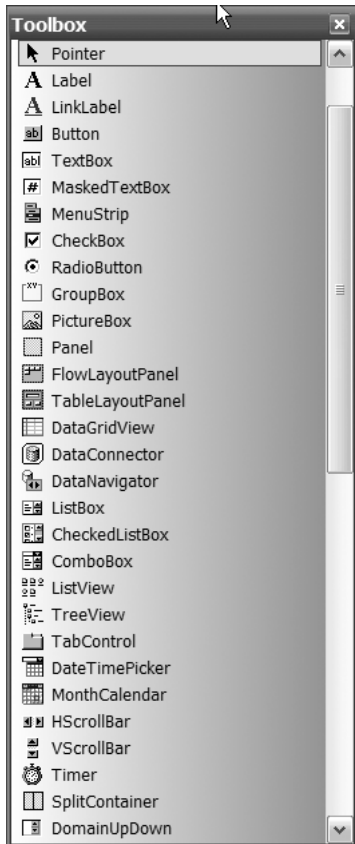
# How C# Express Helps You Code

**Y**ou've hopefully already noticed that Visual C# 2005 Express provides quite a few tools and neat IDE tricks to help you develop your applications. Before you dive into learning all about writing Windows applications, now would be a good time for me to introduce each of these features properly. These features, if you master them, can make you incredibly productive at your development work. A chap on the team I'm currently working on, for example, has memorized every single shortcut key within Visual Studio 2005 and can crank out code at a blinding pace. Whether that's a good thing remains to be seen, but as you'll see there are plenty of features given to you in C# Express to help you be really productive.

## Building a User Interface

Microsoft likes to say that Visual Studio .NET 2005 (and that includes C# Express) provides a great “design experience.” What they mean is, C# Express includes a bunch of tools to make life for the user-interface builder a heck of a lot easier than it used to be. Some of us, for example, still remember a time when it took about 50 lines of meticulously crafted code just to get a window to appear with the words “Hello, World” inside it. In C# Express this can be achieved without manually typing in any code at all.

The key to this ease is, of course, the Toolbox, shown in Figure 7-1.



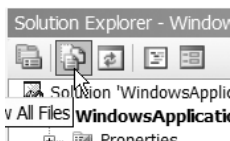
**Figure 7-1.** *The Toolbox is the source of all the elements of your user interfaces.*

The Toolbox contains the controls that you will want to use to build your user interfaces. You've come across it already in the few samples in previous chapters that required you to build up a Windows user interface. All you need to do is click and drag a control from the Toolbox onto the windows in your project. You can also double-click anything in the Toolbox, and it will magically appear on the current project window. It's worth bearing in mind, though, that the Toolbox is context sensitive; it will show only the tools that apply to what you're currently working on. For example, when you work on source code in the code editor, the Toolbox looks like the one in Figure 7-2.



**Figure 7-2.** *The Toolbox changes based on the work you are currently doing within C# Express.*

So, what really happens when you drag and drop something from the Toolbox onto a window? Well, code happens. Out of sight, C# Express creates a variable in your class (remember, a form or window is really a class) of the type of control that you select, and then goes ahead and sets up property values to size and position the control, as well as to set the properties you define in the Properties window. If you want to see the code that C# Express creates, just click the Show All Files button in the Solution Explorer. You can see this button in Figure 7-3.

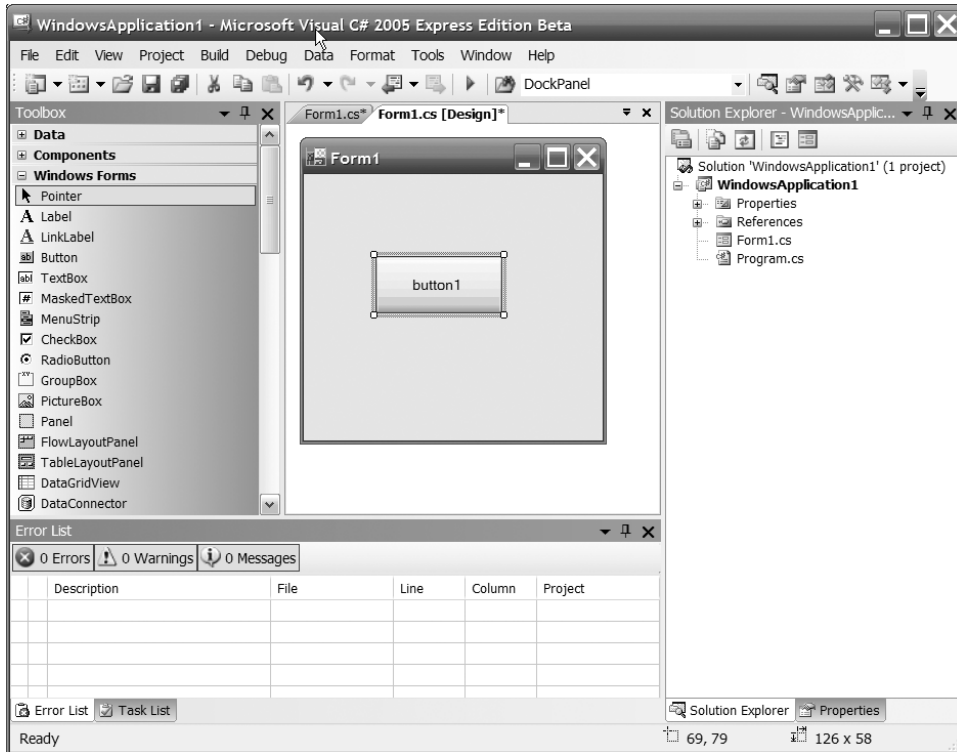


**Figure 7-3.** *The Show All Files button shows you all the files that C# Express creates behind the scenes. Handy for digging around.*

Let's go ahead and take a peek behind the scenes.

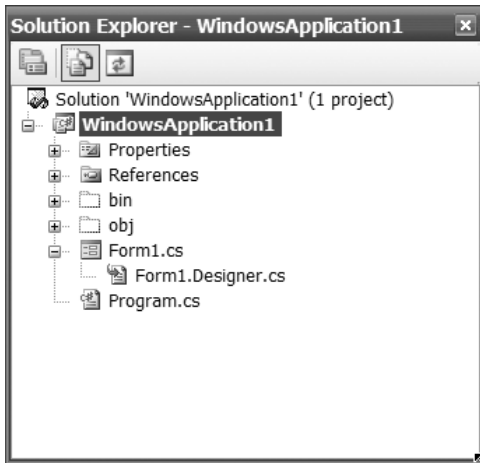
## Try It Out: Exploring the Behind-the-Scenes Code

Fire up a new Windows project in C# Express; then drag and drop a Button control from the Toolbox to the window in your application. You'll end up with C# Express looking similar to Figure 7-4.



**Figure 7-4.** Drag and drop a button onto a form. Behind the scenes, C# Express writes code in response.

With that done, click the Show All Files button and you'll see your Solution Explorer change, as in Figure 7-5.



**Figure 7-5.** When you click the Show All Files button, new files appear in the Solution Explorer.

Notice that a plus sign appears next to your form name in the Solution Explorer. Click the plus sign and you'll see the source file that C# Express has been working on behind the scenes, a file called `Form1.Designer.cs`. It's a partial class just like the others you looked at back in Chapter 5. Double-click that file to view the code.

Expand any hidden regions in the code editor and take a browse. The method that will be of most interest to you is the `InitializeComponent()` method. It starts off like this:

```
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();

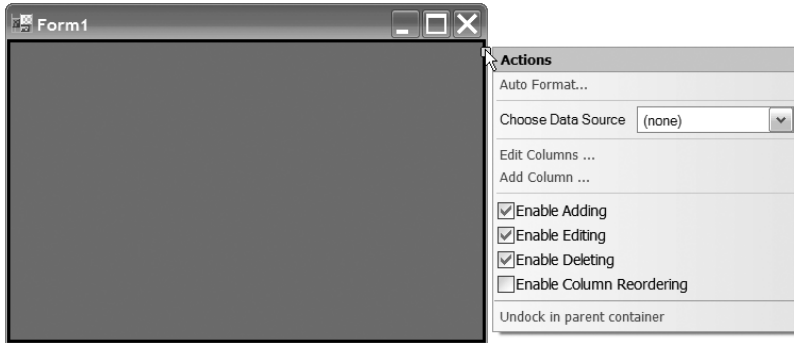
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(69, 79);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(126, 58);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
```

Notice the button stuff. You dragged and dropped a button on the form, and C# Express created a member in the class called `button1`. What you see here is a bunch of code setting up the `Location`, `Name`, `Text`, and `Size` properties of the button. In short, this code sets up the button exactly where you have positioned it on the form, at the size you set, with any other properties you set.

Obviously, the more controls you add to the form, and the more properties that you set on those controls, the bigger and more complex this code gets. But you never have to see this code or work with it. It's nicely hidden away in a partial class, letting you focus on the real code in the main class file.

## Using Property Smart Tags

Talking of properties, now is a good time to introduce the property smart tags feature. Start up a new Windows project, if you don't already have one open, and drag and drop a DataGridView control onto the form. What you'll see will look like Figure 7-6.



**Figure 7-6.** Controls now have smart tags attached to them to provide you with quick and easy access to common properties and property-editing tasks for that control.

The smart tag (the small arrow attached to the top right of the control) can be clicked to quickly access common properties and property-editing tasks associated with the control. In the case of a DataGridView for example, most people want to set up the grid's columns and general appearance, and also specify whether the grid should allow sorting, paging, and so on.

Click in the center of the form to select the grid (because the grid by default always takes up the entire space on the form) and press the Delete key to get rid of it. Then drop a simple text box onto the form. Notice that even a control as simple as the humble text box also has smart tags attached, although the text box's smart tag does nothing more than let you specify that the box will hold multiple lines of text.

Experiment with dropping controls onto a form at your leisure to see which do and which do not support smart tag property editing.

## Aligning Controls

I used to have a hard time making my Windows user interfaces look good. How big should the controls be? How much space should I put between them and between controls and the edge of my window? These are important questions, and it took a while to figure out how to consistently come up with the right answers so that my Windows applications looked professional, or at least had some semblance of professionalism to them.

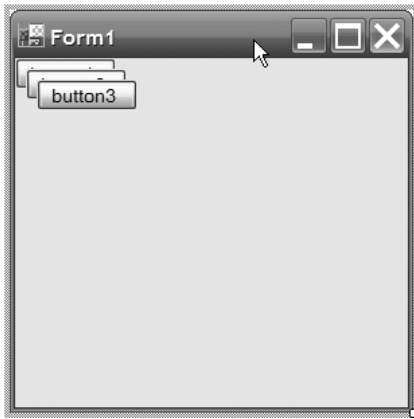
Microsoft realized this of course, and C# Express comes with a handy set of automatic tools to make life simple.

### Try It Out: Aligning Controls

Start up a new Windows project (if you already have one open, don't worry about getting rid of it—a great feature of C# Express is in-memory projects that you don't have to save and that don't litter your hard disk with files you don't need).

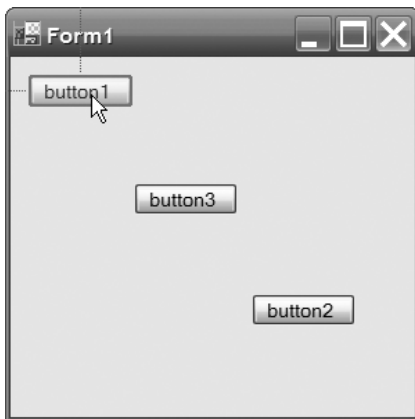


When the form appears, double-click the Button control in the Toolbox three times, so that you create three different buttons on your application's form. The form will look like mine in Figure 7-7.



**Figure 7-7.** Double-click the Button control three times to create three buttons on your application's form.

Drag the buttons so that they are spread all over the form, nicely spaced out. Now move one toward the top-left corner of the form. As you near the corner, you'll see guidelines appear, helping you position the button at exactly the correct, recommended distance from the top and left of the form, as you can see in Figure 7-8.



**Figure 7-8.** As you move the controls around, you'll notice guidelines appear to help you position the button precisely the right distance from the edges of your form

When you have both the left and top guides in view, release the left mouse button to drop the Button control. Now grab another Button control and drag it underneath the one you just positioned. This time different guides appear to show you the correct distance to place the control from the one above it, as well as where to position it so that its left or right edge is aligned.

Grab that third button and move it to the right of one of the others. Notice that yet another guide appears, this time aligning the text in the buttons (see Figure 7-9). You can still use the guides to align the tops or bottoms of the Button controls, but you can also align based on the control's textual contents, and all without you having to do a thing.



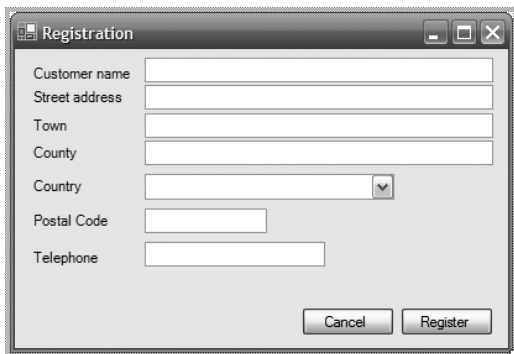
**Figure 7-9.** *Guides even let you align the contents of controls to each other.*

You'll come to love the guides over time.

---

## Setting Tab Orders

Imagine that you had a form like the one in Figure 7-10.

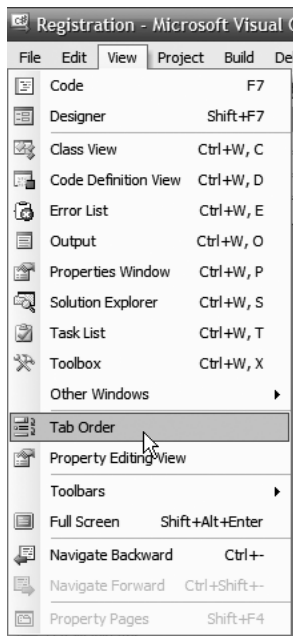


**Figure 7-10.** *All forms require some attention to navigation order, especially when they get a little complex like this one.*

That's a pretty complex form. Now imagine that you came across this form in an application you had installed—perhaps it's part of a registration dialog for an application you adore. Out of the box there are a few assumptions you and any other user would make about how this form works. Perhaps the biggest assumptions are that when you start editing the form, the cursor will start in the topmost text box, and that pressing Tab will take you down through all the text boxes in the form until you reach the bottom one. You would quickly lose faith in the program if pressing the Tab key moved you all over the form in a seemingly random order.

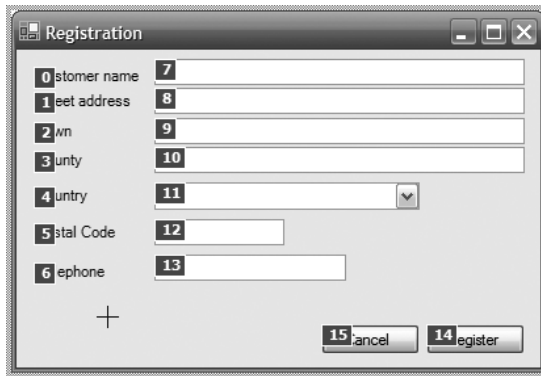
When you design your user interfaces, C# Express provides you with help in specifying the tab order of the controls on forms.

Take a look at the View menu (see Figure 7-11).



**Figure 7-11.** Visual C# 2005 Express's View menu

If you select the Tab Order item from the View menu, the form you are working on changes to Tab Order mode, just like mine in Figure 7-12.



**Figure 7-12.** Selecting *Tab Order* from the *View* menu shows you the current tab order of the controls on the form.

In Tab Order mode, all you need to do to set up the tab order of the controls on the form is to simply click on them one by one. Don't forget to click on the Label controls as well, because you can set a hot key for a label by using `&` in the label's Text property (for example, `&Next` to display `N`ext); when you select a label by pressing its hot key, focus moves to the next control in sequence.

When you are finished clicking away, just press the Esc key on the keyboard to leave Tab Order mode. What could be easier?

## Using IntelliSense

We've touched on IntelliSense in some of the examples in earlier chapters. It's the part of C# Express that pops up tip boxes in the code window to help you remember the syntax of methods to call and so on. You can even bring it up at any time in the code editor by pressing `Ctrl+spacebar`. It's a lot more than just a neat pop-up window, though.

IntelliSense can help you spot bugs and problems with your code before you run the program. It can also automatically resolve some problems for you. In fact C# Express's IntelliSense features can even be used to write code for you in some instances, thanks to a great feature called code snippets.

## Automatically Fixing Namespace Problems

Methods and properties live in classes, and classes live in namespaces. The .NET Framework alone includes a huge number of namespaces, and an even larger number of classes. Remembering what lives where can quickly become a real problem. IntelliSense can help.

Fire up a new console application and I'll demonstrate.

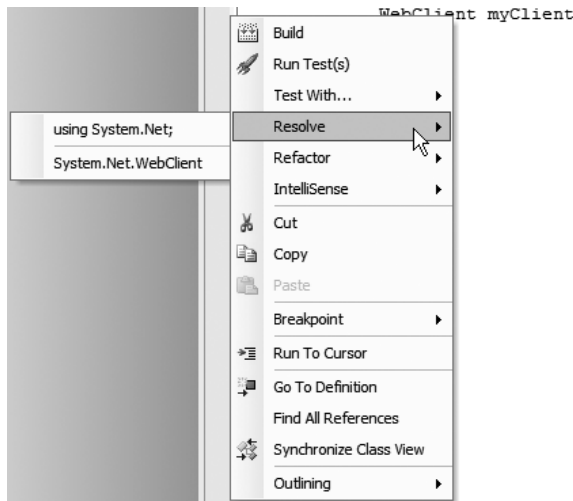
There's a class in one of the framework namespaces called `WebClient`, a great class that lets you download stuff from the Internet (web pages, files, and so on). The only problem I ever have with it, and indeed with a great many other classes in .NET, is remembering where it lives; is it `System.Web.WebClient` or `System.Net.WebClient`?

Add some code to the `Main()` function to create a `WebClient` object:

```
static void Main(string[] args)
{
    WebClient myClient = new WebClient();
}
```

If you try to run this program now, you'll get an error because you haven't told C# Express where to find the class (either by fully naming it or by adding a using clause to the top of the source file).

Right-click on the word `WebClient` in the source and you'll see an entry on the context menu that pops up called `Resolve`. Click it, and you'll see two options, as shown in Figure 7-13.



**Figure 7-13.** The *Resolve* option on the code window's context menu can help you resolve namespace problems.

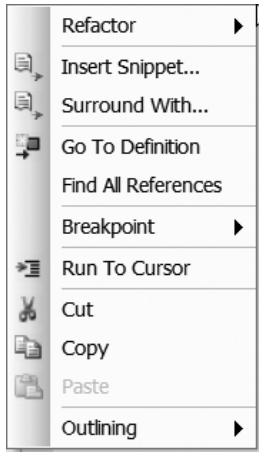
The first option on the `Resolve` menu, `using System.Net;` tells you that the `WebClient` class is part of the `System.Net` namespace, and you need a `using` clause in the code to tell C# that you want to use that namespace in your code. If you click that item, a new `using` clause will automatically be added to the code. Alternatively, the second option on the menu, `System.Net.WebClient`, will change the reference to `WebClient` in the code to `System.Net.WebClient` without you needing to add a new `using` clause. It's your call as to which approach you prefer on a daily basis, but it's easy to see how useful a feature automatic namespace resolution is.

## Using Code Snippets

Just as it's easy to forget which namespaces to use for which classes, new programmers in C# (or any C-based language) frequently have a hard time remembering how to structure blocks in the language. For example, when I learned C years ago, I used to have real issues remembering whether the condition or the increment came first in a `for` loop. (It's the condition, right?) IntelliSense code snippets can help you out here.

If you don't still have the console application open, create a new one just to get into the code editor so you can try this out.

Right-click on a blank line in the `Main()` method and this time choose the IntelliSense option from the context menu. Again you'll see two very interesting options, shown in Figure 7-14.

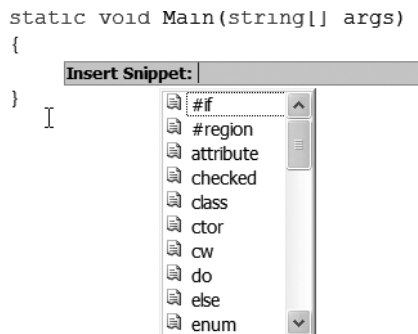


**Figure 7-14.** Note the *Insert Snippet* and *Surround With* items—these are part of *IntelliSense* and can write code for you.

The two items of most interest are *Insert Snippet* and *Surround With*.

The first item on the menu, *Insert Snippet*, dumps a bunch of prewritten code into the code window at the location of the cursor. The second, *Surround With*, is used when you have code selected in the window; it can be used to “surround” that code with something else—a try...catch block for exception handling, for example.

For now, click *Insert Snippet* and you'll see an expansion list appear, as in Figure 7-15.



**Figure 7-15.** When you choose to surround something, or to insert a snippet, the code snippet list appears.

Scroll down the list and select the *For* option. This is the expansion for a simple for loop. Click it, and new code will appear in the code window (see Figure 7-16).

```

class Program
{
    static void Main(string[] args)
    {
        for (int i = 0; i < length; i++)
        {
            }
        }
    }
}

```

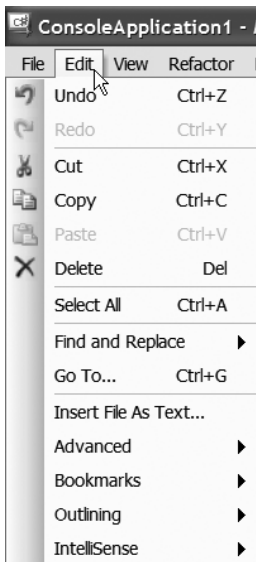
**Figure 7-16.** Clicking an item in the expansion list causes it to be added to the code window.

Notice how the `i` variable and `length` are both highlighted. These are the bits of code that you need to replace with code of your own; when you create IntelliSense code snippets, you can specify parts of the code to be references. References are displayed highlighted when the snippet is used to bring them to the developer's attention to replace. They even have ToolTips attached. Hover the mouse over one of the highlighted items and you'll see a Tooltip appear, offering a clue as to what this replacement is for.

Have fun exploring inserting more expansions, and even trying the Surround With option from the IntelliSense menu, and we'll catch up in the next section.

## Exploring the Edit Menu

It may seem like I'm being a bit picky spending time talking about the Edit menu in a Windows application, because nearly every Windows application has one. The Edit menu in Visual C# 2005 Express, though, is different. Take a look at Figure 7-17.



**Figure 7-17.** The Edit menu in C# Express

There are a few more options here than your average Cut, Copy, and Paste. The Edit menu in C# Express contains a great many tools that can help you immensely while you are coding.

### C# EXPRESS'S EVER CHANGING MENUS

Incidentally, you may find that your Edit menu doesn't contain nearly as many options as mine does. Visual C# 2005 Express changes certain menu contents based on the action you are currently performing within the IDE. For example, the Edit menu looks quite different if you are in the form editor compared to when you are working with the source code editor. The examples here are all talking about the Edit menu when you are working with source code in the source editor views.

The top two-thirds of the menu should be pretty self-explanatory if you've used Windows for a while. It's the submenus Advanced, Bookmarks, Outlining, and IntelliSense that hold the complicated stuff. You'll take a look at the first two in this section.

## Edit ► Advanced

The Advanced submenu is really all about reformatting the code in your code window.

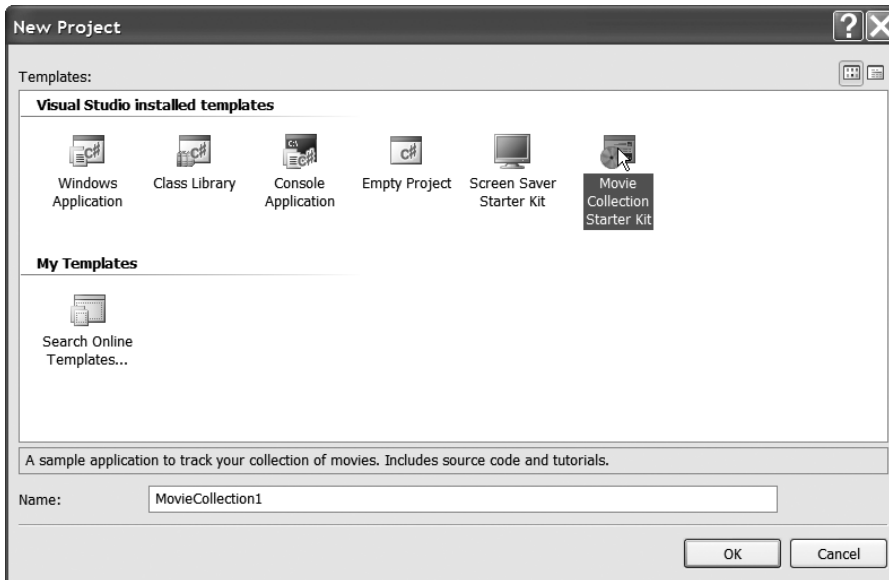
### Try It Out: The Edit ► Advanced Menu

On paper many of the features of the Edit ► Advanced menu look pretty dull. It's only when you see them in action and think about how you might put them to use yourself that their usefulness really shines through. So with that in mind, let's explore the menu in action.

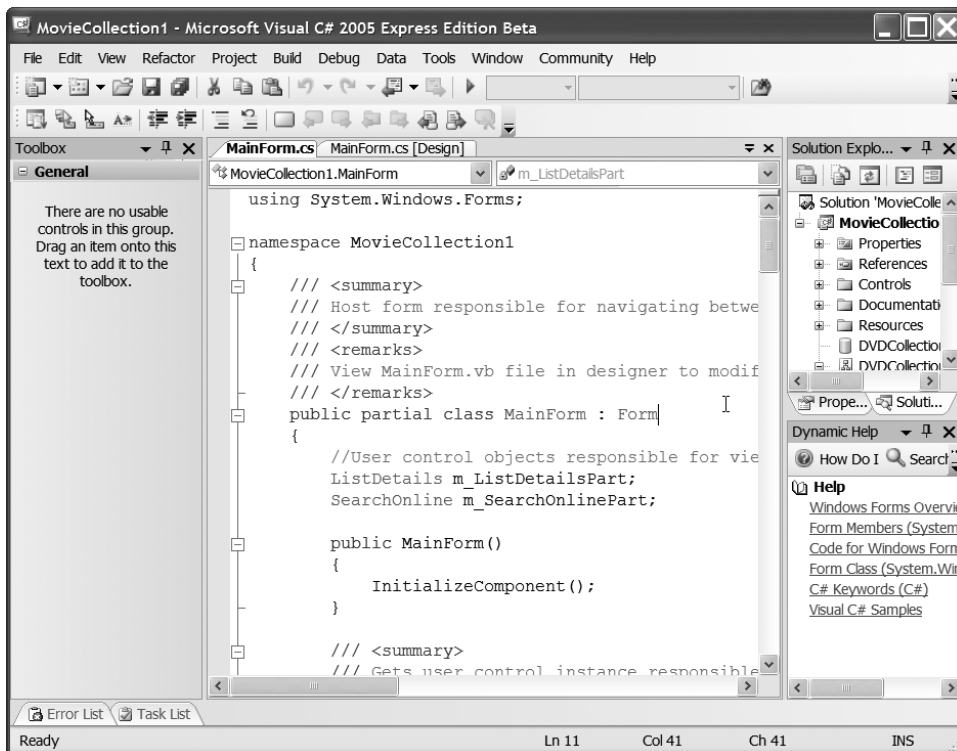
The best way to do this is to work with a project with quite a lot of code in it. Rather than key this all in, let's take a look at one of the starter kit apps. Start up a new project in the usual way and choose Movie Collection Starter Kit from the list of project templates, as shown in Figure 7-18.

When C# Express has finished working its magic and the solution is loaded up, double-click the `MainForm.cs` file; then press F7 to see the code editor view of the form. Your editor will look like mine in Figure 7-19.





**Figure 7-18.** Choose the Movie Collection Starter Kit to instantly get a project with a fair amount of code inside.



**Figure 7-19.** Load up the main form and press F7 to see all the code behind it.

The first options you'll take a look at are the Format options. C# Express's code editor is pretty good at making your code look good by automatically inserting tabs and indents where they should be, and so on. So, you're going to have to go out of your way to make the code look a little odd.

First, find the block of code that looks like this:

```
internal ListDetails ListDetailsPart
{
    get
    {
        //Initialize the variable with a new
        // object instance if nothing exists
        if (this.m_ListDetailsPart == null)
            //creating object
            {
                this.m_ListDetailsPart = new ListDetails();

                //site the control on the host user control and dock fill it
                this.TargetPanel.Controls.Add(this.m_ListDetailsPart);
                this.m_ListDetailsPart.Dock = DockStyle.Fill;
            }

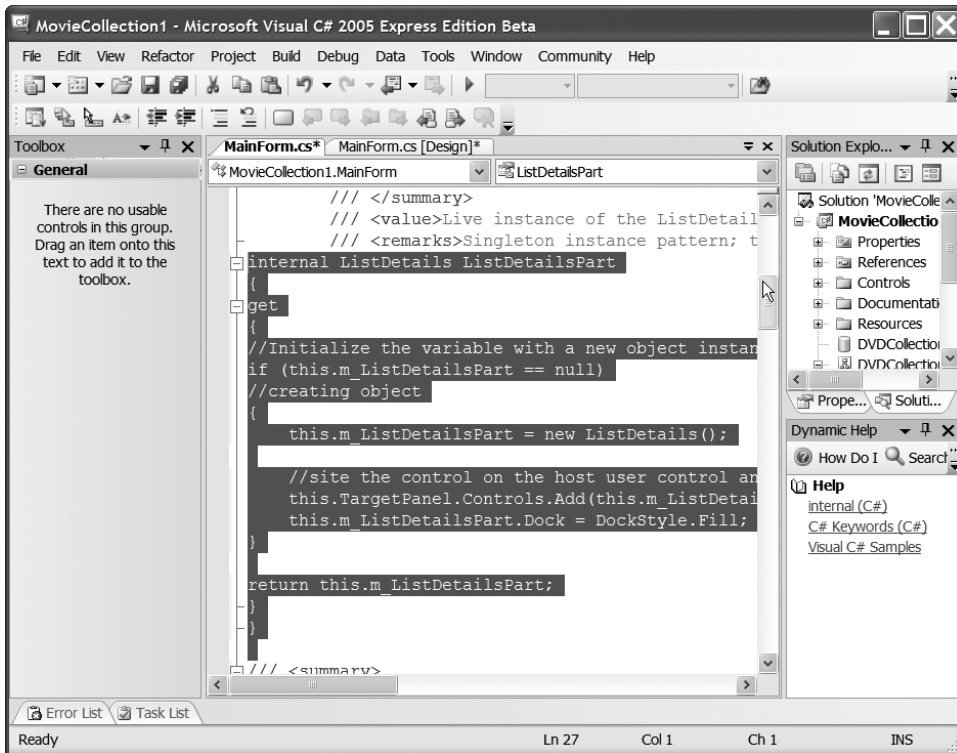
        return this.m_ListDetailsPart;
    }
}
```

Select this entire block of code by clicking and dragging the mouse over it and press the Shift and Tab keys together a few times to move the whole block up against the left edge of the editor window. Don't worry if you choose slightly more than the whole block, or slightly less—it doesn't alter what you're about to do. When you're finished, your editor should look like mine in Figure 7-20.

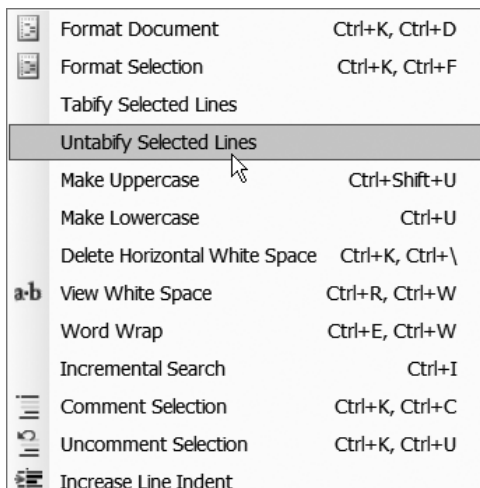
The Format Document and Format Selection items on the Edit ► Advanced menu are designed to rectify just this sort of ugliness, putting the code back the way it should be, according to Microsoft's code style standards. If you go ahead and choose Format Document from the menu at this point, you'll see the code jump nicely back into line.

If you really want to give this feature a workout, just press Ctrl+A to select all the text in the text editor, and then force the entire class left in the code editor window before choosing Format Document to put it back the way it should be.

While it was probably pretty obvious just what Format Document and Format Selection were going to do, you may be scratching your head at the Tabify and Untabify options on the menu (see Figure 7-21).



**Figure 7-20.** Pressing *Shift+Tab* moves the selected block of text to the left.



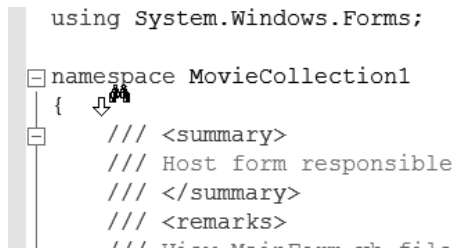
**Figure 7-21.** The *Tabify* and *Untabify* menu items are a little odd on first sight.

These are most often used when you want to cut and paste between C# Express and another application (usually something to do with the Internet). For example, when I post code to my blog, my blog host doesn't work very well with tabs in my code. So when I cut and paste nicely formatted code into my blog, it invariably ends up all left-aligned and screwy. Untabify fixes this, replacing any tabs used to format the code with spaces. Tabify, logically, does the opposite.

The next really interesting item on the Edit ► Advanced menu is the Incremental Search item, also accessible by pressing Ctrl+I. This feature has been in Visual Studio for some time, but it's surprising just how many people have not discovered its usefulness yet.

Incremental Search is a way of “honing in” on something in code, rather than just pressing Ctrl+F to access the Find dialog box, filling it out, and then clicking the Find Next button to move to the next match. It's absolutely perfect for that inevitable moment when you try to remember just what you named that variable a while ago; was it `OnlineSearch` or `SearchOnline`? Scroll the code in the window back up to the very top line, click in that line, and then press Ctrl+I.

The cursor will change to a down arrow with binoculars, like mine in Figure 7-22.



```
using System.Windows.Forms;

namespace MovieCollection1
{
    /// <summary>
    /// Host form responsible
    /// </summary>
    /// <remarks>
    /// ...

```

**Figure 7-22.** Starting an incremental search changes the cursor to the incremental search symbol.

Now just type in (slowly, so you can see what happens) the word **online**.

Notice how as you type, items in the code window are highlighted, showing you the next bit of source code that matches what you have typed so far. We're looking for a variable declaration here, but the first match that gets highlighted for the word **online** is a comment, as you can see in Figure 7-23.



```
and search online modes of the a

onal properties.
```

**Figure 7-23.** Incremental Search will highlight the first match. Hot keys can be used to move from match to match.

To move to the next match, press Ctrl+I. Do it twice now. Notice how the highlight jumps to the next source line that matches the word `onLine`, in this case the variable definition that you were looking for. You can move back to the previous matching line by pressing Shift+Ctrl+I. To leave Incremental Search mode, just click anywhere in the source code window or press the Esc key.

When coding, it's often handy to stop blocks of code from running. I do this when I want to try out an alternate solution to a problem but don't really want to lose the solution I already have. The Comment Selection and Uncomment Selection items on the Edit ► Advanced menu make this easy.

Select the block of code inside the `ListDetailsPart` get statement, as in Figure 7-24



**Figure 7-24.** Select the block of code inside the first get statement.

Now, either choose Comment Selection from the Edit ► Advanced menu, or press Ctrl+K and then Ctrl+C. The selected block will be commented, as you can see in Figure 7-25.

```
I get
{
    ///Initialize the variable with a new object inst
    ///if (this.m_ListDetailsPart == null)
    ///creating object
    ///{
    //    this.m_ListDetailsPart = new ListDetails();

    //    //site the control on the host user control
    //    this.TargetPanel.Controls.Add(this.m_ListDe
    //    this.m_ListDetailsPart.Dock = DockStyle.Fil
    //}

    //return this.m_ListDetailsPart;
}
}
```

**Figure 7-25.** Pressing Ctrl+K and then Ctrl+C comments out the selected code.

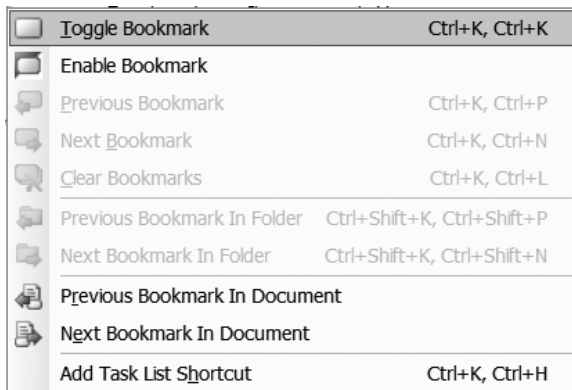
If you were to run the project now, that block of code would never execute, because it's commented.

To put the code back the way it was, just choose Uncomment Selection from the Edit ► Advanced menu, or press Ctrl+K and then Ctrl+U. Personally, I find the shortcut keys a lot quicker to work with than having to find and select a menu item.

## Edit ► Bookmarks

If you're working on a particularly large class, it's sometimes handy to be able to quickly jump to and from specific places in the code. For example, if I have a bunch of instance variables declared at the top of the class, I find it handy to be able to quickly jump to them to refresh my memory as to what certain ones are named before I use them in code.

C# Express includes a bunch of menu options on the Edit ► Bookmarks menu, as well as shortcut keys to set and navigate around bookmarks in your code. You can see the menu in Figure 7-26.



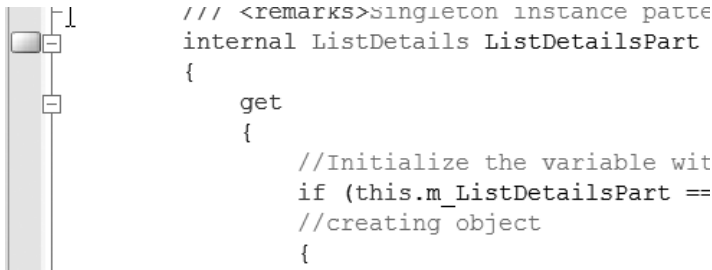
**Figure 7-26.** The Bookmarks submenu lets you set and navigate around bookmarks in your code, and shows you the handy shortcut keys to use them.

### DOUBLE KEY SHORTCUTS

These shortcuts are a little strange, aren't they. Ctrl+K then Ctrl+C to comment stuff, Ctrl+K twice to set a bookmark. The reason is that all the code-editing hot keys begin with Ctrl+K. So, you use Ctrl+K to go into an advanced editing mode, then Ctrl+C for comment code ("C" for comment), Ctrl+K again to set a bookmark (bookmark ends in "K"), and so on. You soon get used to it—like anything, the more time you spend in the editor, the quicker things like this become second nature.

## Try It Out: Working with Bookmarks

Setting a bookmark is easy. Just click on, or move the cursor to, the line of code that you want to bookmark and then press Ctrl+K twice. You'll see a small icon appear beside the line of code, as in Figure 7-27.



**Figure 7-27.** Setting a bookmark on a line of code causes a small icon to appear beside the bookmarked line.

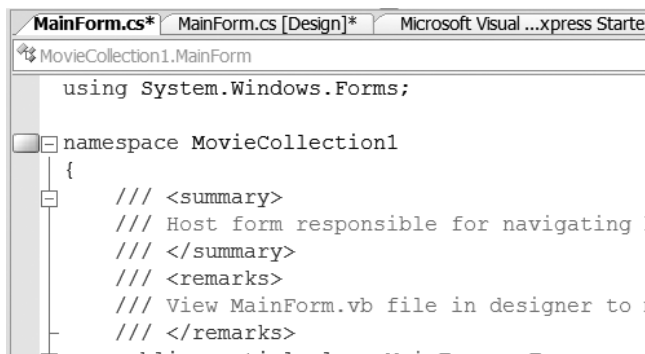
Go ahead and set a few more bookmarks into the code; don't worry about where too much.

To move around the bookmarks, just use Ctrl+K, Ctrl+P to go to the previous one (remember, Ctrl+K to go into an advanced editing mode, then P for previous), and Ctrl+K, Ctrl+N to go to the next one. As always, you can get at these through the Edit ► Bookmarks menu as well.

You can clear out a single bookmark simply by moving to it and pressing Ctrl+K and then Ctrl+K again. Alternatively, you can clear out all the bookmarks in a file by using Ctrl+K, Ctrl+L. Do that now to clear out the bookmarks you set.

Now, let's take a quick look at how you would use this stuff for real when working on code.

Move to the top of the MainForm source in the Movie collection sample and bookmark the namespace line, as in Figure 7-28.



**Figure 7-28.** Bookmark the namespace line.

You can pretend that this marks the point at which some really important instance variables are declared.

Now move down to the very end of the source code and you'll find a method called `ShowSearchOnlinePart()`. Put the cursor somewhere in that method.

Imagine that you were working on this code and needed to glance back at the instance variables you bookmarked (pretended to) earlier in the source. All you have to do is press `Ctrl+K, Ctrl+K` to remember where you are (do it now), and then `Ctrl+K, Ctrl+P` to go back to the previous bookmark (go ahead and try it). When you're finished, just hit `Ctrl+K, Ctrl+N` to return to the next bookmark, and then `Ctrl+K, Ctrl+K` clear the bookmark out.

I know it all sounds terribly awkward, but it really isn't, and once you get used to it, bookmarks are a great way to navigate around complex source files.

Of course, you'll also want to use bookmarks across more than one file. Perhaps you want a bookmark set in one class, and the ability to jump to it while you work on another. The Previous Bookmark in Folder, and Next Bookmark in Folder items on the Edit ► Bookmarks menu let you do just that.

Try it out. Set a bookmark somewhere in the code for `MainForm`, and then take a look at `ListDetails.cs` in the `MovieCollection1` application (just click it once in the Solution Explorer and then press `F7` to view the code).

Now press the rather ornate shortcut `Ctrl+Shift+K` followed by `Ctrl+Shift+P`. Notice how you instantly jump out of the `ListDetails` code and back into the `MainForm` code. Of course, to get back to the `ListDetails` code, you'll need to set a bookmark before you jump.

---

## Summary

Visual Studio has always been the IDE of choice for millions of developers around the globe simply because it makes programming easier. With Visual Studio 2005 and the Express tools, it just got even better.

In this chapter you took a look at some of the GUI aids for Windows user-interface design, as well IntelliSense, and the features hidden away in the C# Express Edit menu. I picked these items for a full explanation because they warrant it, especially if you've never used Visual Studio before. There are still a lot of other neat features to explore on the menus, so feel free to click away to your heart's content before moving on. Don't forget, online help can get you out of a bind if you find something particularly strange.

You're all set now to start learning how to write Windows applications. See you in the next chapter.