Migrating Visual Basic Applications to the .NET Platform

California Software Labs
96 Corporate Park,
Suite 100,
Irvine, CA 92606,
USA.

Phone: (949) 474-1363
Fax: (949) 474-1362

www.cswl.com
info@cswl.com
Migrating Visual Basic Applications to the .NET Platform

Index

* Introduction
* Why Migrate to VB.NET?
* Advantages of VB.NET
* Methods of Migrating
* Issues While Migrating
* Migration Strategies
* Conclusion

Introduction
Migrating to the .NET platform has been under serious discussion ever since Microsoft launched its newest series of products under much hyped vision statements. Visual Basic.NET is one of the key products that enable application development under the new vision. However, VB.NET is not quite backward compatible with prior versions like Visual Basic version 6. This makes migration a serious issue. This article looks at possible reasons that one would choose to migrate, and outlines certain important issues to be considered when such a migration is attempted.

Why Migrate to VB.NET?
The first question that strikes one’s mind is “why should I migrate?” Managed Code is a very important feature of the .NET architectural framework. This is what makes Java applications robust when compared to older generation Microsoft applications. VB.NET code targets the Common Language Runtime (CLR) by compiling into an intermediate language, which is then executed under strict control (managed). CLR manages the code in a very similar fashion to the Java model, thereby making applications much more Robust, Stable and secure. The application also becomes more maintainable because of the managed code.

Under the .NET framework, VB.NET applications are packaged as self-contained, self-describing, versioned assemblies (collection of classes and meta data). One can deploy, remove and manage N different versions of an application or DLL on the same machine without creating conflicts. This enables a product company to easily manage versions and upgrades without causing the headaches so common with usual upgrades or OEM-versioning.

Desktop applications can be much more easily converted to Web Based applications using the WebForms paradigm of .NET made available through VB.NET.

XML Web Services creation and integration as facilitated by the .NET platform are readily available through VB.NET. This makes the creation of hosted applications easier, thereby providing wider access to proprietary intellectual property. XML and web services also enable Enterprise Application Integration through standard methodologies as available through the Microsoft family of .NET servers.

Microsoft strategy is to eventually migrate all its products to the .NET frameWork. This would mean that a couple of years down the line VB6, and other non-.NET products would be obsolete.

.NET applications when developed carefully can be platform independent. Managed code built using .NET will run on platforms that host the .NET framework irrespective of the Operating System or machine type. Such non-Microsoft platforms have started to appear already.

If one is convinced of a necessity to migrate to VB.NET, the next question that arises is “How to migrate?” Let us first take an inside view of the technical advantages of VB.NET and then see the various strategies for migration. VB.NET implements a new paradigm and has many advantages over VB6, few of which are listed below.
Advantages of VB.NET

First of all, VB.NET provides managed code execution that runs under the Common Language Runtime (CLR), resulting in robust, stable and secure applications. All features of the .NET framework are readily available in VB.NET.

VB.NET is totally object oriented. This is a major addition that VB6 and other earlier releases didn’t have.

The .NET framework comes with ADO.NET, which follows the disconnected paradigm, i.e. once the required records are fetched the connection no longer exists. It also retrieves the records that are expected to be accessed in the immediate future. This enhances Scalability of the application to a great extent.

VB.NET uses XML to transfer data between the various layers in the DNA Architecture i.e. data are passed as simple text strings.

Error handling has changed in VB.NET. A new Try-Catch-Finally block has been introduced to handle errors and exceptions as a unit, allowing appropriate action to be taken at the place the error occurred thus discouraging the use of ON ERROR GOTO statement. This again credits to the maintainability of the code.

Another great feature added to VB.NET is free threading against the VB single-threaded apartment feature. In many situations developers need spawning of a new thread to run as a background process and increase the usability of the application. VB.NET allows developers to spawn threads wherever they feel like, hence giving freedom and better control on the application.

Security has become more robust in VB.NET. In addition to the role-based security in VB6, VB.NET comes with a new security model, Code Access security. This security controls on what the code can access. For example you can set the security to a component such that the component cannot access the database. This type of security is important because it allows building components that can be trusted to various degrees.

The CLR takes care of garbage collection i.e. the CLR releases resources as soon as an object is no more in use. This relieves the developer from thinking of ways to manage memory. CLR does this for them.

Methods of Migrating

After getting a feel of VB.NET, let us now look at the methods to migrate the application from VB to VB.NET. The various methods that can be used to migrate the application are covered below. In many cases, these approaches will be combined. A redesign from scratch may be better for some code modules whereas many others might need only language/syntax level porting.

MigrationWizard:
Visual Studio.NET provides a Visual Basic Upgrade Wizard (also called as Migration Wizard). This wizard does almost the entire migration barring some modifications that must be made to complete the upgrade process.

Fresh Design:
Redesign of certain core modules or perhaps the whole application is a possible approach if new features of the .NET architectural framework are considered. Managed code (under CLR) can run in tandem with unmanaged code (e.g. COM components). Therefore it is possible to reuse many existing components as such. However, paradigm shifts towards XML web services, Web Forms, Windows Forms etc. call for some rework in design, coding and deployment.

Use of Customized tools to enhance migration:
Porting driven by a wizard and re-design are easier when migrated using customized tools like the “Cal-Migrater” developed internally by CSWL for a health care product migration. This tool makes use of both the wizard and redesign approach to make faster pace through the .NET porting activity.
Issues While Migrating
However there are a few issues that are to be taken care of while migrating from VB to VB.NET. As mentioned earlier, porting the application using the wizard approach does automate the process to an extent but some modifications must be made to complete the upgrade process. The wizard produces a report in HTML format with the information of the places where it failed to migrate the application to .NET. There are number of changes that can be done to the VB6 application to make the upgrade process much more efficient. Some important aspects are listed below.

Avoid using late binding. This is because properties and methods cannot be verified during the upgrade process.

Specify default properties. VB6 specifies a default property for every component. For example, the default of Textbox is Text property and Caption is default for Label component. But in VB.NET there is nothing called as default property.

Use Zero-Bound Array. In VB we can declare an array with any positive integer as its lower bound. But in VB.NET all the arrays are zero bound.

Examine API calls with fixed length strings in VB application because VB.NET doesn’t support fixed length strings.

Lines and Shapes are not supported in VB.NET and hence cannot be upgraded. Instead a graphic object is provided for shapes.

Use constants instead of underlying values. VB6 constants will convert to the correct value when upgraded to VB.NET, but if actual values are used then it may end up with wrong hard-coded values. For example, for Boolean values using -1 and 0 instead of True and False and will have an adverse effect in VB.NET because in VB.NET True is 1.

Re-Design of the whole application can be considered if the application will Grow over time. VB.NET being object oriented, the code can be reused and the application can be extended with less effort and time. Doing the same in VB6 would need a lot more time&effort.

Be taken online, i.e. web application. VB.NET provides web forms that can be used both for web application and VB application. This gives it dual manageability.

Need free threading modules which perform background tasks.

There are two strategies to migrate using the above methods viz. Vertical migration and Horizontal migration. Both these strategies help us find out key areas that need an immediate upgrade and the modules that can be upgraded later.

**Vertical Migration:**
In vertical migration, a module of the application that is fairly standalone or isolated is taken, and all the tiers of this module are migrated to VB.NET. Vertical migration can be adopted in situations where the modules are mostly loosely coupled and/or we are not sure of the risk and effort factors of total migration. Thus migrating one module will give a good idea on the effort required to migrate other modules.

**Horizontal Migration:**
In horizontal migration, one layer of all the modules is migrated to VB.NET followed by other layers. For example, first the UI would be migrated then the middle tier. Horizontal migration is ideal in situations where the modules are tightly coupled and the effort and risk involved in the migration is known to a large extent.
Conclusion
Microsoft is migrating all its technologies into the .NET platform. There are several reasons and advantages for migrating existing applications to .NET. This article has taken a summarized look at some of the benefits and issues associated with such a migration. From our experience, if a Visual Basic GUI intensive application is well structured and reasonably organized, then it is reasonable to assume that code modifications and additions will be on the order of about 30-40%. CSWL provides methodologies, techniques and tools to reduce the migration time.

Index