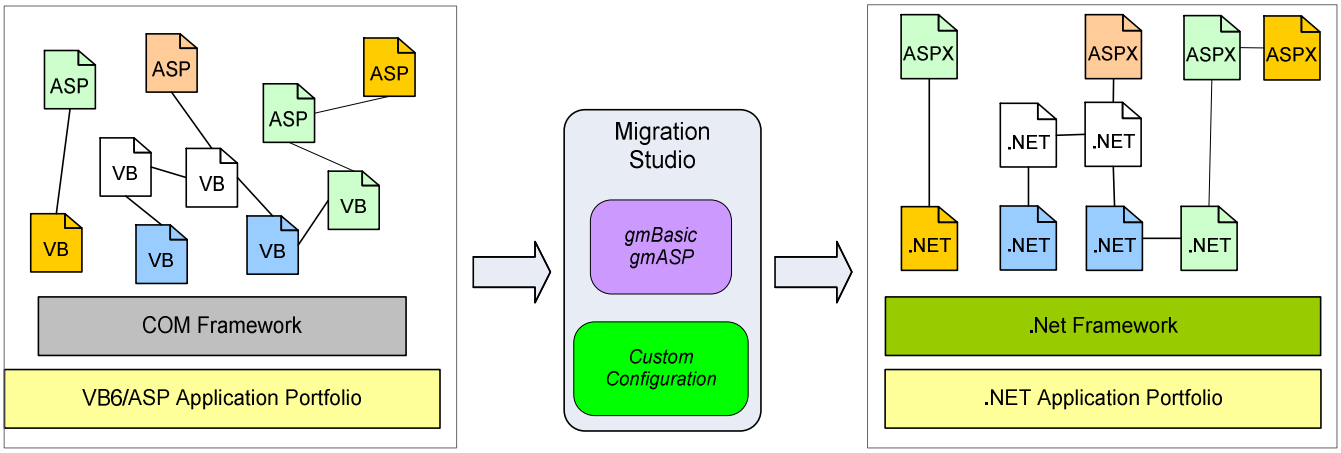
gmBasic Product Overview

George Juras
Great Migrations LLC
June, 2005

gmBasic, a Tool for Taking Visual Basic Applications to .NET

Our translation technology, **gmBasic**, supports custom migrations of Visual Basic/COM Applications to .NET via translation to C# or VB.NET.



We developed and tested this tool in 2004 and we are now looking to apply it to the systematic migration of commercial, large-scale VB applications to .NET.

The Legacy of Visual Basic

Since its introduction by Microsoft in 1991 as Visual Basic 1.0, VB has become the language of choice for millions -- some say more than six million [1, 2] -- of application developers around the

world. Since that time, programmers were hooked on VB because VB makes Windows application development easy - and relatively inexpensive; it is probably cheaper to develop applications on the Windows platform with VB than with any other language. Perhaps it is for this (economic) reason that hundreds of thousands of organizations around the globe have invested in VB to develop hundreds of thousands of applications - ranging in scope from small desktop programs to large-scale, mission-critical enterprise applications.

Of course, all these applications would be lifeless without the substantial VB Runtime Infrastructure (COM architecture) on which they run. This infrastructure, which gives VB applications their powerful and rich functionality, also makes them inexorably dependent on the so called "VB.COM" platform (see **http://classicvb.org/**).

Over the past 13 years, VB and its supporting infrastructure have evolved through six major versions and an ever-expanding feature set. VB1 (March 1991) was the visual generalization for Windows of "classic" BASIC (Beginner's All Purpose Symbolic Instruction Code), a venerable language invented at Dartmouth College in 1963 as a teaching tool and ported to the first PCs in the 70s and 80s by none others than Bill Gates and Paul Allen, later of Microsoft. VB1 supported limited database programming and had no Web development capabilities. VB3 (June 1993) added the DAO data control and allowed application development with Access databases. With Windows 95, VB4 (October 1996) introduced 32-bit development and added support for class modules and DLLs. VB5 (April 1997) delivered programming productivity improvements with Intellisense in code and ActiveX control authoring. VB6 (October 1998) introduced Internet programming with WebClasses and ActiveX DHTML pages.

.NET Breaks Backward Compatibility with Visual Basic

Up to VB6, the transition from one version of the VB methodology to the next was relatively continuous (and relatively inexpensive), except perhaps for VB3, which had some discontinuity from VB2. Since 2001, however, with the introduction by Microsoft of the .NET framework, there is no "VB7" to which the huge existing VB codebase can be upgraded easily and inexpensively. Released in February 2002, Visual Basic.NET, the Visual Basic for the .NET Framework, is not a continuous evolution from VB6. It is a real paradigm shift in programming methodology -- very different from the way of programming familiar to most VB programmers.

A VB6 (or earlier version) project will neither compile nor run in the Windows .NET environment, without significant conversion of its codebase to some target .NET language, say, VB.NET, C#, or C++.NET. Of course, a conversion to Java would make it possible to compile and run a VB6 project on other platforms besides Windows.

Just as VB1 opened up Windows development, the .NET languages opened up the development of scalable Web and server applications. They provide the technologies to bridge the gap from traditional client-side applications to the next generation of Web services and applications. Unfortunately, the owners of VB6 (or earlier version) applications cannot take advantage of these newer technologies without spending a significant amount of time and money converting those applications to run and yield reproducible results and behavior on these newer platforms - let alone upgrading them to take advantage of new features offered by these platforms.

The Cost of VB Migration

For the transition to VB.NET, Microsoft offers the Visual Basic Upgrade Wizard as part of its Visual Studio ..NET, its main programming tool for the .NET platform. Though it helps some, the Upgrade Wizard is by no means a complete solution (see, for example, "The VB.NET Upgrade Wizard" **[2]**). Also, according to this review "Abandoning the Fantasy of VB Migration Wizardry" **[3]**, there are a number of labor-intensive steps in the migration process of a VB application:

- Refactoring the application to change its reliance on unsupported technologies--before the Upgrade Wizard can be of any assistance. If the application uses features like ActiveX documents, graphics, RDO/ADO data binding, and GoSub, then one should be prepared to spend significant time and labor refactoring the code manually.

- Using the Upgrade Wizard to convert VB6 code to VB.NET code. Except for simple applications, this is an incomplete conversion; the result of the wizard is an issues report telling you what some of the problems are with the conversion and what you need to do by hand to complete the conversion.

- Making manual modifications to complete the code conversion process.

- Building and testing the converted application on the .NET Framework.

- Repeating Steps 1 through 4 if you do not get matching results in Step 4

The transition to Visual Basic.NET from VB6 -- with or without the Upgrade Wizard -- is a significant (and costly) undertaking and should not be underestimated. A Gartner Group study [4, 5, 6, 7] estimates that

"The cost of converting applications from Windows DNA to .NET may surprise even those who are familiar with these technologies. Gartner's cost model shows these expenses can be as much as 60 percent of the original development cost."

Since the cost of upgrade even to VB.NET is so significant, a lot of VB application owners are evaluating other migration options, such as migrating to C# for .NET or to Java for J2EE.

Microsoft to Retire Visual Basic 6 by 2008

VB application owners may be asking the question

If it costs so much to convert an application to .NET or any other target platform, why not stay in VB6? If it ain&apost broke, why fix it?

A short answer for this is

Microsoft has declared that mainstream support for VB6 ends in March 2005 and that all VB6 support will end altogether in March 2008. "Visual Basic 6.0 will no longer be supported starting March 2008." [8]

To which a conservative VB application manager may ask a follow-up question

Since VB6 retirement is almost 3 years away, why worry about a conversion now?

There is a short answer for this also

Because the conversion takes a long time to plan and do – especially if you plan to upgrade the application (and not just port it) to take advantage of new technologies available on the target platform.

Of course, Microsoft's decision to end standard support for VB6 by the end of March 2005 has sparked an outcry from VB developers (see http://classicvb.org/), which makes this page all the more relevant and the migration option it covers very timely.

Great Migrations, a Long Tradition in Translation

Great Migrations is carrying forward the legacy of Promula Development Corporation and is home of the world's foremost computer language translation technology. [9] We make industrial-strength code conversion and re-engineering tools and provide comprehensive software migration services to our clients worldwide. "Promula" is a short name for Processor of Multiple Languages.

Since 1987, our translation technology has spawned several translation tools, including the Promula FORTRAN to C Translator (pfc), which is our oldest and best known translator. This tool translates FORTRAN source codes, from a variety of dialects, to correct (compilable) and clean (readable and maintainable) C source code. 100 percent conversion rates are achieved routinely; no manual modifications of the resultant translations are needed to achieve reproducible results on a variety of target platforms.

Over the past 30 years, we have done (or have helped our clients do) hundreds of legacy system migrations worldwide. Some of these migrations are listed in [9]. For the large-scale migrations, in addition to the automatic application code conversions, we have also developed the runtime infrastructures needed to run the applications on their target platforms and achieve matching results. Our migration approach delivers complete re-hosting solutions without re-writing any of the application source code by hand.

In addition to FORTRAN, we have applied our translation technology to the migration of legacy systems from a variety of third generation languages (MU BASIC, COBOL, RPG, PASCAL, PL/1, etc.) and platforms (IBM, DEC, VAX, PDP, SUN, HP, CRAY, PRIME, Data General, UNISYS, Honeywell, etc.) to a variety of contemporary platforms (Windows or various flavors of Unix) via translation to C.

A Translation Approach to Migration

We believe that our approach is ideal for the migration of Visual Basic applications to .NET, because its greatest strength addresses the greatest weaknesses of other available tools. Microsoft estimates that its Upgrade Wizard is able to convert approximately 95 percent of the actual VB statements to VB.NET. This may or may not be true [4, 5, 6, 7]. Regardless of the validity of this claim, however, it is the leftover 5 percent or 10 percent of statements and features of VB that cause the problems. It is what the Wizard leaves undone that will require the bulk of the manual effort, which you wanted to avoid in the first place when you considered using the Upgrade Wizard to aid in the conversion to VB.NET. Of course, the Wizard isn't much help if your target language is C# or Java
.

There are many sources (books and manuals) available that describe the feature by feature and statement by statement correspondences between VB and the target languages of VB.NET, C#, and Java. These correspondences can be implemented using simple rewriting rules for the most part. It is the last 5 percent to 10 percent of statements and features of VB that cause the problems. These cannot be resolved via simple correspondence tables; rather they require digging into the code, understanding what's being done, and integrating that understanding into the functionality of a Promula translation tool. This is the power of Promula.

Other conversion tools view translation as a syntax and symbol management problem. Such tools process statements in the source program, recording all symbols used. Then they make what changes are necessary in the symbols and in the statements to produce a compilable version of the code in the target language. Putting it simply, other tools look at what the program "says" in the source language, and then they try to "say" the same thing in the target language.

A Promula tool also processes statements in the source program recording all symbols used, but it also records all operations performed in the program using a low-level pseudo-code. This low level pseudo- code is a formal, language-independent representation of what the program does. The pseudo-code is then analyzed and reorganized as required. Finally, the target source code is written using the target language completely independently of the original source. Putting it simply, a Promula tool asks what the source program "does", and then it writes the code in the target language to "do" the same thing.

gmBasic -- VB Compiler

The gmBasic tool has six subcomponents:

- A VB Compiler
- A Pseudo-code Analyzer and Re-structurer
- A Pseudo-code to Source Code Author
- An IDL Converter
- An Frx to Resx Converter
- An Information Auditor
-

The VB Compiler is a two-pass pseudo-code compiler which takes the source code of a group of VB projects as input and produces a compiled representation for that source code. The compiled representation contains a hierarchical symbol table, which is formed on the first pass of the compiler, and a pseudo-code listing of all operations performed. This pseudo-code encompasses not just the user operations defined in subroutines, functions, etc., but also the declarations, settings, and references made in the project and form files. With the exception of the expression parser which deals with VB operator hierarchy, no attempt has been made to "generalize" the compiler. We use a simple tokenizer and look at each statement individually in the code. Visual Basic has up to 6 dialect flavors now and with its Variant types and late binding conventions has an open syntax. Each syntax error that we detect is given its own number and description.

The VB compiler has been tested with some 300 simple VB codes that are intended to include all of the traditional VB statements.

gmBasic -- Pseudo-code Analyzer and Re-structurer

The pseudo-code analyzer and re-structurer is the true work-horse of the system. It is this subcomponent that deals with and resolves the types of issues that were discussed in the previous section. Pseudo-code is easy to scan and easy to manipulate. This is the true beauty of our approach. The compiler only has to worry about dealing with the source code and getting its meaning correctly. That "meaning" itself has a surprisingly simple linear structure once it has been separated from the source code.

It is easy for the analyzer to notice that a subprogram has Variant parameters. It can then scan the code for all calls to the subprogram to determine if the calling arguments require different semantics or perhaps only require casts. If multiple versions are required, it can create multiple copies of the subprogram at the p-code level, with each copy having the appropriate parameter type and operators specified. The analyzer can certainly scan the code within a given subprogram to look for scope violations, and then only move those declarations that require it. Finally, adjusting array subscripts is a simple matter of inserting an additional subtraction operation into each array subscript expression. The analyzer is not table driven at this time. As the product matures we will add more and more capabilities to the analyzer. A problem which we have not resolved is that individual migrations often have special re-structuring needs; thus, the staff doing the migration must be able to enhance their own code without explicit support from Great Migrations.

gmBasic -- Pseudo-code to Source Code Author

The pseudo-code to source code authoring tool, as distinct from the compiler and the analyzer, is a generalized table-driven capability using what we refer to as "pattern-strings and codetext-tables". Many clients, especially those with large codebases, have their own target language coding standards or will want to use their own external APIs to implement the runtime capability. The decisions about how to do things in the target language must be made separately for these clients and should not be hardwired in the tool. Obviously, there are default tables for all supported languages, which the migration staff will modify to meet particular needs. Though these tables are not difficult to work with, training is obviously needed, since most people are not used to the concepts of language description at the abstract level.

gmBasic -- IDL Converter

VB projects contain references to external objects. For example

```
Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0"; "comdlg32.ocx"
Object = "{831FDD16-0C5C-11D2-A9FC-0000F8754DA1}#2.0#0"; "mscomctl.ocx"
```

A library must be built of all of the things that people might reference in their VB projects, such as the following:

- OCXs - COM controls
- DLLs - COM class libraries
- TLBs - COM type libraries
- OLBs - COM object libraries

Once the particular entities -- interfaces, coclasses, properties, methods, events, constants, typedefs, enumerations, etc.-- are defined in a library using gmBasic conventions, we can use them to control the compiler and to tell the translator how these things should be expressed in the target languages.

A MicroSoft tool, oleview, can be used to extract (from their binary representations) and display the entity descriptions for all of the above components . Oleview writes these descriptions in IDL (Interface Definition Language) notation. The gmBasic IDL converter then compiles these IDL representations into a form that can be used by the VB compiler to process references to these external components.

For example, the gmBasic XML representation of the COMCtl32.ocx control is **comctl32.xml**.

gmBasic -- Frx to Resx Converter

VB stashes property information assigned at design time to Forms in FRX files. In addition, there are even more binary files that provide the same service for User Controls (.CTX), Property Pages (.PGX), User Documents (.DOX), and Active Designer projects(.DSX). Among the various property information stored in these files are graphics, which can include bitmaps (.BMP and .DIB), GIFs, JPEGs (.JPG), Metafiles (.WMF, .EMF), cursors (.CUR), and icons (.ICO).

Data stored in these binary files is accessed via the binary file offsets assigned to the properties of the accompanying module (i.e., [Form.]Picture = "Form1.frx":0000). The original graphic files themselves are stored in their entirety in VB binary property files. VB also adds binary header information before each graphic file in the binary file, including the length of the graphic file itself. But since this graphic file header information has changed as VB itself has changed, reading frx files is difficult. The only definitive way to determine exactly where the graphic file starts and ends, is to verify the graphic file's format. This entails reading the bits of the binary file and comparing it to the known formats of the graphic file formats mentioned above.

In the .NET environment this same information is stored in resx files using base64 encoding within an XML document. The actual format for Base64 encoding as described by http://www.freesoft.org/CIE/RFC/1521/7.htm as follows.

The Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that need not be humanly readable. The encoding and decoding algorithms are simple, but the encoded data are consistently only about 33 percent larger than the un-encoded data. This encoding is virtually identical to the one used in Privacy Enhanced Mail (PEM) applications, as defined in RFC 1421. The base64 encoding is adapted from RFC 1421, with one change: base64 eliminates the "*" mechanism for embedded clear text.

A 65-character subset of US-ASCII is used, enabling 6 bits to be represented per printable character. (The extra 65th character, "=", is used to signify a special processing function.) The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8-bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first byte, and the eighth bit will be the low-order bit in the first byte, and so on.

The frx to resx converter reads resx file references as they are encountered in the various VB source files and then transforms them into a form that can be stored in and accessed from resx files in the .NET environment.

gmBasic -- Information Auditor

The primary rule for performing source code migrations is **"The problems are in the details."** To determine how particular features are to be dealt with one must often look at the details of the code information as generated by the compilers - VB, IDL, and FRX -and as modified by the analyzer. To make this possible the migration tool has an extensive information auditing tool which can selectively display the details about all information being stored by the tool. This information can be produced either in tabular form for viewing with a text editor or to ease readability in HTML form for viewing by a browser.

The Cost of VB Migration -- Revisited

Our approach to VB migration, we believe, significantly improves on the high cost of conversion (as estimated by the Gartner Group and debated by others [4, 5, 6, 7]). Microsoft claims 95 percent conversion to VB.NET for their Upgrade Wizard. There is confusion about what these cost estimates mean. Gartner is measuring level of effort to do a migration, while Microsoft is counting something like "number of lines converted", which is really a meaningless number. At any rate, as difficult as it is to estimate what the cost of a particular conversion to .NET will be, it is safe to say that it is not going to be trivial.

Though we did achieve 100 percent FORTRAN-to-C conversions using the Promula Fortran tool, we wrote an entire FORTRAN Runtime System to support the converted code.

For VB, we use vendor-supplied APIs to support the translations. We write some wrappers to simplify things, but we do not need to write a VB Runtime System. As a result, we can achieve complete conversions -- for as little as 5 to 10 percent of the original development cost level relative to the much publicized number of 60 percent.

The migrations are not as difficult to achieve, but they are certainly not turnkey. gmBasic achieves very high conversion rates (close to 100 percent) using the standard (right "out of the box") conversion tables of the standard gmBasic tool (at its present level of development and maturity). The tool actually identifies the few places in the VB6 source code which need to be corrected prior to translation -- either because they are in error and need to be corrected anyway or because they are ambiguous (obfuscated code) and need to be re-coded so as to allow a "cleaner" translation to the target language. In fact, the following translation principle is observed rigorously

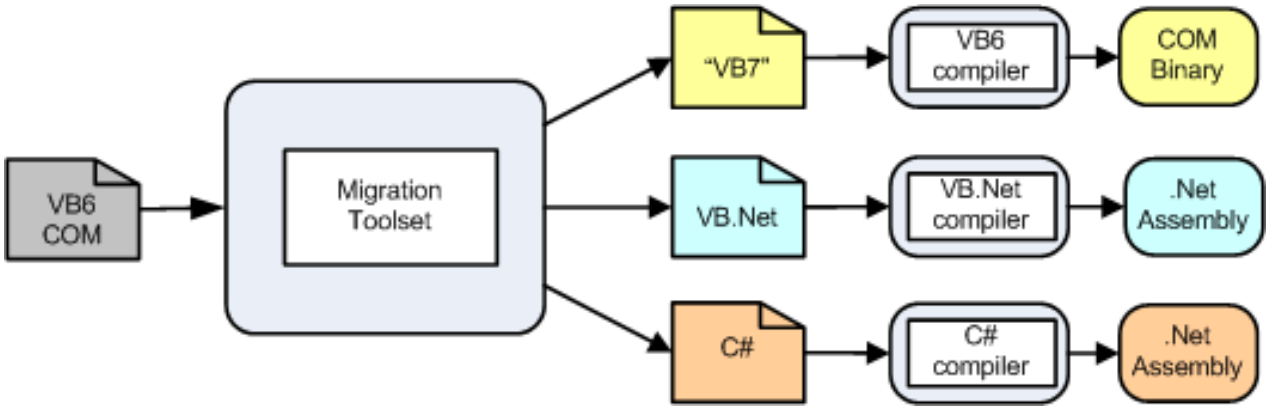
To achieve translation correctness, never change the output of the translation; if you must, correct the input to the translation for the purpose of producing more correct (or cleaner) translations or for the purpose of fixing errors in the original source code, as discovered by the tool.

The tool is then customized and its translation tables are so configured as to achieve 100 percent conversions. Once that is achieved, the entire conversion can be done automatically and, thus, repeatedly, until it satisfies all migration criteria. In fact, the translation tool may be integrated into an existing development process, where the conversion simply becomes a routine compiler pre-processor step -- until such time that a decision is made to retire the original codebase and begin developing from the translated codebase, on the target platform.

Proof of Concept Translations

Recently, we applied our approach to VB migration on two proof of concept (POC) projects. The first POC was designed to show that gmBasic is a state-of-the-art VB translation methodology that covers all aspects of the "classic" VB language. The second POC was designed to show that gmBasic is a competitive migration option that can be used with confidence to migrate and verify major, commercial-grade VB projects to .NET in a timely and cost- effective manner.

As part of the first POC, we constructed a VB Migration Suite, consisting of a select group of about 300 VB projects which deal with the major components of the VB language and which highlight some of its major problems (related to migration). We converted these test codes to three target notations - VB7, C#, and VB.NET -- which we then tested for compilability, using the three compilers, as shown below. Here, VB7 is re-constituted VB6 code written by gmBasic (after compiling the original VB6 code, analyzing the compiled code and authoring it in VB6 output notation).



We then parallel tested the compiled translations on the target platform (against the original VB6 executable applications) to prove functional equivalence of all three translation sets with the original VB6 set.

The second POC was a pilot project done in support of a proposal development effort calling for the migration to .NET of a mission- critical, commercial enterprise suite of applications (about 500 VB projects implemented in about 1 million lines of VB6 code). In this POC, we translated to VB.NET and C# a sample of 16 VB projects that were extracted from the commercial codebase of a potential client. Here, we proved compilability on the Visual Studio .NET platform and we are about to begin proving functional equivalence by parallel testing.

By the way, the translation to C# and VB.NET of all 16 VB projects in this POC (about 60,000 lines of commercial-grade VB6 code with about 50 component dependencies) took 28 seconds on a 1.4 Ghz Windows 2000 workstation.

From our POC experience, we have learned that the Promula approach can produce accurate and complete migrations in relatively short timeframes and at costs well below the much debated 60 percent level - perhaps as low as 5 to 10 percent of the original development cost.

References

1. "Classic VB Petition FAQ", http://classicvb.org/petition/faq.asp.

2. A Programmer's Introduction to Visual Basic.NET, Craig Utley, SAMS Publishing, 2001.

3. "Abandoning the Fantasy of VB Migration Wizardry", Lori Piquet, February 20, 2002, http://www.devx.com/vb/Article/16822.

4. "A Cost Model for .NET Code Conversion", Gartner Group Research Note COM-16-1385, R. Valdes, M. Driver, 14 June 2002

5. ".NET conversion: what's the cost?", Ron Kassen, June 2002 http://www.geek.com/news/geeknews/2002june/gee20020624015099.htm

6. "Microsoft .Net software's hidden cost", Joe Wilcox, June 2002 http://news.com.com/2100-1001-938394.html?tag=fd_top

7. "Microsoft Response to Gartner Report (was: "Re: Cost of migrating to .Net") ", Ari Bixhorn (Microsoft), June 2002 http://aspn.activestate.com/ASPN/Mail/Message/DevelopMentor- dotNET-Advocacy/1254149

8. "Product Family Life-Cycle Guidelines for Visual Basic 6.0" http://msdn.microsoft.com/vbasic/support/vb6.aspx

9. Promula Development Corporation Website, http://www.Promula.com.

10. Upgrading Microsoft Visual Basic 6.0 To Microsoft Visual Basic .NET, Ed Robinson, Michael Bond, and Robert Ian Oliver, Microsoft Press, 2002.