Upgrading a Visual Basic® Application to .NET:

The e-volutionVisualizer Example

Introduction

The emergence of a new technology brings the opportunity to develop new and more powerful applications. The cost of taking advantage of the new features usually is having to develop entirely new systems and replicating a lot of the functionality of the existing systems.

By using the Internet to enable software applications to work together more easily, Microsoft® .NET promises easier integration within and among businesses while creating opportunities to connect more meaningfully with consumers. Microsoft .NET framework has the ability to turn existing IT infrastructures into a competitive advantage.

To take advantage of Visual Basic .NET it is not necessary to rewrite your existing VB applications from scratch. Microsoft is providing an evolution path by means of an upgrade tool called Upgrade Wizard. This tool, which is embedded in Visual Studio® .NET, has been co-developed by ArtinSoft®.

This document discusses the migration strategy and its implications and also outlines the conversion process of a Visual Basic application to Visual Basic.NET performed by ArtinSoft using the Upgrade Wizard.

Why Migrate To .NET?

If your company develops Web applications or products that run on Windows, then it's virtually certain you'll want to migrate to .NET at some point.

A popular reason for moving an application to Visual Basic .NET is to either Web-enable the application, or enhance an existing Web-enabled application with ASP.NET features such as tracing, flexible state management, scaleable data access, and improved performance.

Visual Basic .NET provides a first-class object-oriented programming language with support for implementation inheritance, free threading, structured exception handling, attribute-based programming, and much more.

The .NET framework from Microsoft, offers the opportunity to develop Web Services and personalized applications as well as a platform for their efficient development which includes the new Windows Forms Designer, Rapid Application Development for the Web with the drag-and-drop WebForms Designer, full Visual Basic .NET code behind forms, and HTML statement completion. Developers can build and consume powerful, integrated XML Web services that reduce development time by enabling software aggregation from any platform.

Additionally, developers have full access to the Microsoft .NET Framework, a comprehensive library of classes and functionality for data access, security, XML support, and more.

It is also possible to build applications that target a vast array of handheld and wireless devices using the Microsoft Mobile Internet Toolkit. If your business data is stored in a database, then it will be unaffected by the upgrade – upgrading the application does not affect the database. Certainly if the application is poorly designed, re-writing can be a good option since it provides an opportunity to ‘do it right’.

If the application already supports your business needs, does not need enhancements, and you already have support staff trained in Visual Basic 6, then leaving the application in Visual Basic 6 is a good option. If there is a business need to move the application to Visual Basic .NET, then we need to look closer at upgrading versus re-writing.

As mentioned previously, re-writing sometimes yields an improved application. The downside is that the development cost will be much greater than upgrading. A good Visual Basic programmer writes between 464 and 667 lines of deployable code per month. The same programmer upgrading an application will process between 40,000 and 60,000 lines per month. Upgrading is 30, and sometimes 80 times faster! The main reason for this huge difference is that a re-write involves writing, testing and debugging new code, whereas an upgrade means simply ensuring code runs as it used to.

There are benefits to re-writing. Re-writing allows you to correct a previous poor design, and COM objects can be replaced with .NET objects that are more scaleable and don’t require registration during deployment. The flip-side of this is that upgrading is much quicker; COM objects can easily be replaced with .NET objects after upgrading; and a well designed Visual Basic 6 application upgrades naturally to a well designed Visual Basic .NET application.

After upgrading the performance is usually very similar to that of the original application and even better if it involves going from ASP to ASP.NET.

Migration Planning

Before you can start your migration planning you need to answer two important questions: When should you migrate, and what is it going to cost you in terms of money and effort.

The best time to upgrade is when the application codebase is stable, but is due for some enhancements. This way, you can combine the upgrade with feature enhancements.

We recommend that you do not attempt an upgrade if your application is undergoing a period of intense change. Upgrading a large real world application may take several weeks or even months. During this time, if developers are also making changes to the Visual Basic 6 codebase, then you have to either make changes in both codebases, or re-synchronize the codebases at a later time by upgrading the changed Visual Basic 6 application again. You can’t re-upgrade a project without losing the changes you already made in Visual Basic .NET. If the nature of the project is such that you have to work on the Visual Basic application while also upgrading it, we recommend you to minimize the disruption by choosing stable modules to upgrade.

Cost considerations

There are a number of expense factors to evaluate when migrating to .NET. You need to calculate both the acquisition and implementation costs of migrating to .NET in general and also estimate how much you can save in intangible productivity gains and maintenance savings.

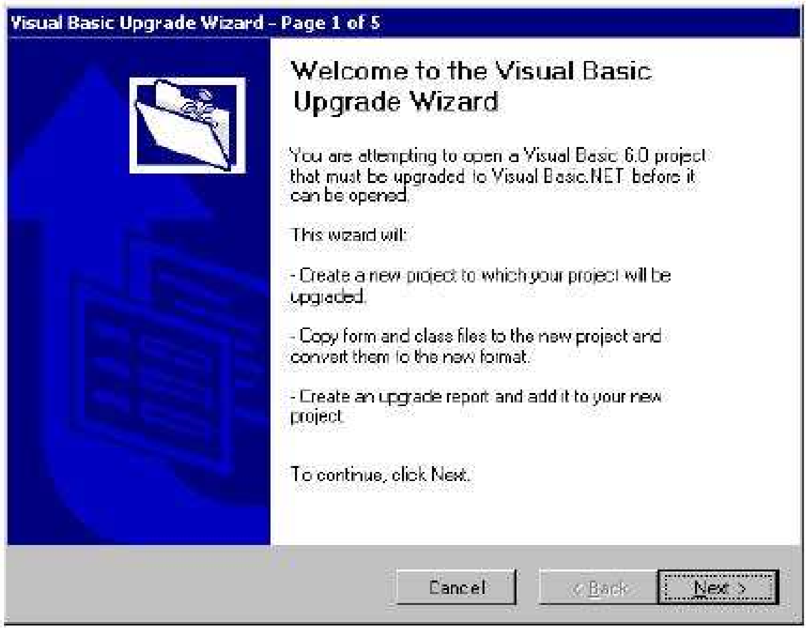
Although there are some short-term costs in migrating to .NET, long-term benefits far exceed these costs.

The stability of your personnel might also factor into your migration timeline. If your staff is about to undergo significant growth or change, nail down your .NET migration strategy first—before making the staff changes. The type of personnel and the desired skills mix could vary significantly if you choose to use legacy technologies or .NET technologies. For example, legacy Web apps need skilled scripting programmers, while .NET Web apps need skilled OO programmers with Web development experience. If you have legacy C++ apps, you wouldn't want to hire a pure VB programmer to maintain your code. However, for .NET development, VB programmers can work side by side with C++ and C# developers, because they're all using the same model and libraries in the

.NET Framework. It then just becomes a difference in syntax instead of a completely different programming model, as it is now.

The Upgrade Wizard

The upgrade's main tool is the Upgrade Wizard that comes along as part of the Visual Studio .Net.



Visual Basic Upgrade Wizard - Page 1 of 5

Welcome to the Visual Basic Upgrade Wizard

You are attempting to open a Visual Basic 6.0 project that must be upgraded to Visual Basic.NET before it can be opened.

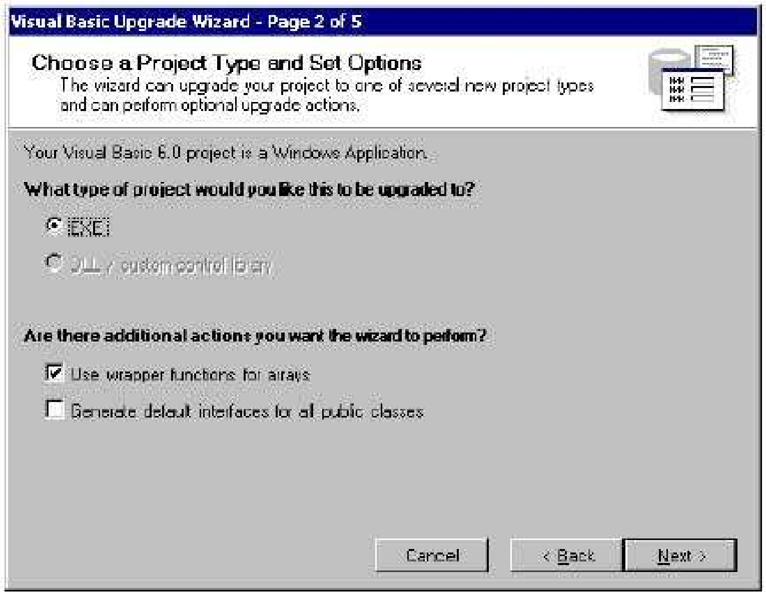
This wizard will:

- Create a new project to which your project will be upgraded.

- Copy form and class files to the new project and convert them to the new format.

- Create an upgrade report and add it to your new project.

To continue, click Next.



Visual Basic Upgrade Wizard - Page 2 of 5

Choose a Project Type and Set Options

The wizard can upgrade your project to one of several new project types and can perform optional upgrade actions.

Your Visual Basic 6.0 project is a Windows Application.

What type of project would you like this to be upgraded to?

[✓ EXE]

DLL / custom control library

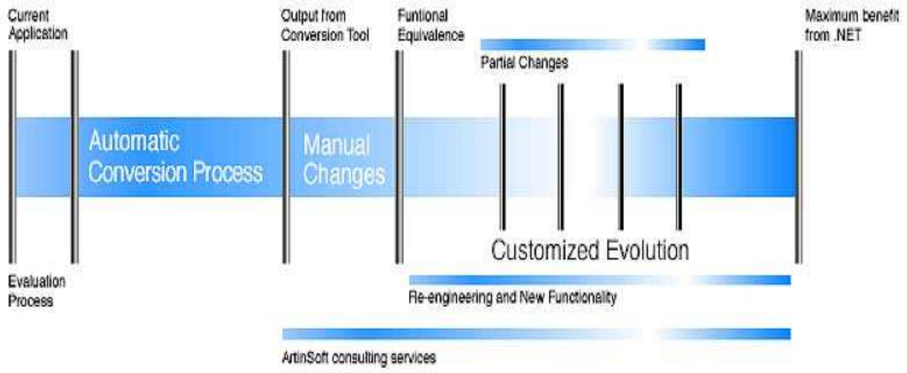
Are there additional actions you want the wizard to perform?

[✓] Use wrapper functions for arrays

[ ] Generate default interfaces for all public classes

The upgrade process consists of three main steps:

- Upgrade Wizard, where most of the code is converted automatically.
- Manual changes, where corrections are made to the migrated code and the features that were not automatically upgraded. Functional equivalence is achieved here.
- Application enhancement, where re-architecture is performed in order to Web-enable the application.


Current Application
Output from Conversion Tool
Funtional Equivalence
Maximum benefit from .NET
Partial Changes
Automatic Conversion Process
Manual Changes
Customized Evolution
Evaluation Process
Re-engineering and New Functionality
ArtinSoft consulting services

The Upgrade Wizard performs the following actions:

- Updating of Intrinsic Controls: The VB Upgrade automatically updates the PME’s of available intrinsic controls in VB6.0.
- Color conversion: The VB Upgrade automatically converts the VB5 colors (OleColors) to .NET colors (Color objects)
- Font conversion: The VB Upgrade automatically converts the VB6.0 fonts (stdFont) to .NET fonts.
- Coordinate conversion: The VB Upgrade automatically converts the VB6.0 coordinate system (Twips) to the .NET coordinate system (Pixels) thus correctly positioning the objects in the forms so they have the same interface as their equivalents in VB.NET.
- Updating of Activex Controls: The VB Upgrade converts the ActiveX controls with their PME’s to the Wrappers used by .NET to support COM controls.
- Updating of "Control Arrays": The VB Upgrade automatically converts the VB6.0 control arrays to the .NET equivalents.
- Conversion of Resources and Binary Resources files
- Conversion of Sstab Activex to .NET intrinsic control

Language (Code):

- Conversion of constants and numeric values: The VBUpgrade updates constants and numerals of VB6.0 to their .NET equivalents when these are present in the code either as literals (i.e. VBCr) or numeric values.
- Renaming of Reserved Words
- Expansion of “Default Properties”
- Conversion of "Casting" data types
- Interface generation for classes implementing others classes.

Miscellaneous:

- Update report
- Generation of EWI’s
- Updating of "WebClasses"
- Updating of AdoDC DataBinding
- Updating of DataEnvironments: Data Environment is a Visual Basic 6 feature that provides an interactive design time environment for creating programmatic, run time data access. Visual Basic .NET provides even more powerful environment for building database applications, but because the underlying data access library is conceptually different (Visual Basic .NET uses ADO+, VB 6 uses ADO), there is no direct mapping between Data Environment in VB 6 and data access features in Visual Studio .NET.

The following features are not automatically upgraded and require re-architecture in .NET:

Feature	**Strategy**
ActiveX Documents	Leave in VB6.0
DHTML Pages	Leave in VB6.0
Add-in Extensibility	Use New Object Model
Lset	Custom Method
DAO RDO Binding	Use ADO or ADO.NET
Diagonal Lines	Use Form Paint Event
Property Pages	Property Browser
ObjPtr/VarPtr/StrPtr	Use Memory Pinning
OLE Control	Use WebBrowser Control
GoSub/Return	Procedures

e-volutionVisualizer: a Case Study

e-volutionVisualizer is a tool developed to provide online access to information through an interactive graphic environment. Designed to be used in the manufacturing industry, e-volutionVisualizer provides a very high level of generality that allows using it for the display of any other kind of space or resource associated information.

From the technical point of view, e-volutionVisualizer is a 9430 code-line application, of which 3304 lines correspond to the visual layout of forms. e-volutionVisualizer is thoroughly developed in Visual Basic 6.0 and consists of 16 forms and 12 modules.

The application was first upgraded from Visual Basic 6.0 to Visual Basic .NET. After the upgrade was completed, the application was enhanced using some of the new features of the .NET platform and programming language.

PART I - MIGRATION

In order to carry out the upgrade process of e-volutionVisualizer, the following tools were required:

- Visual Basic 6.0: to check and modify the original code.
- Visual Studio.NET: to carry out the upgrade process toward Visual Basic .NET and make the required corrections to the migrated code.
- Application's source code: code to be migrated.
- Database: e-volutionVisualizer makes enquiries to a database. A Microsoft database is included in order to run tests.

The Upgrade Process

Revision of the application in Visual Basic 6.0

The Upgrade process starts with an inspection of the original code in Visual Basic 6.0. The purpose of this step is to prepare the code to minimize the number of upgrade issues. Some common improvements you can make to the original application in Visual Basic 6.0 code are: remove implicit object instantiation, cleanup late binding, change the use of variant by the proper variable type and remove references to underlying constants.

The best way to identify the critical points is to run the Upgrade Tool and review the Upgrade Report it generates. To do this pre-conversion, simply open the original VB6.0 application from Visual Studio .NET and the Upgrade Wizard will open automatically.

Freedom classes

For those concepts of Visual Basic 6.0 that do not have an equivalent in Visual Basic .NET, ArtinSoft developed a set of compatibility classes called Freedom® Classes, that help replicate that functionality without having to manually replace the equivalent code.

Correcting compilation errors

The elements in the Task List window show the Visual Basic 6.0 code that could not be automatically upgraded because there is no direct equivalence for those features in the new language.

Data binding: Changing RDO for ADO

Data bindings to a data source DAO or RDO are not supported in VB.NET; Data control and RemoteData control have no equivalent. DAO and RDO can only be used through code.

Drag&Drop

The way to implement Drag&Drop operations has changed in Visual Basic .NET. There is a series of controls in charge of this operation in the original application. To obtain the same functionality in the migrated application, it is required to first activate the Drag&Drop capacity of the controls.

The DoDragDrop method receives two parameters: the first one contains the information on which drag will be applied, and the second one contains the type of drag to be applied.

It must be verified that the type of information in the DragEnter event is correct and in the DragDrop, drop is activated.

Printer Object

To solve the problems concerning the printer, PrinterClass (part of Freedom classes) was used. In the UpgradeSupport.vb a global variable was defined.

After defining these variables, the methods of Visual Basic 6.0 Printer Object can be easily used. This is a huge advantage as it minimizes the number of required changes while accelerating the upgrade process.

Forms Collection

Visual Basic .NET does not provide any support for Forms Collection. To preserve functionality, one of Freedom classes was implemented.

Load

The Load method in Visual Basic 6.0 could be used to load a form without deploying it. Besides, references could be made to any of its properties without the form being visible. The Load method is not supported in Visual Basic .NET. When a form's application is created using New, it is loaded but not deployed until the Show method is called on. To solve this error, the code lines containing Load method were removed. If the form is to be loaded without showing it, New must be used.

ShowWhatsThis

There is no ShowWhatsThis equivalent in Visual Basic .NET. However, having analyzed the project, it was concluded that this was not an essential property in this application. This is due to the fact the programmed functionality can be acquired in Visual Basic 6.0 by simply using the ToolTip object. Thus, to solve this type of compilation errors, the code lines that used it were removed.

To

The reserved word To is used to separate the start and final coordinates of the line to be printed in the references to the Printer.Line function. This To must be replaced with a comma as the Printer Object functionality is obtained from one of Freedom classes. This change must take place in the FrmPreview.vb, in the Grid_Print function and in the PrintManagementGrid.vb form in the Print_Grid.

Execution error prevention: warning messages

We now have an application with no compilation errors. However, before execution, warning messages reported by the upgrade tool, must be carefully checked.

The Upgrade Wizard provides messages warning about possible execution errors. This code does not normally generate compilation errors – however, the application might suffer behavior variations that require a revision of these messages before running the application. This process saves time by distilling the application, as it is possible to correct errors before they emerge.

Generic Namespace Form

When a form goes through as a function's parameter, the upgrade tool generates the base name of the form class (System.Windows.Forms.Form) as a parameter type. By making reference to objects contained in this form, an error is obtained because the generic object does not contain any component. In order to solve this problem the parameter System.Windows.Forms.Form has to be changed to Object to allow these names to be solved by Late binding.

Forms Activated event

The Activate event in Visual basic 6.0 only produced when a change was made between the same application's forms; in Visual Basic .NET this also occurred when an application change took place. The code must be checked in order to preserve the application's behavior. In the migrated e-volutionVisualizer application, the behavior remained intact and no changes were made to this code.

Solving execution errors

Once all the compilation errors have been solved and the upgrade warning messages checked, the process moves on to running the application in Visual Studio .NET's debugger to observe how the application performs and look for execution errors. The following errors were found:

Pixels vs. Twips

The position and size of all the controls are expressed in terms of Píxels, the new predefined graphic unit. The code lines that work with twips must be checked in order to correctly manipulate both units because the application controls the dimension of the objects and the coordinates in twips.

Dynamic creation of a control array

This error is also derived from the warning message generic namespace form.

Functional Equivalence

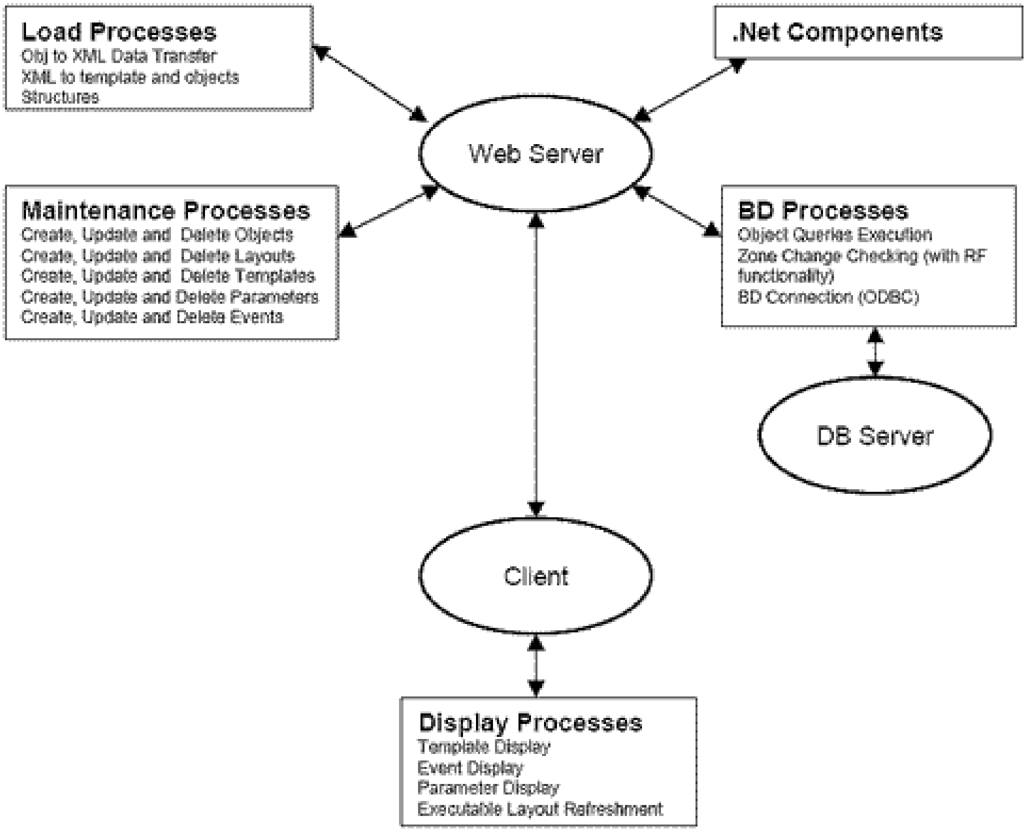
Once we have managed to correct all the detected errors thus far, functional equivalence is achieved. This is to say, the e-volution Visualizer application can now be run in Visual Basic.NET exactly as the original application in Visual Basic 6.0.

We have reached a very important stage in the development of our e-volution project: we have transplanted the functionality from one language to another and we are now ready to take on the next stage fully certain of our product’s stability and functionality.

PART II – APPLICATION ENHANCEMENT

Implemented Design Decisions

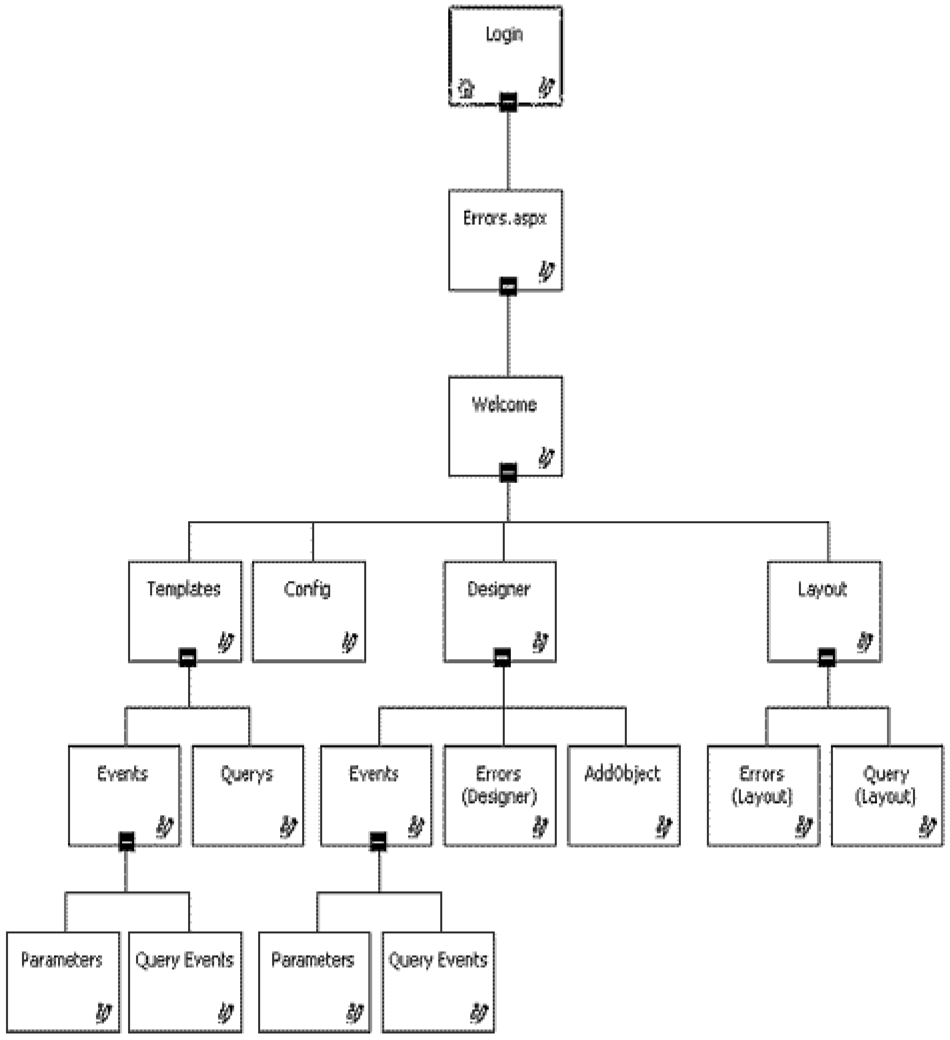
The main re-architecture performed on e-volutionVisualizer was to Web-enable the application. Figure 3, shows the architecture of the enhanced e-volutionVisualizer and the relations between the different processes in the application.


Load Processes
Obj to XML Data Transfer
XML to template and objects Structures
.Net Components
Web Server
Maintenance Processes
Create, Update and Delete Objects
Create, Update and Delete Layouts
Create, Update and Delete Templates
Create, Update and Delete Parameters
Create, Update and Delete Events
BD Processes
Object Queries Execution
Zone Change Checking (with RF functionality)
BD Connection (ODBC)
DB Server
Client
Display Processes
Template Display
Event Display
Parameter Display
Executable Layout Refreshment

Moving Templates from OBJ format files to XML format files

The e-volutionVisualizer Visual Basic 6.0 version uses OBJ files in order to create templates and objects. These files are very important because they save the structure of the objects, queries, images, and map’s logic.

The file structure was complex and not well formed (see Figure 4), so it was decided to implement a function in order to translate these files into XML format.



Moving plain files to XML

The objects files were used by the e-volutionVisualizer Visual Basic 6.0 version in order to define new drawings and edit the existing ones. The object structure was similar to the template structure, so it was necessary to translate their format to XML format.

The original OBJ file was defined with twips. It resolved several migration issues.

The translation process transforms the main file and its sub-levels recursively into XML files, and saves them in the same directory where the original OBJ file is located.

Improvement Description

The following are some of the main improvements achieved after the enhancement of the Visual Basic 6 application to VB .NET:

- OBJ to XML: the files used to store the map's information were converted from a proprietary format to XML format, to take advantage of all XML resources and benefits.
- Web enabled: the application is now accessible from anywhere, through the Internet
- With the Web service the program provides cross-platform interoperability: real world customers may have different platforms in different plants and it becomes necessary to have interoperability between them.
- Several database configuration options
- Improvement in the definition of the data tier: all database access is made on the server side
- Both OBJ files and XML files Format are supported
- Simplified file maintenance, because there is no reason to install new versions of the e-volutionVisualizer application in the client machine since you will always have the latest version through the Web
- Scalability: .NET provides the capability to increase the capacity to handle more visitors and more complex business logic
- Application integration: the same data could be reused in other e-volutionSoft application
- Having distributed Web-based data sources enable costumers to scale to a better architecture (e.g. Web farms)
- Part of the code in charge of the application's calculations and logic (business layer) was isolated from the GUI in order to create the new Web presentation layer

Conclusion

In the case of e-volutionVisualizer, it only took 10 days for a single programmer not only to migrate the entire application from Visual Basic 6.0 to .NET but also to enhance it with some of the new features.

In comparison, it was estimated that re-writing the application would have taken no less than 10 weeks.

The productivity of the migration process until achieving functional equivalence, including completing the conversion and testing of the application is between 30 and 80 times greater than the productivity of a programmer developing the application from scratch.

The possibility of code reuse directly depends on the structure of the original application. In the case of e-volutionVisualizer re-engineering process, the modules built for the DB and file access were totally reused. There are 2761 lines of code reused in the reengineering process, i.e. almost 40% of the original code.

Productivity during the re-architecture process was lower because it is essentially a manual process. However, re-architecture was only required in the critical parts of the application.

Migrating to .NET is a significant undertaking, but many industry analysts and developers who have been studying .NET come to the same conclusion: the longer-term benefits of productivity, capability, and maintenance savings are likely to prove worth the investment.