What's New in Visual Basic.NET

Billy Hollis
General Manager, Oakwood Systems Group, Inc.

Introduction

Visual Basic (VB) developers need to get ready for the biggest transition they've ever had to make. The next VB (probably called Visual Basic.NET) is designed to make web development as drag-and-drop easy as VB1 made Windows development. VB will be integrated into a completely new architecture called Microsoft.NET, which is Microsoft's next generation platform for Windows and Internet software development.

It's hard to overstate the importance of Microsoft.NET. In his keynote speech at the Professional Developer's Conference (PDC) in Orlando this year, Bill Gates stated that a transition like this only comes along once every five or six years. The last comparable shifts were the migration from 16-bit to 32-bit development, and to COM technologies.

The result is huge changes. On the plus side, VB gets wonderful new capabilities for web development, object-orientation, error handling, threading control, and much more. On the downside, numerous syntax incompatibilities will cause the migration from older versions to be complex and painful.

VB topics to be discussed include new object-oriented capabilities (full inheritance, overloading, parameterized construction of class instances, and shared members of classes), changes in data types, changes in argument passing and calling conventions, and changes in syntax used to declare and initialize variables. Syntax examples and demonstrations will be included whenever possible.

On the Visual Studio.NET side, this session will briefly cover Web Forms, Web Services, changes to the Visual Studio Integrated Development Environment (IDE), and an overview of the new Common Language Runtime, which is the architectural reason for many of the coming changes.

Attention will also be focused on what you can do today to prepare for future capabilities. For example, designing for easy migration to Web Forms will be discussed, and tips will be included on making current code easy to convert by avoiding syntax that will no longer be supported.

Billy Hollis has been developing software for over twenty years. He has written for many technical publications and is a frequent speaker at conferences, including Comdex and the Visual Basic Insiders Technical Summit. He is also the author of the book "Visual Basic 6: Design, Specification, and Objects".

Billy is General Manager of the Nashville branch of Oakwood Systems Group, a consulting firm based in St. Louis which specializes in software development with Microsoft technologies. He is also MSDN Regional Director of Developer Relations in Tennessee for Microsoft.

The Microsoft.NET Framework – An Overview

Starting in late 1995, Microsoft refocused on marrying their Windows platform to the Internet. They have certainly succeeded in making it a serious Internet platform.

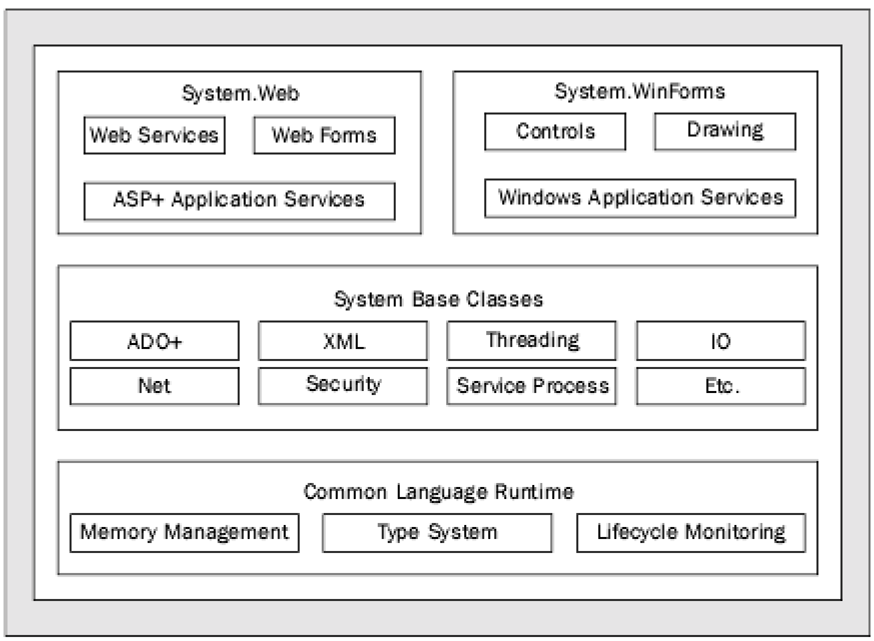
However, Microsoft had to make some serious compromises to quickly produce Internet-based tools and technologies. In particular, Active Server Pages (ASP) have always been viewed as a bit clumsy. After all, writing reams of interpreted script is a real step backwards from structured and object-oriented development. Debugging and maintaining such unstructured code is also a headache.

Other languages have been used in Internet applications, but only as components which worked through ASP. Presently, Microsoft tools lack the level of integration and ease-of-use for web development that would be ideal. Except for WebClasses in VB, there hasn't even been an attempt to place a web interface on traditional languages.

Microsoft.NET promises to change this. First and foremost, it is a framework that provides the richest level of integration among presentation, component, and data technologies ever seen on a Microsoft, or perhaps any, platform. Secondly, the entire architecture has been created with the Internet in mind.

A Common Substrate for all Development

The major components of the Microsoft.NET framework are shown in the following diagram:



The framework starts all the way down at the memory management and component loading level, and goes all the way up to multiple ways of rendering user interfaces. In between, there are layers that provide just about any system-level capability that a developer would need.

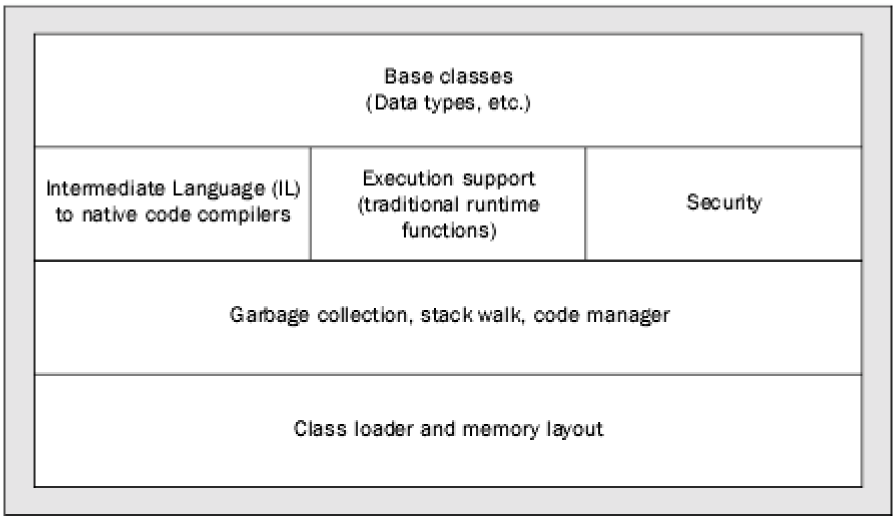
At the base is the Common Language Runtime (CLR). Many of the upcoming changes in VB are driven by the technologies in this layer. It includes, for example, a common system of data types. These common types, plus a standard interface convention, make cross-language inheritance possible. All these changes are discussed in more detail below.

The middle layer includes the next generation of standard system services such as ADO. Other sessions at the conference discuss many of the changes taking place in this layer.

The top layer includes user and program interfaces. WinForms is a new and more advanced way to make standard Win32 screens. Web Forms provides a new web-based user interface. And Web Services provides a mechanism for programs to communicate over the Internet, using the Simple Object Access Protocol (SOAP). This layer also drives some important changes in Visual Basic.NET.

The Common Language Runtime

We're all familiar with runtimes. They go back to DOS languages. But the CLR is as advanced over traditional runtimes as a machine gun is over a musket. Here's a quick diagrammatic summary of the major pieces of the CLR:



That small part in the middle called "Execution Support" contains most of the capabilities normally associated with a language runtime. The rest is new.

Impact on Visual Basic.NET

The design of the CLR is based on a number of important goals. The ones that have the most impact on Visual Basic.NET include:

- Simpler, safer deployment of applications
- Scalability
- Multiple language integration

Simpler, Safer Deployment of Applications

Other sessions have covered the fact that the .NET framework makes registration, GUIDs, and the like unnecessary. Applications produced in the .NET framework can be designed to install with a simple XCOPY. That's right – just copy the files onto the disk and run the application. We haven't seen this since the days of DOS (and some of us really miss it).

This has implications that might not be apparent at first. For example, running a program off of a CD (without installation) was not feasible in VB after version 3. That capability will reappear with Visual Basic.NET.

Scalability

Since most of the system-level execution functions are concentrated in the CLR, they can be optimized and architected to allow a wide range of scalability for applications produced in the Microsoft.NET framework. As with most of the other advantages of the CLR, this one comes to all applications with little or no effort.

Memory and process management is one area where scalability can be built in. The memory management in the CLR is self-configuring and tunes itself automatically. Garbage collection is highly optimized, which helps programs run faster and thus scale up better. Threads are also pooled for reuse.

The effect for VB developers is to level the playing field. The performance and scalability differences with other languages become smaller. It becomes unnecessary in most cases to look to other languages when performance is an issue.

Multiple Language Integration and Support

The most ambitious aspect of the CLR is that it is designed to support multiple languages and allow unprecedented levels of integration among those languages. By enforcing a common type system and by having complete control over interface calls, the CLR allows languages to work together more transparently than ever before.

Previously, one language could instantiate and use components written in another language by using COM. Sometimes calling conventions were difficult to manage, especially when VB was involved, but it could generally be made to work. But subclassing a component written in a different language required a sophisticated wrapper, and only advanced developers did such work.

The CLR makes it straightforward to use one language to subclass another. This means that a class written in VB can inherit from a base class written in C++, or in COBOL for that matter. And we're talking full implementation inheritance, with no problems requiring recompilation when the base class changes.

A Common Type System

A key piece of functionality that enables multiple language support is a common type system, in which all commonly used data types are actually implemented as objects. Coercion among types can now be done at a lower level for more consistency between languages. And, since all languages are using the same library of types, calling one language from another doesn't require type conversion or weird calling conventions.

This results in the need for some readjustment, particularly for VB developers. (The differences in VB are summarized below.) But the adjustment is worth it to bring VB in line with everything else. And, as a by-product, other languages get the same superb support for strings that VB has always had.

Namespaces

One of the most important concepts in the .NET framework, which VB developers must understand, is namespaces. They help organize object libraries and hierarchies, simplify object references, prevent ambiguity when referring to objects, and control the scope of object identifiers.

A full discussion of namespaces is beyond the scope of this session. For now, it is useful to know that class libraries can be referenced with an `Imports` statement, and that this can be thought of as similar in concept to checking a box in the "References" dialog in VB6. For example, a typical VB form module in the new version might have the following lines at the beginning:

```
Imports System.Winforms
Imports MyDebug = System.Diagnostics.Debug
```

The first line simply makes all of the standard form properties and methods available to the code in the form module. The second line illustrates the use of an alias. A branch of the object hierarchy can thus receive its own identifier, which is only valid in that code module. Instead of referring to the Debug object, the code in this module would refer to the MyDebug object.

The First Step – Visual Studio.NET

The first technology to be released in the Microsoft.NET framework will be the next generation of Visual Studio, which has been tagged Visual Studio.NET. Here are some of the confirmed changes and new features.

Common IDE for All Languages

Microsoft has gradually been merging the Integrated Development Environment for all their languages into one code base. In Visual Studio 6, VB was the last major holdout. That process is now complete, and the exact same IDE is to be used for all languages in the Visual Studio.NET suite. Vendors of third-party languages can also plug into the IDE.

Management of Multiple Language Projects

Since any number of languages can now be used in a project, the Visual Studio IDE will now look at projects in terms of all the modules being used, no matter what language they are in. The project explorer is the same no matter what combination of languages is used.

Absence of Visual Interdev and J++

Visual Studio.NET does not have a separate piece identified as Visual Interdev. In effect, the functions of Visual Interdev have migrated into the IDE as a whole. There is an HTML editor, for example, which works across the whole IDE, and the new project explorer bears a strong resemblance to the resource window in Interdev. Also, Web Forms have basically taken the place of the drag-and-drop visual designer in Interdev, and they work with any language.

J++ is gone for a different reason. Microsoft's continuing legal troubles with Sun have frozen Java-related efforts at Microsoft. The current J++ will be supported, but it cannot be enhanced and placed in the new Visual Studio at this point. In a sense, the new language C# (which is extensively covered in other sessions) takes its place, but it's a sure bet that a real Java implementation from some third party will be available at or near Visual Studio.NET's release.

Summary of Visual Basic Language Changes

VB gets the most extensive changes of any existing language in the suite. These changes pull VB in line with other languages in terms of data types, calling conventions, error handling and, most importantly, object-orientation.

Most new features, especially object-oriented ones, are courtesy of the CLR. VB basically piggybacks on the stuff that was going to be implemented anyway for C++, C#, and all the rest.

Lots of Possible Incompatibilities

These are the biggest changes ever in VB – bigger even than the jump from VB3 to VB4. This introduces significant potential for incompatibilities. Some are known and are summarized below. Others will probably crop up soon.

Microsoft intends to supply a conversion tool that will assist in porting VB6 projects to .NET, but it will not do everything required. There will be some areas where the conversion tool merely places a note that indicates something needs to be done. And there are sure to be areas where it fails to realize that a change is needed.

Object Features

Many of us have been clamoring for VB to support full object-oriented development since classes were introduced in version 4. With Visual Studio.NET, we are granted our wishes.

Full Inheritance

Finally, VB gets full inheritance. The syntax is simple. A class notes the base class using the `INHERITS` keyword. The base class is referred to inside the subclass' code with a reference to `MyBase`.

Full overloading of properties and methods is supported, so that the subclass can substitute logic for a given property or method of the base class. The overloaded logic can be completely separate from the base class, or can merely wrap the base class property or method with some additional logic.

Parameterized Constructors

Another useful object-oriented capability found in most true object languages is parameterized constructors. That means that, at instantiation of an object, parameters can be passed in to affect the object's behavior immediately. Previously in VB, any such changes in behavior had to be done with property settings or methods after the instantiation was complete, which was sometimes clumsy and limiting.

In Visual Basic.NET, the constructor is called `Sub New`. This routine can have parameters specified just like any other sub.

Shared Members

Proper object designs usually isolate object instances from one another completely. However, it is occasionally helpful to have information shared by all active instances in a class. In the new VB, shared members (properties or methods) can be constructed to do this. The concept is identical to what are called "static members" in C++.

The `Shared` keyword is used to create shared members. For example, to share a string data member called `CommonName`, this line of code would be used in the declarations section of the class:

```
Public Shared CommonName As String
```

Support for Web Forms and WinForms

All the languages in the new Visual Studio will support Web Forms and WinForms, but the changes will look the largest to VB developers. WinForms is descended from the Windows Foundation Classes, a library which was originally written for J++. It is likely that WinForms will replace the current VB forms engine, but presumably Microsoft will make this transition reasonably transparent. The addition of Web Forms will have more impact, increasing the practicality of VB as a web development tool by a huge step. It should now be practical to develop a web application completely within VB.

Web Forms depend on a new type of component, called a server-side control. This component contains a representation of a web control on the server with the intelligence to render the control in the web interface using HTML 3.2. It also handles its own HTML responses, and incorporates the returned data.

Web Forms are designed to accommodate a broad range of browsers, from the most sophisticated down to simple browsers on wireless mobile devices. Server-side controls need significant intelligence to render HTML for all these different levels of browsers, and to coordinate events with the client on which the page is running. A wide variety of controls is expected to ship with Visual Studio.NET, bringing web-based interfaces much closer to Win32 interfaces. Third parties are expected to add even more options for server-side controls. Tools vendors will have a brand-new market to attack.

As a side effect, earlier attempts to make up a web interface, including WebClasses and DHTML forms, will probably not be supported in Visual Basic.NET. They won't be missed if Web Forms lives up to expectations.

Structured Exception Handling

Java and C++ developers have grown used to far better error handling capability than exists in VB. The `On Error` constructs used in VB have not improved significantly since QuickBASIC (the DOS ancestor of VB). Catching an error in a specific block of code was particularly clumsy, with the need to turn various kinds of error handling on and off at specific places. One of the side effects was that `On Error Resume Next` was often over-used and abused.

That is going to change. Now in VB, the following syntax can be used to handle errors and exceptions:

```
Try

    ' Some typical processing logic
    rsRecordset.Update

Catch

    ' This code runs when an error occurs in the code above
    LogError ("Unable to update the recordset")

Finally

    ' This code always runs, either after the Try code if there
    ' was no error, or after the Catch code if there was
    rsRecordset.MoveNext

End Try
```

If need be, the logic in the `Catch` portion can use an expression named `Throw` to force an error. This is much like `Err.Raise` in earlier versions of VB.

`On Error` syntax is still going to be supported and the `Err` object will still be available.

No More Default Methods or Properties

Let's examine an example of typical code in VB6 or earlier. In this example, assume `strName` is a string variable, and `txtName` is a text box.

```
strName = txtName    ' works in VB6 and earlier
```

This code works because the text box control has a default property, namely the text it holds. The above line is actually compiled as:

```
strName = txtName.Text
```

The first form will no longer work in the next version of VB because default properties and methods are no longer supported. The second form will be required.

Good coding practices have recommended against the first syntax example for years. It is not easy to read and can be misinterpreted. But one related syntax form has been used rather commonly when working with data. It's not at all unusual to see lines like this:

```
strName = rsRecordSet("Name")    ' works in VB6 and earlier
```

This works because `Fields` is the default property of the `Recordset` object. This code actually means:

```
strName = rsRecordSet.Fields("Name")  ' will work everywhere
                                        ' including VB.NET
```

Some early sources claimed that the first form would not work in the next VB. However, it appears to work with the tech preview released at PDC. Presumably, this type of default member (in which parameters tip off the compiler that the default member is intended rather than the object reference) will be available in Visual Basic.NET, but I'd recommend that this not be assumed until later beta versions are available.

SET and LET No Longer Supported

Why take away default properties and methods? Having them made it necessary to use the `Set` keyword in VB for all object references. That is, to set an object reference to a `Recordset`, current and previous versions require syntax like this:

```
Set rsNewRecordSet = rsOldRecordSet    ' VB6 and earlier style
```

Without the `Set`, the code is not assigning object references – it is trying to assign a value to a default property. So, to keep things straight for the compiler, `Set` was used for objects, and `Let` statements (where the `Let` has long since been optional) were used for assigning values.

Now that default properties no longer exist, there is no longer any need for `Set` or `Let`. Both are being dropped from the next VB. The operation above will merely be coded as:

```
rsNewRecordSet = rsOldRecordSet       ' new VB.NET style
```

Different Syntax for Class Properties

In VB4, VB5, and VB6, up to three separate property procedures (`Let`, `Get`, and `Set`) are used for each property, although it was only required to use one. In Visual Basic.NET, `Let` property procedures are eliminated because there's no need to differentiate between object references and common variables. Set and Get procedures are then tied together with some new syntax, which resembles equivalent syntax in C#. Here's an example:

```
Private mintIntegerProperty as Integer

Public Property IntegerProperty As Integer

    Get
        IntegerProperty = mintIntegerProperty
    End Get

    Set
        mintIntegerProperty = IntegerProperty
    End Set

End Property
```

This is a good deal cleaner than the equivalent VB6 (and earlier) syntax. It also removes a lot of pitfalls, such as changing the data type for a Get, but forgetting to do so for the equivalent Let.

With this syntax, you can't make a property read-only by merely leaving out the Let and Set procedures. You must explicitly declare it with the ReadOnly keyword at the beginning of the property declaration. Then the Set block can be left out. There is also a WriteOnly keyword to make a property write-only, in which case the Get block is left out.

Required Parentheses on Methods, Functions, and Subroutines

Many of us (including the author) have gotten sloppy about many calling conventions in VB. For example, here's some commonly used code:

```
MsgBox "Hello, World"
```

and here's another common construct:

```
Dim sDate As String
sDate = Date
```

Neither of these will work in the new VB because the compiler now requires the developer to always include parentheses, even for null argument lists. The equivalent working versions of the above examples in Visual Basic.NET are:

```
MsgBox ("Hello, World")
```

and:

```
Dim sDate As String
sDate = Date()
```

Note that you can't even insert the second example into VB6 code. The development editor will remove the parentheses as soon as you leave the line. But they are required for Visual Basic.NET.

Parameters are ByVal by Default

In VB6 and before, parameters in an argument list which were not declared ByRef or ByVal were assigned a default based on what type of parameter was involved. Intrinsic data types, such as integer, string, Boolean, etc., were defaulted to ByRef. Object references and other non-intrinsic types were defaulted to ByVal.

If a ByRef parameter is changed in the called routine, the changes are reflected in the calling code. Sometimes this is desirable, but it is generally considered poor programming practice, especially when the parameters become ByRef by default. It can lead to subtle and hard-to-find bugs.

Visual Basic.NET removes the problem. All parameters are ByVal unless explicitly declared otherwise.

Changes in Data Types

The data types for whole numbers are being revamped in Visual Basic.NET. Here is a comparison table:

Old type	***New type***	***Size***
Integer	Short	16 bits
Long	Integer	32 bits
(N/A)	Long	64 bits

This has a lot of potential to cause confusion but, again, is being done to bring VB in line with other languages. The biggest issue is that this can break calls to routines in DLLs or the Windows API that are expecting certain lengths. It can also affect code which does bit-wise arithmetic.

The Currency data type will no longer be supported, and should be replaced by the Decimal data type (which is 96 bits – enough for financial calculations).

There's another new data type for single characters called Char. Note, however, that it's two bytes long because it holds Unicode character values.

The Variant type will no longer be available in Visual Basic.NET. The object type can be used in its place, since even intrinsic data types such as integer and string are considered objects in the .NET framework.

New Declaration Capabilities

VB's syntax for declaring variables has always been a bit quirky, and most of the quirks are being removed. For starters, it is now possible in Visual Basic.NET to declare a new variable and assign an initial value at the same time. Here's a sample:

```
Dim intHoursAvailable As Integer = 10
Dim intMinutesAvailable As Integer = intHoursAvailable * 60
```

Another old VB quirk is allowing mixed declarations on a single line. This line is perfectly legal in VB6:

```
Dim strName as String, intAge As Integer
```

It will not work in Visual Basic.NET, however. The only way multiple variables can be declared on one line is if they are all the same type, and the declaration is made like this:

```
Dim strFirstName, strLastName As String
```

Notice that this line is syntactically valid in VB6, but has a different effect. In VB6, `strFirstName` would actually be declared a variant. But in Visual Basic.NET, both `strFirstName` and `strSecondName` are declared as strings.

The bottom line is that it is always good coding practice to declare every variable on it's own line. If you've been doing this for a long time, you'll have less trouble changing over to Visual Basic.NET.

Changes in Declaring Arrays

VB developers will need to be especially on guard for a change in the way arrays are declared. Again, as per other language conventions, all arrays are zero-based, and the declared size is the actual number of elements.

This leads to differences with earlier versions. This line in VB6:

```
Dim strNames(10) as String
```

actually creates *eleven* elements (assuming that arrays are left at the default which is zero-based), starting at index 0 and ending at index 10.

That line doesn't work the same way in Visual Basic.NET. The exact same line declares an array of exactly ten elements, starting at index 0 and ending at index 9. A reference to `strNames(10)` would cause an out-of-bounds error. VB developers will be familiar with this convention because it's the way that list elements in list and combo boxes work.

The syntax to initialize values during declaration has an equivalent for arrays. In Visual Basic.NET, a whole array can be initialized like this:

```
Dim strNames(3) as String = ("Moe", "Larry", "Curly")
```

Finally, arrays must be declared with `Dim`, and can no longer be declared with `Redim` and an empty index. `Redim` can only be used to resize arrays in Visual Basic.NET.

No Implicit Loading of Forms

VB forms have always been loaded whenever the first reference was made to them. This was not necessarily good coding practice, but it was supported. Implicit loading is not expected to be supported in Visual Basic.NET, so forms will have to be explicitly declared and loaded just like other objects.

Many Keywords Move to Object Hierarchies

Many traditional VB keywords are being removed from the base language. In most cases, the equivalent capabilities are being replaced with members in an object hierarchy. Here are some of the keywords being replaced, with their new name and location:

Keyword	*Location in Visual Basic.NET (namespace)*	*Method/Property*
Circle	System.Drawing.Graphics	DrawEllipse
Line	System.Drawing.Graphics	DrawLine
Atn	System.Math	Atan
Sgn	System.Math	Sign
Sqr	System.Math	Sqrt
Rnd	Microsoft.VisualBasic.Compatibility.VB6	Rnd
Round	Microsoft.VisualBasic.Compatibility.VB6	Round
Lset	System.String	PadRight
Rset	System.String	PadLeft
DoEvents	System.Winforms.Application	DoEvents
VarType	System.Object	GetType (returns an object of class Type, which has properties to get information)

This list should not be considered exhaustive. It just highlights some of the major changes.

Miscellaneous Changes

Optional parameters are still supported in Visual Basic.NET, but they now require a default value. The Is Missing construct is not supported.

Evaluation of conditionals is being brought into line with other languages. In particular, if a conditional has two parts, but evaluation of the first part makes it unnecessary to look at the second part, then no evaluation of the second part is made. Consider this line:

```
If (Len(strFirstName) <> 0) And (Len(strLastName) <> 0) Then
```

If the length of strFirstName turns out to be zero, then the conditional is false and it is unnecessary to evaluate the second part. This speeds up execution, which is why other languages do it this way.

Sloppy coders have been known to put function calls into conditionals which carry out some action that changes values. They might write code such as:

```
If (Len(strFirstName) <> 0) And (InitializeRecord(intID)) Then
```

Such code could be affected by the change. In VB6, the InitializeRecord function would always be executed when the If is tested. In Visual Basic.NET, it will not be executed if the first part of the conditional is false. Bugs introduced by this change could be really difficult to track down.

The Debug object gets some changes in Visual Basic.NET. It is at System.Diagnostics.Debug, and its commonly used Print method has four replacements: Write, WriteIf, WriteLine, and WriteLineIf.

Other new Visual Basic.NET capabilities include new options for formatting strings and parsing numbers, and new shorthand syntax for incrementing and decrementing a variable.

Impact on Current Projects

Designing for Web Forms

Many current VB projects will be candidates for converting to Web Forms to gain web functionality. This will not be a transparent conversion, but there are some things that can make it easier. Many of them are simply good design principles, but they are worth reiterating.

Any tiered design will transfer more easily to Visual Studio.NET. Fat VB clients, where all the business logic is mixed up with the user interface code, will probably not be practical candidates for migration. This gives even more reasons for current projects to be undertaken with good multi-tiered design.

VB designs depending heavily on code in control events may have problems in migration. The difficulty is that many events in the Web Forms world will require a server roundtrip. In a traditional VB form, events can be fired constantly and handled without noticing a performance hit, but that will not be true for Web Forms.

Calling Controls Directly

Another commonly used VB technique is to call a control on a form directly to get a value out of it. Controls are always considered "Public" in all existing versions of VB, so code like this works fine:

```
' code executed from a .BAS module in VB6 or earlier
frmEmployee.txtEmployeeID.Text = strNewEmployee
```

This syntax does not work in the PDC tech preview version of Visual Basic.NET. The error message indicates that the control is now considered Private.

There are good alternatives to this technique. In VB4 and later, custom properties are typically the best choice. So, to fix the example above, `frmEmployee` would have a property procedure that looks like this:

```
Public Property EmployeeID As String

    Get
        EmployeeID = txtEmployeeID.Text
    End Get

    Set
        txtEmployeeID.Text = EmployeeID
    End Set

End Property
```

The code in the `.BAS` module would then become:

```
' code executed from a .BAS module
frmEmployee.EmployeeID = strNewEmployee
```

This technique has other advantages which include the capability to change out controls on the form without affecting the calling logic in other routines. That is, if the employee ID in the form were moved to a label instead of a text box, the code in the `.BAS` module would not need to change. Only the code in the form's property procedure would have to change to work with the label instead of the text box.

Syntax Conventions in Visual Basic Projects

There are several areas where coding conventions for present VB projects can relieve some of the problems likely with later migration to Visual Studio.NET. Here is a quick summary:

- Don't use default properties or methods
- Use parentheses even when they are optional
- Avoid API calls, or wrap them in a component for ease of conversion
- Only one variable on each declaration line
- Don't use `Currency` data types and avoid use of `Variants`
- Consider controls to be private to the form they are on – do not refer to them from outside the form (use property procedures instead)

Recommendations

What else can you do to prepare for Microsoft.NET? Here are some suggestions.

Get the latest beta of Visual Studio.NET you can lay your hands on

The beta program for Visual Studio.NET will be very different from the beta cycle for previous versions. Some later beta versions will be available to anyone who requests them. As of writing, only the PDC tech preview of Visual Studio.NET is available, and this was only distributed to attendees at the conference. But the first real beta was promised "in the fall" by Bill Gates in his keynote at PDC, so it should be appearing soon. It is not known yet how widely this beta will be distributed.

Start using the SOAP toolkit for Visual Studio 6

As previously mentioned, SOAP is a foundation technology of Microsoft.NET, and Web Services uses it as a protocol. Anyone expecting to remain current with leading-edge Internet-based systems should become very familiar with SOAP and systems based on it. Fortunately, that familiarity can be gained right now using Visual Studio 6 plus the SOAP toolkit, which is available for download at http://msdn.microsoft.com/xml/general/toolkit_intro.asp.

Visual Studio.NET will use SOAP extensively, and will hide a lot of the implementation details. The Visual Studio 6 SOAP toolkit requires a lot more manual work to get a distributed component working than will be needed in the .NET framework. But it's worth the effort to become knowledgeable about the capabilities and limitations of SOAP – especially if you plan to implement SOAP-compliant components on a non-Microsoft platform.

Start development projects with beta versions of Visual Studio.NET

With a change this big, and with some significant migration costs from applications based on older architectures, it will be desirable for some new projects to be undertaken with Visual Studio.NET even before its official release. This is not for everyone, but if a project looks long enough and probably won't be implemented until Visual Studio.NET is in full release, the risks involved might be worth it.

The safest course is to only begin such projects after beta 3 of Visual Studio.NET is available. In the Microsoft world, beta 3 usually signals that a real release product is not too far in the future. It is also usually stable enough to do work without too many distractions from unfixed (or previously undiscovered) bugs. Realize, though, that beta 3 of Visual Studio.NET could be quite a way off.

If one's appetite for risk is high enough, and a project looks like it might take a very long period of time, it might be worth starting with earlier beta versions. But this option should be considered carefully. No one wants to get a project finished and not have production-ready software to run it, and the delays associated with bugs in beta versions get worse the earlier the version that's involved. Also consider that, the earlier a beta is, the greater the chance that some functionality will be changed or dropped in the released version.

Conclusions and Cautions

No one knows for sure if Microsoft.NET will live up to it's billing. The possibilities look very good. Many of us were surprised at PDC to see how far Microsoft had progressed with this very ambitious effort, but there remains a lot to be done. Building in the intelligence that will be required for Microsoft.NET to work is a huge undertaking, and there are risks that some pieces of it might take another generation or two to be truly ready for prime time.

Remember that we are discussing unreleased products. There is always the possibility of changes during the development cycle. In particular, many of the changes relating to language syntax and features are subject to revision. Bottom line – don't bet the farm on the information presented here.

There is also lots of uncertainty in the time frames that will be required before the first .NET technology rolls into production status. In the meantime, there are ways we can prepare for the next generation in our current projects, and we need to watch Microsoft.NET carefully to see how it develops.

Much of the material here is included in a chapter on Microsoft.NET in the upcoming book Professional Windows DNA, from Wrox Press (ISBN 1861004451).