



65 Tips for Migrating to Visual Studio .NET

Table of Contents

Contributor	Tip Number
Andrew Brust	1-3
Avonelle Lovhaug	4-6
Bill Vaughn	7-10
Dan Appleman	11
G. Andrew Duthie	12-14
Jeff Prosis	15-26
John Alexander	27-29
Keith Pleas	30-31
Ken Getz	32-34
Laura J. Baker.....	35-37
Michael Amundsen	38-40
Paul D. Sheriff.....	41-45
Peter Vogel	46-47
Robert Patton.....	48-50
Rockford Lhotka.....	51-52
Roger Jennings.....	53-58
Todd Kimble.....	59-61
Yasser Shohoud	62-65

[Register for VSLive! San Francisco Now](#)

Contributed by Andrew Brust, who is presenting at VSLive!

Reach Andrew at www.progsys.com

1. Close the Dynamic Help window unless you're actively using it. Keeping it open all the time forces it to frequently load topics and can slow down Visual Studio .NET significantly.
2. Need an easy way to build a SqlDataAdapter object against a table at design time, without even having to use a Wizard? Just drag and drop a table from the Server Explorer window onto your Windows Form or Web Form and Visual Studio .NET will automatically create a SqlDataAdapter object against the table, and will build the necessary Select, Update, Delete and Insert commands as well. As a bonus, if you didn't already have a SqlConnection object pointing to the database containing the table, Visual Studio .NET will create that for you as well.
3. Want to create a strongly typed DataSet without having to do a lot of work at design time? You can! Just create a DataSet in code, and export its schema to an XSD file using the DataSet's WriteXMLSchema method (which accepts a file name as a parameter). After doing this, you can add the file to your project, open it in the XML Schema Designer by double-clicking on it in the Solution Explorer window, then Choose Schema/Generate Dataset from Visual Studio .NET's menu to create a strongly typed DataSet based on it.

Contributed by Avonelle Lovhaug, who is presenting at VSLive!

Reach Avonelle at www.coolbits.nu

4. With ASP.NET (just as with ASP), it is easy to change the source code files and reload the page in a browser to test. However, some files require more effort. Remember that when changing the global.asax, you will need to recompile the project in order for changes to take effect.
5. With Visual Interdev, FrontPage server extensions were used for integration with SourceSafe. While this is still possible, the recommended approach is to use "File Share" mode. Set up your Web applications with file share mode, then use "Add solution to source code control" to easily add the entire project to SourceSafe.
6. There are two quirks of the Page_Load event in ASPX pages in ASP.NET that should be kept in mind:
 - a. Sometimes it may appear that the Page_Load event for your ASP.NET pages is executing multiple times. One possible reason for this behavior could be if the AutoEventWireup value in the ASPX page is set to True. If so, then the code "Handles MyBase.Load" after the "Sub Page_Load (ByVal Sender as System.Object, ByVal e as System.EventArgs)" is unnecessary. Since Visual Studio .NET will automatically add the handles section for you, you can easily leave AutoEventWireup as False.

[Register for VSLive! San Francisco Now](#)

- b. Occasionally it may appear that code placed in the click event for a button is not being fired. You should check the Page_Load event to make sure that any code which loads data (such as code for binding data to drop down lists) only occurs during the initial load of the page, and not during subsequent post backs. An easy way to check this is to add a test of the Page.IsPostBack value in your Page_Load event - False means it is the first time the page is loaded, and True means that a post back has occurred.

Contributed by Bill Vaughn, who is presenting at VSLive!

Bill's new book ADO .NET and ADO Examples and Best Practices—Second Edition (Apress) will be available at VSLive! Reach Bill at www.betav.com

7. Converting Control Arrays to .NET

Implementing control arrays has always been one of the most common ways that Visual Basic 6 developers have reduced code and application complexity. The use of common event handlers dealt with the events associated with the controls.

Those now working with Visual Basic .NET have discovered that control arrays (as such) are not supported. You'll also find that if you convert an existing Visual Basic application over to Visual Basic .NET using the Conversion wizard, Visual Basic .NET uses a compatibility interface to fake the control array. It's said, however, that these compatibility interfaces won't be supported forever (but at least through next week). I think it better to code "control arrays" using "native" techniques.

It turns out there is an easy way to get a single event handler to trap events from a variety of operations. Just add additional events to the Handles argument of the event handler. The following example illustrates how I added the TextChanged event to the Handles argument for both TextBox1 and TextBox2. You can add as many events as you choose to the Handles argument, but I suspect they had all better pass the same arguments as the source event prototype.

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object,  
    _  
    ByVal e As System.EventArgs) _  
    Handles TextBox1.TextChanged, TextBox2.TextChanged
```

The next issue is figuring out which TextBox fired the event. Well, in Visual Basic .NET the event handler returns an argument named "sender" which points to the object that fired the event. To access this object and inspect it's properties, declare a like object (in this case a TextBox) and address it with Sender.

```
Dim tbSent As TextBox = sender
```

After this is done, you can easily reference the properties of the control firing the event.

[Register for VSLive! San Francisco Now](#)

```
Select Case tbSent.Name
    Case "TextBox1"
        TextBox3.Text = "TextBox1 contents:" &
tbSent.Text.ToString
    Case "TextBox2"
        TextBox3.Text = "TextBox2 contents:" &
tbSent.Text.ToString
    End Select
End Sub
```

Unfortunately, the Index property that used to be passed along in “classic” Visual Basic was dropped in Visual Basic .NET. Since the Index property did not have to be any particular number (just a unique integer), I used it to pass along a value so my event handler could simply use it as a data value.

For example, when I wanted to illustrate the use of different ADO LockType properties, I would create a set of CheckBox controls with their Index property set to the various LockType enumerations. When the CheckBox Change event fired, I could simply assign the Recordset LockType property from the Index.

As a work-around, you might consider using the Tag property to pass along control-specific values or simply set the Name property in a way that this additional value can be referenced. It would be nice to have our Index property back—but I’m not holding my breath.

Another tip: When creating your application you might delete a control on a form and expect the code to remain behind in your project. It does—at least most of it. In Visual Basic .NET, when you delete or cut a control from a form, the event handlers already coded for the control are altered; Visual Basic .NET strips the Handles keyword and arguments and tosses them. When you paste in your new control you’ll find that Visual Basic .NET will create a new event handler for it with the Handles keyword filled in. Now you have two (or more) event handlers—one with a Handles keyword and several without. I have started saving the Handles clause in a comment to make sure this does not cause problems. This way I can copy it back into my old handlers.

8. Saving Oft-used Code in the Toolbar

One of the coolest features of the new Visual Basic .NET IDE is its ability to save code snippets in the Toolbar. This feature is really easy to use. When in the code pane, select a block of code you want to reuse (I save the application Dim statements and ConnectionString values), and simply drag it to the Toolbar. Once it’s there, getting it back is just as easy; just drag it back to your code pane and drop it. These values are saved from project-to-project so it’s easy to paste in boilerplate text into your applications.

9. Choosing the “Right” Data Access Vehicle

I think we’re going to have quite a bit of discussion in the next few years about which data access interface is best to use. Since ADO .NET does not support all of the COM-based ADO (I call it ADOc) functionality, developers have an important choice to make. I just spent the last nine months writing a new book *ADO .NET and ADO Examples and Best Practices* (Apress) where I help developers make this decision.

At this point developers can choose to use ADOc and get access to server-side optimistic or pessimistic cursors as well as static, keyset, dynamic, or disconnected (batch optimistic) cursor implementations. In ADO .NET they can choose from optimistic concurrency through the DataSet or use the low-level (and faster) DataReader to return a “firehose” stream.

The problem with using ADOc is that I expect the performance to be somewhat degraded because each and every reference to the unmanaged MDAC DLL (ADO) has to traverse a translation routine called COM Interop—twice—once going and once on returning from MDAC. The ADOc approach also depends on what I call OSFA (one size fits all) data interfaces such as OLE DB.

Yes, ADO .NET exposes it’s own OSFA OleDb .NET Data Provider, but it also implements the new SqlClient provider which talks directly to TDS. This is the first TDS interface since DB-Library (and VBSQL). This (entirely) new way to access data will revolutionize the way developers access data. Okay, it only works against SQL Server. But that does not mean that other vendors (or Microsoft) can’t create other “native” data access interfaces that can easily outperform OLE DB or ODBC for that matter.

Okay, assuming you can do without the ADOc features like keyset cursors and pessimistic locking. Should you use the DataSet or the DataReader to manage your data? It turns out that the DataSet uses the DataReader for its low-level I/O. The DataReader is a very simple, but very fast, interface. It exposes a single row at a time. While you can fetch the individual columns into an array, you won’t want to because of the additional overhead and performance penalty. This means you loop through the rows one-at-a-time and each column is fetched individually using datatype-specific Get statements.

Even with all of this code, fetching data, using the DataReader is still faster than the Fill method used to populate DataTables managed by the DataSet. That said, consider that the DataSet is an order of magnitude more sophisticated, more flexible, and more powerful than the ADOc Recordset. It can handle hierarchical data with ease (no special Shape statements needed) and seamlessly binds to complex bound controls. It can also update the data through developer-defined Command objects that contain the appropriate UPDATE, DELETE, and INSERT SQL statements or stored procedures.

Yes, the debate will continue as there are many new and hitherto untested applications that will attempt to make the best of this new set of data access techniques.

10. Make sure to Close Your Database Connections

One of the big (really big) differences in the new .NET Common Language Runtime (CLR) is its garbage collector. It seems to be modeled after the technique used in New York during the last garbage worker's strike. Instead of tearing objects down when set to Nothing, objects (like ADO .NET Connection objects) are simply left out on the street (in memory) to rot.

When the garbage collector has no more curb space (memory) it goes through and collects up all of the least-frequently-used objects and sends them to the land fill bit bucket. This means that when you let a Connection object (a SqlConnection, OleDbConnection, or OdbcConnection) object fall out of scope without closing it, the connection remains open and stuck in the connection pool until sometime in the spring. ADO .NET does not guarantee to ever close these connections. If you have lots of free RAM, this problem is made worse, as the CLR garbage collector does not feel the need to collect the trash until all available RAM is full.

This means it's essential (critical) to use the Close method on your opened Connections *before* they fall out of scope. When you use a DataReader, you can ask it to automatically close the connection for you, but this does not happen *until* you close the DataReader itself. If you don't want your application to start stinking up the office, I suggest a more rigorous approach to your own housekeeping.

Contributed by Dan Appleman, who is presenting at VSLive!

Reach Dan at www.desaware.com

- 11.** Don't believe everything you read. There's a rumor going around (that has supposedly appeared in print) that there's a difference between the way C# and VB .NET pass method parameters when called by reference - that VB .NET makes a local copy of parameters and then prior to return copies the local copy back to the original variable. It's a lie. To prove it, create sample VB .NET and C# programs that pass a parameter to a function by reference. Then use ildasm to disassemble the resulting IL code. You'll see that VB .NET and C# produce exactly the same intermediate code for by reference method calls.

Contributed by G. Andrew Duthie, who is presenting at VSLive!

- 12.** Even if you're still working mostly with classic ASP, take the time to learn as much as you can about ASP.NET now...and apply those lessons to your current ASP code. For example, in Visual Basic .NET, the default for parameters passed to procedures is ByVal, which is the opposite of the behavior in Visual Basic 6 and VBScript. By explicitly adding ByVal or ByRef to your existing code, you can reduce the likelihood of it breaking when you migrate it to .NET

[Register for VSLive! San Francisco Now](#)

- 13.** It's never too early to start separating code from business logic. ASP.NET provides several ways to reduce the intermingling of code and HTML. An important one is new restrictions on the kinds of code you can write in render (<% %>) blocks, and server-side <script> blocks. You can no longer write procedures in a render block, and you are required to use procedures in server-side <script> blocks. These restrictions will help you avoid the spaghetti code habit so familiar to ASP developers.
- 14.** Get out of the habit of using Response.Write and using render blocks. Both of these common techniques in classic ASP are less efficient (and result in more difficult to maintain code) than newer techniques available in ASP.NET. For example, if you need to write text to the page, consider placing an ASP.NET Label or Literal control on the page, then setting its Text property to the desired text. This allows you to keep the code that writes the text where it belongs (in an event handler such as Page_Load in a server-side <script> block), but still allows you precise placement of the text based on the location of the server control tag:

Contributed by Jeff Prosise, who is presenting at VSLive!
Reach Jeff at www.wintellect.com

Rules of thumb for performance optimization with ASP.NET:

- 15.** Don't use server controls when static HTML will do.
- 16.** Use StringBuilder to build strings dynamically.
- 17.** When using Hashtable and other collection classes, try to initialize them with an item count to avoid unnecessary reallocs.
- 18.** Design your apps to be compatible with server farms.
- 19.** Do as much as possible on the client side; avoid unnecessary postbacks.
- 20.** Don't call old COM components if you can help it.
- 21.** Used stored procedures for common database operations.
- 22.** Use caching whenever possible (ASP.NET has some wonderful caching features).

[Register for VSLive! San Francisco Now](#)

Coding mistakes VC++ programmers should avoid with their first C# projects:

- 23.** Say you implement a Hashtable class in C++. The following statement declares an instance of that class on the stack:
`Hashtable table;`

If you write a Hashtable class in C# (or use the .NET Framework Class Library's Hashtable class), the same statement compiles fine but DOESN'T create a Hashtable; it simply creates a reference to one. You should write it this way instead:

```
Hashtable table = new Hashtable ();
```

Otherwise you'll generate a null reference exception the first time you access the Hashtable.

- 24.** Another common problem that C++ programmers encounter is that in C#, conditional expressions MUST evaluate to booleans. In C++, this is perfectly legitimate code:

```
int x = 0;  
if (x) { ... }
```

In C#, these statements won't compile. They have to be written this way instead:

```
int x = 0;  
if (x != 0) { ... }
```

Subtle difference, but enough to cause a headache now and then.

- 25.** Another potential trouble spot for C++ programmers moving to C# is the difference between reference types and value types. Your code can behave weirdly if you're not cognizant of whether a given data type is a reference type or a value type.
- 26.** Most of all, C++ programmers must remember that in C#, destruction is non-deterministic. Classes that wrap file handles and other unmanaged resources are mistakes looking for a place to happen if not used properly. See the previous question for more information.

Contributed by John Alexander, who is presenting at VSLive!
Reach John at www.gasullivan.com

- 27.** Deploy assemblies as private whenever possible. Private assemblies live in the application folder and aren't registered in the GAC. Assemblies should only be

[Register for VSLive! San Francisco Now](#)

deployed as shared when they contain functionality that several apps will use. The CLR has to go through more work when resolving types within shared assemblies.

- 28. If you are creating Web forms in the HTML view, a quick way to validate your code is to click the design tab, or press Control-Page Up on your keyboard. Before switching to design mode, Visual Studio will validate your code and notify you of any errors.
- 29. Use the Environment.StackTrace static property to inspect the stack trace without throwing an exception. This might prove beneficial during testing to log the execution of a long running business process that involves multiple objects or might complete one or more database updates.

Contributed by Keith Pleas, who is presenting at VSLive!

Reach Keith at www.deeptesting.com

- 30. In the Solution Explorer, click the "Show All Files" button on the mini toolbar to see code-behind Web Forms pages directly.
- 31. The SDK samples are buried in subdirectories many levels deep. To add a context menu item to a folder that creates a command window for that folder, create and run the following 2 line VBScript:
 - a. Set WshShell = CreateObject("WScript.Shell")
 - b. WshShell.RegWrite "HKCR\Folder\Shell\Command window\Command\", "CMD /K cd /d %L"

Contributed by Ken Getz, who is presenting at VSLive!

Reach Ken at www.mcwtech.com

32. Renaming ASP.NET Files and Folders

ASP.NET projects created in Visual Studio .NET bury a few path dependencies in places that you wouldn't expect, and sooner or later you'll get bitten by these. If you must rename files within your project, make sure you do it within the Solution Explorer in Visual Studio .NET. If you do, VS .NET will take care of all the dependencies based on the file names. If you must rename a folder, things get more complicated. Once you've renamed the folder using Windows Explorer, you'll need to fix up the path in two locations: in YourProject.sln, and in YourProject.vbproj.Webinfo (replace vbproj with projects created for your language of choice). You'll easily find the old path buried in these files, and you'll need to fix them up by hand.

The same issues apply when some other developer hands you a project to be loaded on your machine. If they developed the project as `http://localhost/demo` on their machine, you'll need to create a folder (name it whatever you like) on your own

[Register for VSLive! San Francisco Now](#)

computer. Set up a virtual root that points to that folder, and name the virtual root demo. You should be able to double-click on the vbproj or sln file, and simply load the solution within VS .NET. If you change the virtual root name, you'll need to again fix up the names in the sln and vbproj files.

33. Close Your Connections

It's important that you close Connection objects once you're done with them. Because of the changes to the way objects are destroyed, in the .NET framework (compared to the way things used to work in VB6), it's quite possible that connections you thought were disposed are, in fact, still sitting around consuming resources. Imagine this scenario: You create a component that returns a DataReader object to you, and you use that DataReader in your code. You can close the DataReader from your client code, but you have no way to get at the connection that provided the DataReader--you end up with an open connection that you can't close.

The solution is simple: if you want to ensure that the Connection object's Close method gets called automatically when you're done with the DataReader, indicate that when you retrieve the DataReader, using the CommandBehavior.CloseConnection constant:

```
Dim dr As SqlDataReader
Dim cmd As SqlCommand
' Set up the Command in here...

dr = cmd.ExecuteReader(CommandBehavior.CloseConnection)
Return dr
```

Once you set things up this way, when you call the Close method of the DataReader object, the .NET Framework will automatically close the connection for you. No more dangling Connection!

34. Untangling the Windows

If you ever get the various dockable windows in Visual Studio .NET so out of whack that you simply can't figure out how to unwind them back to any sane locations, you have two alternatives. From within Visual Studio, you can simply choose the Tools|Options menu, and then click Reset Window Layout button. You can also wait until you shut down Visual Studio, and you'll find a file with the same name as your solution, with a .SUO extension. Delete that file, and you'll lose all your user options, including saved breakpoints, but you'll also lose your selected window positions. Next time you start up, all the windows are back to the Visual Studio defaults again. If all else fails, this puts you back to square one.

[Register for VSLive! San Francisco Now](#)

Contributed by Laura J. Barker, who is presenting at VSLive!
Reach Laura at www.empowered.com

- 35.** When using ADO .NET - think "Read" the data before you can "Set" the data. ADO .NET makes use of data readers and data sets and it's now up to the developer to know when to use each one. For simple reads use the data reader, for updates use commands. If you have a need for disconnected recordsets use a data set. When data sets are created, "behind the scenes" they use a data reader to create the actual data set, but datasets are big objects. This makes the data set more expensive and slower than just the data reader alone. You wouldn't want to use a data set to populate a drop down list that is read only and won't change during an application. This is one reason why in ASP.NET solutions in most cases Data Readers are the proper method of reading data. So remember, to help with reducing cost of data retrieval and assist in reducing the time needed to get the data, Always..."Read" before you "Set".
- 36.** When starting with ASP.NET, don't automatically use a server side control for everything on your page. It's not necessary. Remember, server side controls make a round trip to the server during every POST. This can be expensive if you use them for items such as Labels or Headers which don't need that level of programmatic control. Instead, make use of standard HTML, which aren't created as server side controls.
- 37.** When working on the performance of the application understand the performance needs for a given solution. Don't simply tune an application to get the best possible speed for a process. Think to yourself, will a user notice that a task takes .5 seconds to complete or .4 seconds when performance tuned? Will taking the extra time to tune an application for the extra performance gain be worth it? And the performance tuned code, how maintainable will it be? Understand, sometimes you will need to give up pure performance for a few users for the sake of scalability to 1000's of users.

Contributed by Michael Amundsen, who is presenting at VSLive!
Reach Michael at www.amundsen.com

38. IMPLEMENTING PRE-JIT SERVICE FOR YOUR ASP.NET PAGES

To help speed the just-in-time (JIT) compiling of your ASP.NET Web applications, place the following directive at the top of every page in your Web:

```
<%@ page debug="false" %>
```

When you add the debug="false" directive to the top of the pages, the ASP.NET will 'JIT' all the pages in the directory at the same time. For example, create a new Web and drop ten pages into the Web. The first time you visit the default.aspx page, all ten pages will be JIT-ed and ready for use.

[Register for VSLive! San Francisco Now](#)

There are some exceptions to keep in mind. First, if the compiler encounters an error on any page, the pre-JIT process stops. If the error was on any page other than the one requested, no compiler error will be thrown, either. Also, the pre-JIT process happens by directory. If you have a sub-directory of files, these files will not be JIT-ed until someone requests one of the pages within that sub-directory.

39. GET A QUICK VIEW OF THE GLOBAL ASSEMBLY CACHE

If you want to see what assemblies are stored in the global assembly cache (GAC), all you need to do is point your file explorer to the C:\WINNT\assembly folder. All machines that have the .NET runtimes installed will display the contents of this folder in a custom view. The "detailed" view shows the assembly name, type, version, culture setting, and even the public key token. You can also right click on each assembly to get a detailed property listing or even delete the assembly from the global cache.

40. TAME THE VISUAL STUDIO .NET HTML REFORMATTER

If you get frustrated when the Visual Studio .NET editor reformats your carefully crafted HTML and XML markup, you can take control again by adjusting the rules under which the editor reformats your text. To do this, select **TOOLS - OPTIONS** to bring up the options dialog. Then select **TEXT EDITOR - HTML/XML** from the tree control on the left of the dialog. You'll find sections for **GENERAL**, **TABS**, **FORMAT**, **HTML**, and **XML** options that will allow you to modify the reformatting rules or even turn them off completely.

Contributed by Paul D. Sheriff, who is presenting at VSLive!
Reach Paul at www.dotnetjumpstart.net

41. The StringBuilder Class

Just about every business application you create will eventually require that you work with strings in some manner. String manipulation is a very common activity in most applications, but is also one of the most expensive in terms of processor power and memory. Creating, adding to, or deleting from strings takes a lot of processor time, and takes a lot of memory. If you create a variable of type `String`, you are actually creating a `String` object; all base data types in .NET inherit from the base `Object` class, so all data types are actually objects. The `String` object cannot be changed, despite the fact that it can appear to be changed just by assigning a new string to a `String` variable. For example, there is a method of the `String` class called `Concat` that allows you to concatenate strings (or objects).

However, because the `String` object is immutable, a new string object is actually created in memory and reference to that new object is returned from the `Concat` method.

[Register for VSLive! San Francisco Now](#)

The above process consumes a significant amount of processor cycles and memory, but you do have an alternative: the `StringBuilder` class. If you are performing a few simple assignments or concatenations, don't worry about the overhead. However, if you are building a string inside a loop, the `StringBuilder` is going to give you much better performance with lower overhead.

The `StringBuilder` class is found in `System.Text`. `StringBuilder` allows you to change a string without creating a new `String` object. Therefore, if you want to append, insert, remove, or replace characters in a string, the `StringBuilder` may be a better choice if you have a significant amount of manipulation to perform.

The `System.Text` namespace also contains classes for encoding characters into bytes and decoding bytes into characters. There are also encoders and decoders for converting to and from ASCII and Unicode.

System.Text Example

In this example, you create a `System.Text.StringBuilder` object and set it to a string using the constructor. Next, you display the 7th character in the string into a `Label` control on the page. You can then invoke the `Replace` method to change one string with another string. Finally you can insert a new value into any location within the string. In this case you will add this new value to the end of the string.

```
Private Sub StringBuilderExample()  
    Dim sb As New _  
        System.Text.StringBuilder(".NET is cool")  
  
    ' Display the 7th character  
    lblMsg.Text = sb.Chars(6)  
  
    ' Replace .NET with Microsoft.NET  
    sb.Replace(".NET", "Microsoft .NET")  
    lblMsg.Text = sb.ToString()  
  
    sb.Insert(sb.Length, " and fun.")  
    lblMsg.Text = sb.ToString()  
  
End Sub
```

What makes this more efficient than using a normal `String` object, is that when you create a new `StringBuilder` object it automatically adds on some additional room for growth. In this way you can always work with the same memory space until you outgrow the initial space. In fact, you can pass the initial space to the constructor to specify how much space to allocate when a new object is built.

42. Handling Option Strict

In VB .NET, Option Strict is Off by default-this is a poor decision, in our eyes, since it allows the same sort of code you might have written in VB 6, including potentially dangerous type conversions. With Option Strict turned on, you can catch these errors at compile time, instead of at run-time-basically, there's less to worry about at run-time, with the trade-off being more to worry about at coding time. It also forces you into a better understanding of the objects you are working with, and it enables code to be reused in a project where Option Strict is on. As it is, you must open the project properties for each new project you create, in order to turn on Option Strict or, you can manually add it to the top of each file, which is certainly an onerous task. You can set Option Strict on in VB .NET, for all new projects you create, but it's a non-trivial task. To do this, find the folder containing all the project templates. For example, this might be a folder like H:\Program Files\Microsoft Visual Studio .NET\Vb7\VBWizards. Starting in that folder, use Windows Explorer to search for *.vbproj. Load all the files named *.vbproj into a text editor, and modify the XML at the top of the file so that it looks something like this (what we're showing here is just the beginning of the vbproj file. Leave all the existing tags alone-simply insert the OptionStrict attribute within the Settings element):

```
<VisualStudioProject>
  <VisualBasic>
    <Build>
      <Settings
        OptionStrict = "On"
        -- more attributes and elements follow --
```

Save each file, and the next time you create a project based on one of these templates, Option Strict will be on, by default. Most likely, you want Option Strict on for development in VB .NET, and you'll forget to turn it on manually.

43. Using String.Format()

Instead of concatenating strings together all the time when you need to insert values into the middle of a string, use String.Format instead.

```
strSQL = "SELECT * FROM Products "
strSQL &= " WHERE CategoryID = {0}"
strSQL &= " AND SupplierID = {1}"

strSQL = String.Format(strSQL, 5, 12)
```

When this string comes back the 5 will be placed where the {0} is located in the string and the 12 will be placed where the {1} is located.

```
"SELECT * FROM Products WHERE CategoryID = 5 AND SupplierID = 12"
```

44. Application Class

If you wish to retrieve the current folder of where the EXE is located, just like App.Path in VB 6, you will need to use the Application object's Executable Path. This will return the full path, which by default is in the \Bin folder under your project folder. For example, if you want to get at a file called Contact.XML file that is in your project folder, you will need to remove the \bin folder from this path. When you are using the "Debug"

Build mode you can use the DEBUG compile constant to remove that line. When you compile for release the EXE and this file would be in the same folder, so you do not want to look for a \Bin folder.

```
Public Function GetFileNameAndPath() As String
    Dim strFile As String

    ' Executable Path returns everything including
    ' the name of the .EXE
    strFile = Application.ExecutablePath
    #If DEBUG Then
        ' Look for the \bin\ and get everything
        ' up to that location
        strFile = strFile.Substring(0, strFile.LastIndexOf("\bin\")) &
        "\"
    #End If

    ' Add on the file name
    strFile &= "Contact.xml"

    Return strFile
End Function
```

45. Environment Class

In VB 6 you had to use a combination of built-in VB commands and windows API calls to get things like the command line, the current directory, the machine name, the OS version, etc. Now there is a class called Environment that has properties that will return all this information for you. Below is a sample of using this system class.

```
Dim oEnv As System.Environment

With oEnv
    txtCommandLine.Text = .CommandLine()
    txtCurDir.Text = .CurrentDirectory()
    txtMachineName.Text = .MachineName()
    txtStack.Text = .StackTrace()
    txtOS.Text = .OSVersion.Platform
    txtSysDir.Text = .SystemDirectory()
```

[Register for VSLive! San Francisco Now](#)

```
txtTicks.Text = .TickCount()  
txtUserDomain.Text = .UserDomainName()  
txtUserName.Text = .UserName()  
txtVersion.Text = .Version.ToString  
End With
```

Contributed by Peter Vogel who is presenting at VSLive!

- 46.** In order to let you access a Web Service asynchronously, the proxy classes for the service include a Begin* and End* version of each method in the service. For instance, if the service has a method called CreditCheck, the proxy class will have methods called BeginCreditCheck and EndCreditCheck. Using BeginCreditCheck will call the underlying CreditCheck method asynchronously and will return a IAsyncResult object. You can pass this IAsyncResult object to the EndCreditCheck method to wait for the underlying method to finish and to catch the result. The Begin* version of the method will have two extra parameters that you can pass Nothing to. As an example, the following code calls the CreditCheck method of a service. The CreditCheck method normally requires one parameter and returns a single integer result:

```
Dim Ias As IAsyncResult Ias = myServ.BeginCreditCheck(creditCheckParm, Nothing,  
Nothing)...other processing... intResult = myServ.EndCreditCheck(Ias)
```

- 47.** When calling a Web Service method asynchronously (using the Begin* and End* versions of any method call) you can pass the address of a callback function as the second parameter to the Begin version of the method. When the underlying method is finished, your routine will automatically be called and be passed an IAsyncResult object. You can use that IAsyncResult object with the End* version of the method to retrieve the result returned by the method. For instance, to call a method called CreditCheck (which accepts one parameter and returns an integer result) you could use code like this to invoke a routine called ProcessCreditCheck when the CreditCheck method is done:

```
myServ.BeginCreditCheck(creditCheckParm, addressOf  
ProcessCreditCheck,  
Nothing)  
...other processing...  
Sub ProcessCreditCheck(ByVal Ias as IAsyncResult)  
intResult =  
myServ.EndCreditCheck(Ias)  
End Sub
```

Register for VSLive! San Francisco Now

Contributed by Robert Patton, who is presenting at VSLive!
Reach Robert at Robert.Patton@purchasingfirst.com

- 48. Cookieless sessions are only a configuration option away! In ASP you had to rely on very cumbersome programming, write your own ISAPI filter or use the IIS resource kit Cookie Munger to maintain a session GUID without a cookie. In ASP.NET, all you have to do is change the cookieless parameter in config.Web and Voila, the GUID is embedded in the URL for you.
- 49. Use the language (VB, C#, Jscript, etc.) with which you are comfortable. All the .NET languages use the same Common Language Runtime and, if you write logically equivalent code, the performance will be the same no matter what the language.
- 50. Be prepared to rewrite all your COM components. Sure you can use them in .NET, but it is much cleaner and you will get far better performance if you migrate them to .NET. In addition, be wary of upgrading COM components using the Upgrade Wizard. The Wizard can help you see what parts of your application need the most changes and perform some of them for you, but it will not help you develop good .NET applications. It makes the minimal amount of changes necessary, and it will not help you take advantage of the innovations in .NET that are the most important. For instance, if you have a VB6 DLL used for data access, the Upgrade Wizard will continue to have the application utilize ADO instead of ADO .NET.

Contributed by Rockford Lhotka, who is presenting at VSLive!
Reach Rocky at www.lhotka.net

- 51. Use only OLE Automation types and interfaces in your COM components and avoid using Variant data types in your public interfaces. This will make COM interop work as smoothly as possible.
- 52. Be patient. When moving to VB .NET there is a period of learning where things can be very frustrating - simple things seem hard to do, etc. This is temporary - some of it is due to moving from VB 6 to VB .NET, some of it is due to moving from the Windows platform to the .NET platform. All of it will fade as we learn the new (and often improved) ways of doing the tasks.

[Register for VSLive! San Francisco Now](#)

Contributed by Roger Jennings, columnist for .NET Insight and contributing editor for Visual Studio Magazine. Reach Roger at Roger_Jennings@compuserve.com

Tips for Designing Data Components with .NET in Mind:

- 53.** Add the `.Value` property to every `Recordset.Fields(...)` statement in your project. VB .NET doesn't support the default property of late-bound objects. (ADODB objects are late-bound by the COM Interop helper.)
- 54.** Wrap all `Recordset.Fields(...).Value` statements with the appropriate type conversion operator (`CStr()`, `CInt()`, `CBool()`, and so on). Visual Basic's Evil Type Coercion (ETC) "feature" is gone for good with VB .NET's strict type checking. While you're at it, do the same in code that concatenates multiple variable types with the `&` operator.
- 55.** Use read-only, forward-only (firehose) cursors wherever you can. ADO .NET's forward-only cursor lets you take full advantage of lightweight `DataReader` objects and the high-performance SQL Server .NET data provider (formerly "managed provider") to maximize scalability. `SqlConnection` objects use SQL Server's native Tabular Data Stream (TDS) wire format and don't add the overhead of OLEDB and ADODB layers.
- 56.** If you need scrollable `Recordsets`, use static (client-side) cursors when feasible. ADO .NET `SqlConnection` and `ADOConnection` objects don't support server-side cursors and there's no equivalent to the `ADODB.Connection.CursorLocation` property.
- 57.** Design your classes' public functions to complete their tasks with a single method call. For example, several of my original COM components return XML or XHTML, depending on the value of a *DocType* argument. Instead of returning XML and then instantiating another XHTML transformation component, the public *GetDocument* function calls the class's private *TransformXHTML* function when XHTML is required.
- 58.** Pass your components through the Visual Basic Upgrade Wizard early and often. Fix compilation errors and then deal with issues. (Some issues, such as inability to determine the default property for `Collection.Item(n)`, won't go away, at least in VB .NET beta 2.) Make minor code changes to minimize the number of "UPGRADE NOTE:" comments. For example, replace constants with conventional variables and assign their values in the `Class.Initialize` event handler, because VB .NET doesn't support the `Const` reserved word. (You can't get rid of messages that report the `IsDBNull()` function replaces `IsNull()`.)

[Register for VSLive! San Francisco Now](#)

Contributed by Todd Kimble, who is presenting at VSLive!

Reach Todd at www.claritycon.com

59. ASP.NET

Keep application specific settings in the Web.config file instead of hardcoding in the application. This file is cached by the environment during run time for fast access, and it can easily be modified by administrators as the system moves between test and product environments.

60. Windows Forms

Use the Assembly object in the System.Reflection namespace to create Windows programs that can automatically download updates from your Web server. The downloaded portions of the application are stored in the local Assembly Cache, and new versions of the application are downloaded only when the assemblies on the Web server have been updated.

61. General

In VB .NET many of the utility functions such as "Mid\$", "Cint" and "Now" are still available but most of those functions have equivalent methods in the .NET framework. Check out the System.Convert, System.DateTime and System.Environment libraries for some of the new versions of your favorite functions. Use the new versions of these functions for the applications you write now to avoid problems in future versions of VS .NET as Microsoft retires the older syntax of prior VB versions.

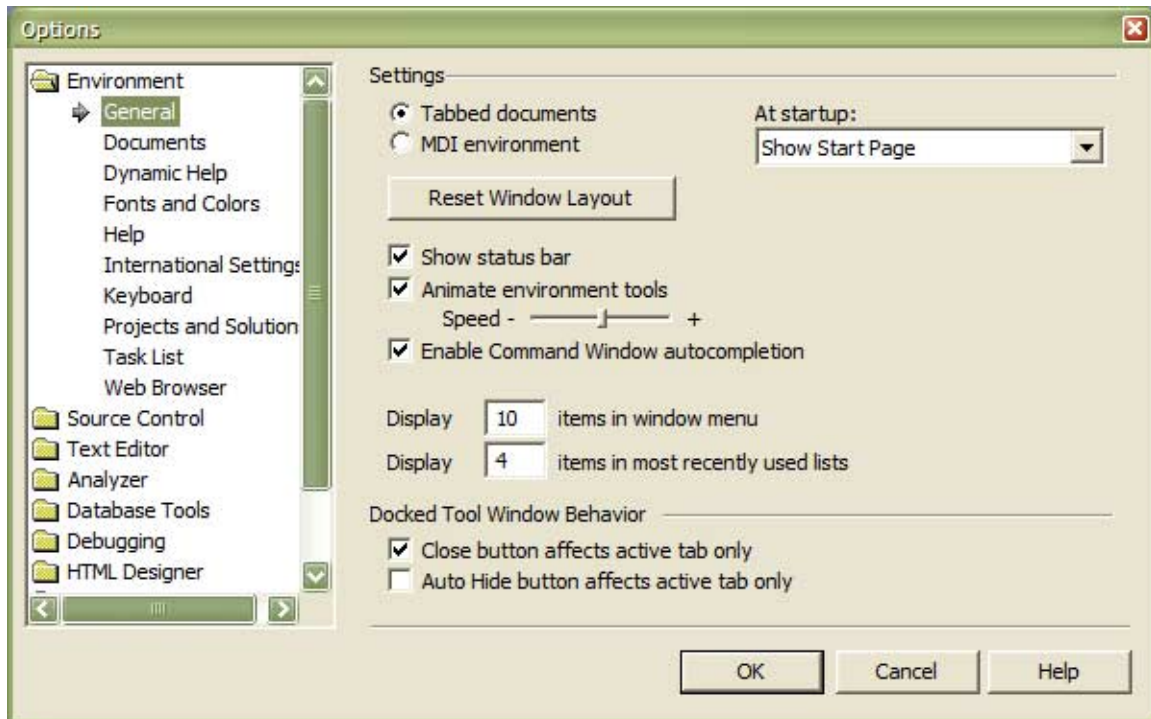
Contributed by Yasser Shohoud, who is presenting at VSLive!

Reach Yasser at www.LearnXmlWS.com

62. Resetting Window Layout

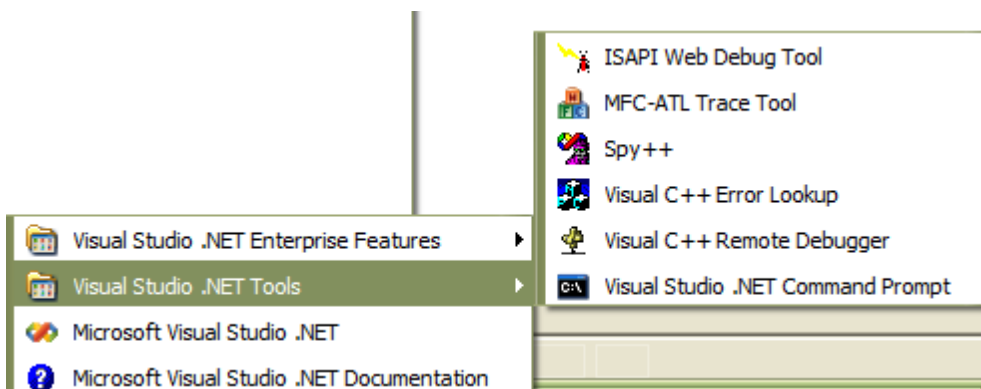
One of the first things you'll encounter when using VS .NET is the large number of Windows that you can dock, auto-hide and close. After some playing around you are likely to want to put things just the way they were. To put all Windows back where they belong, go to the Tools menu and choose Options. Choose Environment, General from the left navigation tree and click on Reset Window Layout. When you click OK to close the Options dialog, all Visual Studio Windows will be back in the location and state (docked, auto-hide etc.) where they originally were.

[Register for VSLive! San Francisco Now](#)



63. Using .NET Framework SDK Tools

Beyond Visual Studio .NET, the .NET Framework SDK includes a variety of useful tools many of which are command based. For example, you could use `xsd.exe` to create classes from XML Schemas and vice versa. To use these command-based tools, you'll need to use a command prompt with the right environment settings. At the very least, you'll need the framework SDK in your path so you can run these tools without typing the fully executable path. Fortunately Visual Studio .NET installs a shortcut to a command prompt that has all the right environment settings. You can find this shortcut in the Visual Studio .NET Tools Folder as shown below.



[Register for VSLive! San Francisco Now](#)

64. How to Master Web Services Development

Web services offer a powerful programming model where applications can easily integrate with one another over the Internet or an intranet. There are many standards and technologies that make up Web services and new technologies and standards are invented and proposed almost every day. So how does one master Web services development in the face of all this change? Learn XML. Web services are built on core XML standards such as XML Namespaces and XML Schemas. These standards are less volatile than other things in the Web services world and are the basis for all new Web services technologies and standards.

Many people say you can build Web services without ever dealing with XML. This statement is true only for Hello World Web services, building business Web services requires a good understanding of core XML technologies including XML Namespaces and Schemas. Learn XML now, it will help you master Web services development and give you the foundation for the many changes to software development that will happen over the next two years.

65. How to disable automatic formatting

Most Web developers don't like the idea of an IDE formatting their HTML. While VS .NET by default formats your HTML to whatever it thinks appropriate, you can turn off this behavior and save yourself some frustration.

From the Tools menu choose Options. Choose Text Editor, HTML/XML, and Format from the left navigation tree. Uncheck the two boxes under Apply Automatic Formatting then click OK to save the settings.

