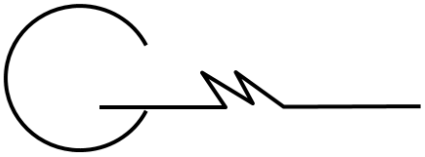
gmBasic Technical FAQ



Mark E. Juras, President
Great Migrations LLC
November, 2006

.NET Versions Supported 1
ASP to ASP.NET Capability 1
Comparison with Other Tools 2
A More Effective Translation Paradigm 2
Industrial Strength Features 2
Source and Target Awareness 2
Multi-Component Awareness 3
Robustness 3
Flexibility 3
Job Control Language 3
Source-Code Analytics 3
Incremental Processing 3
Automate the Automation 3
gmBasic Architecture Value Proposition 3
The gmBasic Compiler 4
The gmBasic Analyzer 4
The gmBasic Author 4
The gmBasic Value Proposition 4

.NET Versions Supported

Q	What version of .NET is your converter written to convert to? 1.0, 2.0, 3.0?

A: gmBasic is only loosely coupled to specific versions of the .NET framework. We currently convert to both 1.1 and 2.0. We also support both VS2003 and VS2005 project formats. We are putting much more focus on VS2005 right now because it is dramatically better than VS2003 and our default is target is Framework 2.0 with VS2005. We will support VS2007 and Framework 3.0 shortly after it is available. We can generate VB.NET and C#.

ASP to ASP.NET Capability

Q	When looking at converting ASP pages, does your converter separate the HTML & Script to create code-behind pages? Or does it simply "rename" the files to ASPX and then fix any compilation errors related to the migration?

A: gmBasic compiles the ASP pages and related business objects into a semantic system model. It then analyzes the entire model, restructures it to conform to the desired architecture, and emits clean, correct target code. It separates source pages into a code-behind file for the script logic and an ASPX file for the HTML.

Note: This is not the same as re-engineering all the pages to use a yet-to-be-determined set of server controls and server-side event handlers. That could also be done, but it is more difficult (both by tool and manually) and you risk changing the look and feel of your application if it is not done just right. Until we understand how you want this done for your application, we recommend that the application development team do next-level refactoring incrementally on the specific pages where it provides the most value *after* the system is migrated to .NET and they are comfortable with ASP.NET.

ASP pages typically contain a mix of static HTML and client-side java script, ASP processing directives, VBScript code blocks, and VBScript fragments called rendering functions. The list below summarized what we can to for ASP to ASP.NET out of the box.

- Blocks of static HTML and client-side java script logic will be passed through to the ASPX page.
- ASP processing directives will be updated to corresponding ASP.NET page directives.
- All VBScript code will be translated to a.NET language; you will have your choice of VB.NET or C#. The resulting .NET code can be organized into code behind files if desired.
- The .NET code from VBScript translations will conform to the same coding standards as the .NET code translated from your VB6 codebase.
- All the.NET code will be restructured to use .NET components instead of COM components.
- The translation process will also produce one or more .NET project files that you may use to pre-compile the site if desired.

Comparison with Other Tools

Q	How/why your tool is different from other conversion tools?

A More Effective Translation Paradigm

A: gmBasic is not an off-the-shelf conversion tool. Rather, it is part of an iterative, tool-assisted migration methodology that refines and balances automated translations with manual work in a way that gets the results you want faster and with less risk than any other approach.

gmBasic is "tuned" to the unique characteristics or your codebase and for the .NET architecture/ coding standards that are appropriate for your migration project. Properly configured, gmBasic translations meet your specific standards and require little or no post-translation coding.

"Other tools view translation as a syntax problem.
They change how things are "said" in one language to how they are "said" in the other language.

gmBasic determines what the program does and then asks how do you "do that" in the target language -- a much easier question and a question whose solution can be achieved systematically using the theories developed by linguists to process natural language.

gmBasic ... writes a program in the target language based solely on the semantic representation. "

Fred Goodman, Chief Technology Officer Great Migrations LLC

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 syntax to produce a 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.

gmBasic processes statements in the source codebase recording all symbols used, but it also records all operations performed in the codebase. This information is combined to build a detailed semantic model of what the program does. The model is then analyzed and reorganized as required. Finally, the target source code is written using the target language independently of the original source. Putting it simply, GmBasic asks what the source program "does", and then it writes the code in the target language to "do" the same thing.

Industrial Strength Features

In addition to using a fundamentally more effective translation paradigm, gmBasic has a number of other features that make it "industrial strength". These features are described below:

Source and Target Awareness

Your source codes are an extremely detailed specification for your applications, but they are only part of the translation picture. The other part is the desired target: architecture design, API replacements, new language, SCM, and coding standards, and even information about anomalies and quirks in your source codes. gmBasic synthesizes both source and

target information to create a holistic picture of the translation requirements. In this context, gmBasic translations preserve the meaning of the source codes and adhere to the target architecture and language standards.

Multi-Component Awareness

gmBasic catalogs the components that it has translated and uses this information to translate dependent components consistently. The result is much cleaner translations and less rework. Although this capability was originally developed for large, multi-VBP application portfolios, it has direct applicability to large VB6/ASP applications as well. In addition to VB6, COM and ASP metadata, gmBasic can also use information from the SQL layer (stored procedure parameters) to bring even more intelligence into the translation process.

Robustness

gmBasic adapts to problems in the source codebase that stop other tools dead in their tracks. For example, gmBasic has an adaptive mode of operation that can work despite missing files, missing references, and VB syntax errors. This mode of operation is not recommended for production translations, but it allows the translation team to overcome roadblocks that can occur in some migration projects.

Flexibility

gmBasic is controlled by configuration files (XML) that provide a high degree of control over translation details. Some of the things controlled via configuration are:

- Rules for mapping of internal and external COM/Win32 APIs to .NET replacements;
- Rules for mapping ASP/VB6 coding patterns to .NET coding patterns (On-Error-Goto to try-catch-finally)
- Processing order for a multi-component translation
- Instructions or codeblocks to address problems or anomalies in the source codes
- Rules for target file names, folder names, and the breakdown of source files into separate targets
- Rules for target project settings (csproj, vbproj)
- Rules for code formatting – blank lines, comments, indenting, boilerplate code (i.e. error handling, comment blocks)
- Special processing instructions that control low-level details of translator operation

Job Control Language

gmBasic has a job control language that allows the migration team to script solutions to one-off translation problems. Job-specific solutions provide an added degree of freedom, and they can be easier to implement than global solutions. This means greater efficiency and lower cost in pushing translation quality to the desired level. Also, scripted solutions are documented in the job control files and applied systematically every time the translation job is executed.

Source-Code Analytics

gmBasic is the fastest, most detailed VB6/ASP reporting tool on the planet. These reports can be useful in helping to plan and estimate a large migration by identifying dependency details and repeating code patterns.

Incremental Processing

gmBasic allows the migration team to break a large multi-component translation into logic units that can be processed separately. Other translation tools are painfully slow and require long-running, all-or-nothing translations.

Automate the Automation

gmStudio, the Great Migrations Studio, is a migration project management tool. It wraps gmBasic and its configuration to facilitate an extensive set of migration tasks. As a result, the migration team does not get bogged down with painstaking and tedious tasks associated with translating large codebases, so they are free to focus on value-added work. gmStudio also facilitates metrics gathering and tracking the progress of large migrations.

gmBasic Architecture Value Proposition

Q	What does "compile & analyze" means and why it's worth so much?

The gmBasic Compiler

The Compiler converts a set of VB projects, ASP pages, and related metadata (COM/SQL) into an extremely detailed system model. The system model describes all of the COM libraries, VB projects, ASP files, functions, variables, types, and operations, in extreme detail. The model describes how the system is composed and how the pieces are inter-connected. This is often referred to as the **Low-Level Semantic Model** because it describes the semantics, or meaning, of the system not its syntax. Furthermore, the semantic model is stored in a proprietary data management system that provides extremely efficient access to and manipulation of the model.

The gmBasic Analyzer

There are many sources (books and manuals) available that describe the feature by feature and statement by statement correspondences between VB6 and the target languages of VB.NET, C#, even Java. The majority of these correspondences can be implemented using simple re-writing 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. The Analyzer enables that integration.

Analyze means to "make sense" of the pseudo-code produced by the compiler and to map the meaning of that code into the desired target forms, reorganizing and otherwise amending the model as necessary. The Analyzer uses information about the desired target platform in many of its algorithms.

The analyzer is the true work-horse of the system. However, because it takes advantage of the pre-preprocessing done by the compiler, and the power of the data management system, we are able to implement surprisingly sophisticated transformations in much less time and with much better results than other translation technologies.

For example, 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 overloads of a routine are required, it can create copies of the subprogram, 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. Adjusting to 1-based array subscripts is a simple matter of inserting an additional subtraction operation into each array subscript expression.

The gmBasic Author

In addition to the Compiler and the Analyzer, gmBasic has a third subcomponent -- the pseudo-code to source code **Author**. The Author uses the migrated semantic model and other information to generate computer codes in the target notation.

The author, as distinct from the compiler and the analyzer, uses 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 and will want to use their own external APIs. 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.

The gmBasic Value Proposition

So why is the Compiler-Analyzer-Author worth so much? The answer is: simply because it is the foundation of a translation methodology that minimizes cost and risk while preserving quality and control. It allows the migration team to refine and channel their collective knowledge into a tool that then reproduces and multiplies what they would do manually. It systematically produces very good code very quickly, and by a repeatable, measurable, improvable process.

With gmBasic, you will be able to translate an entire production codebase from VB6 and ASP to a "known good" .NET codebase – often in a matter of minutes – not just once, but several times, each time learning and making specific improvements until you are satisfied with the translation. When the team is "ready to cutover to .NET", the latest, greatest production source code is translated one final time. The resulting.NET code will meet your coding, architecture, and configuration management targets and you can expect a smooth test cycle and transition.