Comparing VB Migration Partner with Upgrade Wizard

The Upgrade Wizard was a tool that a company developed for Microsoft and that was included in Visual Studio .NET editions, from version 2002 to 2008. Microsoft ceased to distribute the Upgrade Wizard when support for Visual Basic 6 officially ended. Very few developers missed it, because it never gained a great reputation. Worse, it contributed to create a bad reputation for all VB6-to-.NET code translators on the market.

The same company that developed the UW for Microsoft later released a more powerful version of that tool, even though it basically uses the same conversion engine.

When we launched VB Migration Partner we wanted to create a conversion tool that was significantly better than the Upgrade Wizard. We managed to do so by implementing a parser that was specifically written for VB6 and a support library that could fill the functional gap between that language and the .NET Framework. It is therefore interesting to compare VB Migration Partner with the UW ***in terms of compilation errors and warnings***.

In the first part of this document, we illustrate the results you get when running both programs over a number of VB6 projects. In the second part, it describes whether and how both programs solve the most common issues you face when migrating VB6 apps to .NET.

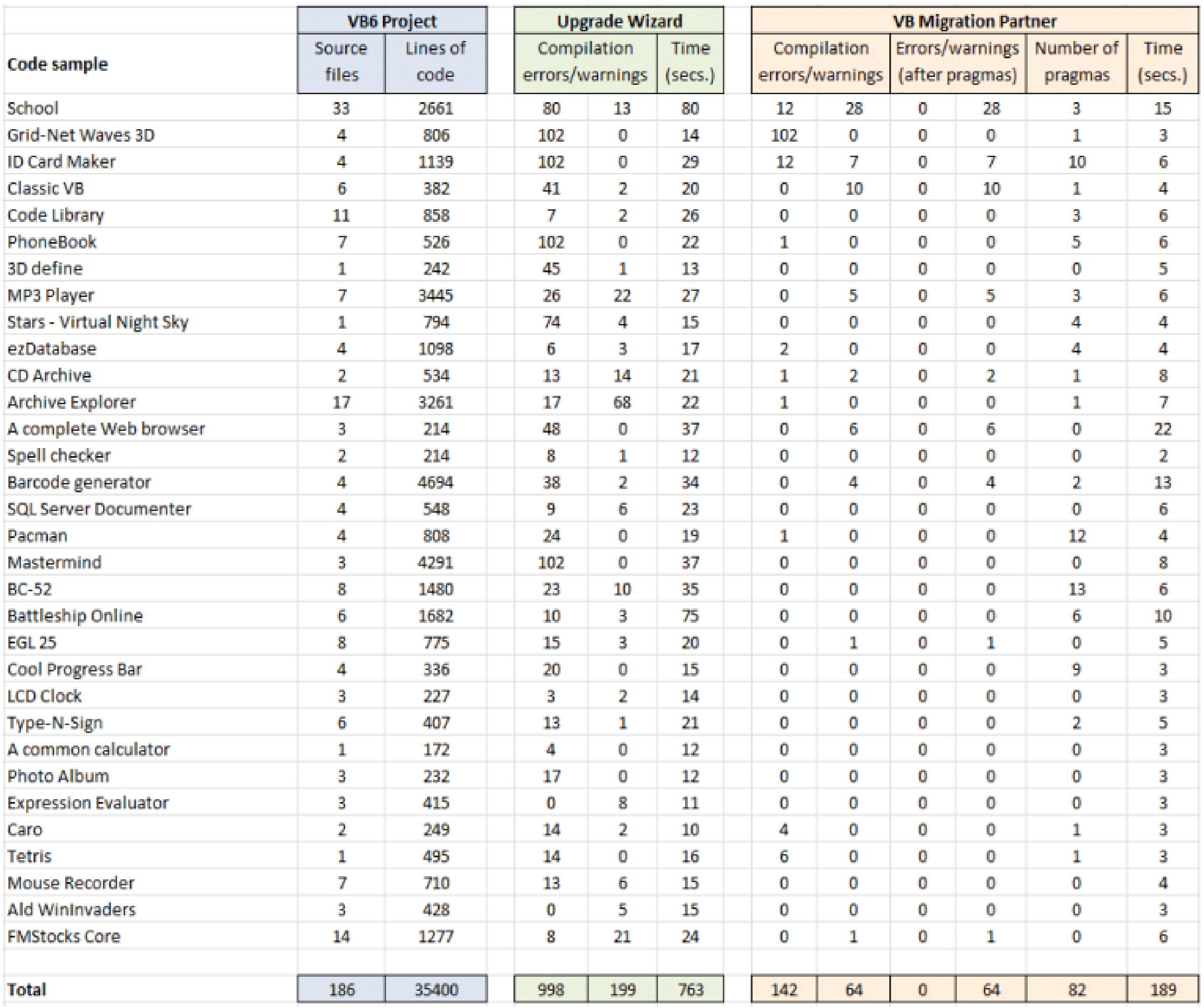
Test #1: Migrate open source VB6 projects

Instead of just providing a series of unverifiable numbers, we run both tools against a group of open source code source which offer a variety of challenges, including rarely-used controls, data-binding, graphic methods, and drag-and-drop.

We didn't edit any executable statement in the original VB6 nor in the converted VB.NET code. In some of these cases, however, we made a second pass through VB Migration Partner after adding all the pragmas that were necessary to reach a fully functional .NET project.

We benchmarked the Upgrade Wizard installed with Microsoft Visual Studio 2005 and release 1.00 of VB Migration Partner, on an Intel Core Duo 7800 @ 2.66GHz running Microsoft Windows Vista 64-bit SP1. ***Note:*** *We could have run these tests again on more recent releases of both the UW and VB Migration Partner and on more powerful computers, but the ration between the two programs' speed and correctness would be substantially identical.*

Comparing VB Migration Partner with Upgrade Wizard



Code sample	VB6 Project		Upgrade Wizard			VB Migration Partner					
	Source files	Lines of code	Compilation errors/warnings		Time (secs.)	Compilation errors/warnings		Errors/warnings (after pragmas)		Number of pragmas	Time (secs.)
School	33	2661	80	13	80	12	28	0	28	3	15
Grid-Net Waves 3D	4	806	102	0	14	102	0	0	0	1	3
ID Card Maker	4	1139	102	0	29	12	7	0	7	10	6
Classic VB	6	382	41	2	20	0	10	0	10	1	4
Code Library	11	858	7	2	26	0	0	0	0	3	6
PhoneBook	7	526	102	0	22	1	0	0	0	5	6
3D define	1	242	45	1	13	0	0	0	0	0	5
MP3 Player	7	3445	26	22	27	0	5	0	5	3	6
Stars - Virtual Night Sky	1	794	74	4	15	0	0	0	0	4	4
ezDatabase	4	1098	6	3	17	2	0	0	0	4	4
CD Archive	2	534	13	14	21	1	2	0	2	1	8
Archive Explorer	17	3261	17	68	22	1	0	0	0	1	7
A complete Web browser	3	214	48	0	37	0	6	0	6	0	22
Spell checker	2	214	8	1	12	0	0	0	0	0	2
Barcode generator	4	4694	38	2	34	0	4	0	4	2	13
SQL Server Documenter	4	548	9	6	23	0	0	0	0	0	6
Pacman	4	808	24	0	19	1	0	0	0	12	4
Mastermind	3	4291	102	0	37	0	0	0	0	0	8
BC-52	8	1480	23	10	35	0	0	0	0	13	6
Battleship Online	6	1682	10	3	75	0	0	0	0	6	10
EGL 25	8	775	15	3	20	0	1	0	1	0	5
Cool Progress Bar	4	336	20	0	15	0	0	0	0	9	3
LCD Clock	3	227	3	2	14	0	0	0	0	0	3
Type-N-Sign	6	407	13	1	21	0	0	0	0	2	5
A common calculator	1	172	4	0	12	0	0	0	0	0	3
Photo Album	3	232	17	0	12	0	0	0	0	0	3
Expression Evaluator	3	415	0	8	11	0	0	0	0	0	3
Caro	2	249	14	2	10	4	0	0	0	1	3
Tetris	1	495	14	0	16	6	0	0	0	1	3
Mouse Recorder	7	710	13	6	15	0	0	0	0	0	4
Ald WinInvaders	3	428	0	5	15	0	0	0	0	0	3
FMStocks Core	14	1277	8	21	24	0	1	0	1	0	6
Total	186	35400	998	199	763	142	64	0	64	82	189

A summary of the results:

- VB Migration Partner generates nearly **5x fewer compilation errors** than the Upgrade Wizard before adding a single pragma. In four cases, Upgrade Wizards generates just too many compile errors for Visual Studio to display (VS2005 couldn't display more than 102 compilation errors), therefore the **actual ratio is even higher**.

- In most cases, VB Migration Partner allowed us to get rid of all these errors with just **one or two pragmas**.
- Considering all the pragmas used in all samples delivers we get an average of one pragma every 430 lines of code; in larger applications the ratio is even more convenient, because a single pragma can affect all the files and statements in the project.
- The UW processes about 46 LOCs per second on the average, VB Migration Partner runs at 187 LOCs per second, i.e. **4 times faster** while performing many additional tasks, for example the creation of code statistics and reports. When fed larger applications, we've seen that VB Migration Partner consistently performs 7-8 times faster than the Upgrade Wizard.

Interestingly, most of the compilation errors you get from VB Migration Partner come from the first two code samples – School and Grid-Net Waves 3D. If you exclude them from the stats, it turns out that VB Migration Partner averages at **one compilation error every 1140 LOCs** before adding a single pragma, and is therefore **about 30 times better than the Upgrade Wizard** (which delivers one compilation error every 40 LOCs). On large, real-world VB6 projects we've seen that the actual ratio is between 8x and 12x.

It's important to bear in mind that compilation errors and warnings tell only a part of the story, because even a VB.NET project with zero compilation errors might raise one or more runtime errors. In other words, the samples that show zero compilation errors after the migration with the Upgrade Wizard might require a lot of additional work to run correctly. By comparison, the column labelled as *Number of pragmas* indicates how many pragmas were needed to have a **fully functional** VB.NET application that raises no runtime errors.

Along the same lines, the number of pragmas that are necessary to reach a fully functional VB.NET application tend to decrease - in percentage over the total number of executable lines – when the VB6 application gets larger, because often one single pragma scoped at the project-level can solve all similar occurrences of a given compile or runtime error.

Test #2: Aivosto's compatibility checklist

Another way to compare VB Migration Partner with Upgrade Wizard is checking which migration issues either software can solve automatically.

VB Project Analyzer is a popular tool available on Aivosto's web site. It performs code analysis, dead code detection and removal, coding and naming rule enforcement. It can find common programming errors (including memory leaks caused by undisposed API handles), can optimize your code much faster than the fastest and smartest developer, and can generate a thorough documentation of all classes, forms, and members (including cross-reference data to detect who call whom). Best of all, it works with VB6, VB.NET, and VBA.

If you are preparing your VB6 apps for migration to .NET, Aivosto's VB Project Analyzer is also very helpful, because it can automatically spot most VB6 language elements that the Update Wizard doesn't convert correctly to VB.NET. The online help includes the list of all compatibility checks that VB Project Analyzer performs. Please refer to the original list for an explanation of each compatibility issue.

NOTEs

Comparing VB Migration Partner with Upgrade Wizard

VBMP stands for VB Migration Partner, **UW** stands for Upgrade Wizard

✓ means that a given feature is supported by VB Migration Partner (VBMP)

✓ * means that the feature appears in Aivosto compatibility checklist (created in 2005) but is supported by Upgrade Wizard 2008 (UW)

✓ **P** means that VBMP supports the feature only partially: for example, it doesn't cause a compilation error yet it doesn't ensure functional equivalence at runtime.

Add-in model changed in VB.NET		VBMP is unable to migrate add-ins because the IDE object model is too different, but it emits a warning.
ADO required for data binding in VB.NET	✓	VBMP supports DAO, RDO and ADO data-binding, including binding with DataEnvironment objects, ADO data source classes, and simple-bound data consumer classes.
Array must start at 0 in VB.NET	✓	VBMP supports several strategies for migrating arrays with non-zero LBound. Not only does it fix the array declaration, it can even modify the index used to reference individual array elements.
As Any not allowed in VB.NET	✓	VBMP correctly converts As Any arguments by producing one or more overloads of the Declare statement.
As New doesn't auto-instantiate if object released in VB.NET	✓	VBMP optionally supports the lazy-instancing feature of As New variables.

As New unsupported for arrays in VB.NET	✓	VBMP can correctly translate As New arrays by preserving the VB6 semantics.
ByRef property params unsupported by VB.	✓	VBMP converts ByRef arguments inside properties into ByVal arguments because VB.NET requires it; however it emits a warning if the argument appears to be modified inside the property procedure – or is passed to another method that can modify it.
ByVal/ByRef not allowed in API calls in VB.NET	✓	VBMP safely resolves ByVal and ByRef in calls to API methods.
Circle and Oval unsupported by VB.NET	✓ *	VBMP correctly converts Line and Shape controls, and even translates graphic methods such as Line, Circle, PSet, PaintPicture, etc.
Class Instancing changes in VB.NET	✓	VBMP deals with SingleUse objects as if they were MultiUse, because .NET has no notion of “single use” objects. Global objects are converted correctly.
COM module methods not callable from VB.NET		VBMP can’t handle COM module methods.
COM+/MTS not upgradable to VB.NET	✓	VBMP correctly converts all frequently used MTS/COM+ features into the corresponding .NET features.
Conditional block will not upgrade to VB.NET		VBMP doesn’t migrate code inside an #IF block whose condition is false.
Control unsupported by VB.NET	✓	VBMP converts 60+ controls, including all those included in the VB6 toolbox with the only exception of OLE Container. It supports other

		commonly used controls such as WebBrowser and ScriptControl, and all the windowless controls in the MSWLESS library.
DDE unsupported by VB.NET	✓	VBMP supports DDE communications, among migrated VB.NET apps.
Diagonal line unsupported by VB.NET	✓ *	VBMP supports the Line control, with any inclination and style.
DoEvents() returns no value in VB.NET	✓	VBMP provides a DoEvents6 replacement statement that returns the number of open forms.
Drag-and-drop requires rewrite for VB.NET	✓	VBMP fully supports OLE drag-and-drop, in both the manual and automatic flavors. Starting with version 1.20, VBMP version also supports "classic" (non-OLE) drag-and-drop.
Event behavior changes in VB.NET	✓	VBMP fully supports all these (and other) events, no work is required after update.
Event log model differs in VB.NET	✓	VBMP supports all the Event Log-related properties and methods.
Initialized arrays in UDTs unsupported by VB.NET	✓	VBMP correctly initializes UDTs containing arrays, fixed-length strings, and auto-instancing (As New) object variables. It even generates special code to correctly convert assignments between UDTs that contain these members. (UW doesn't even emit a warning in that case.)
MDIForm event unsupported in VB.NET	✓	VBMP correctly handles mouse-related events inside MDI forms.

Member cannot be default in VB.NET	✓	VBMP offers the same degree of support that UW does. In addition, VBMP can convert a default method with parameters into a VB.NET ReadOnly Property and then mark it as the default member of that class.
Module not upgradable to VB.NET	✓ P	VBMP doesn’t migrate DHTML and WebClass components. However, it converts UserDocument and PropertyPages into VB.NET UserControls, thus you have something to work with after the migration even though you'll need additional manual coding to have it work as expected.
No control arrays in VB.NET	✓	VBMP correctly converts all sorts of VB6 control arrays, including arrays of menus and 3rd-party ActiveX controls.
Old VB project not upgradable to VB.NET		VBMP has the same limitation as UW and requires that you migrate VB3, VB4, and VB5 projects to VB6 before attempting the migration to VB.NET.
OLE Automation unavailable in VB.NET	✓ P	VBMP converts OLE Automation features into do-nothing members that are marked as obsolete. Calling these members has no effect or throws an exception, but at least you can start testing other portions of the application without having to fix one or more compilation errors.
ParamArray is ByVal in VB.NET	✓	VBMP can automatically generate code that ensures that ParamArray use by-reference semantics.

Parameterless default properties unsupported in VB.NET	✓	VBMP behaves like UW when the object variable is typed; when the variable uses late binding, VBMP can generate code that determines the default member at runtime.
Property mixes scopes	✓ *	VBMP correctly converts property procedures with mixed scope.
Property passed ByRef	✓	VBMP can convert ByRef parameters into ByVal parameters if the parameter isn't assigned inside the method.
Resource file requires work in VB.NET	✓	VBMP converts both the resource file and all LoadRes* methods; it even converts them to My.Resources members if possible.
ScaleMode must be vbTwips for VB.NET	✓	VBMP supports all ScaleMode settings, including 0-vbUser.
Setting .Interval does not enable/disable timer in VB.NET	✓	Projects converted by VBMP don't suffer from this issue.
String byte functions unavailable in VB.NET	✓	VBMP partially support byte-oriented string functions, such as LenB or InStrB; it also support implicit conversion between strings and byte arrays, and conversions between ANSI and Unicode strings.
Sub Main not executed in VB.NET	✓	VBMP fixes this issue: a VB.NET class library project that is the result of converting a VB6 ActiveX DLL project correctly executes the Sub Main method before any class in the library is instantiated.
Sub Main	✓	.NET program exits at End Sub: VBMP can handle this issue by adding a proper pragma.

TTF/OTF fonts required by VB.NET	✓	VBMP allows you to determine how fonts are converted during the migration to VB.NET.
Type unsupported by VB.NET	✓	VBMP comes with a VB6FixedString type that perfectly mimics the VB6 fixed-length string type; also, VBMP can convert arrays of fixed-length strings; additionally, fixed-length strings in UDTs can be converted into standard strings and still retain the fixed-length behavior
Unavailable in VB.NET	✓	VBMP supports CVErr, GoSub, Return, vbDataObject, vbUnicode, vbFromUnicode, IsEmpty, and a limited form of LSet that works with UDTs. (It doesn't support VarPtr, ObjPtr, and StrPtr undocumented functions, though.)
Underscore _names not hidden in VB.	✓	VBMP doesn't deal names with a leading underscore in a special way, however it recognizes VB6's hidden members and convert them correctly to VB.NET.
VB5 project may not upgrade to VB.NET	✓ P	VBMP converts only VB6 projects, therefore VB5 projects must be converted to VB6 first. However, it supports the Common Windows controls released with VB5.
WebClasses upgrade to ASP.NET		VBMP doesn't convert WebClass projects. We strongly believe that such projects should be upgraded to ASP.NET in all cases.
Function without type specification	✓	VBMP emits a warning if a function or property has no As clause; you can use the SetType pragma to define the type returned by the VB.NET function without altering the VB6 code.

Variable without type specification	✓	VBMP emits a warning if a variable or parameter has no As clause; however, you can use the SetType pragma to define the type of the VB.NET variable without altering the VB6 code.
ByVal/ByRef missing	✓	By default, VBMP doesn't take any action if an explicit ByRef/ByVal keyword is missing and converts these parameters as ByRef parameters; however, it emits a warning if a ByRef parameter can be safely converted as a ByVal parameter and you can use the UseByVal pragma to automatically convert such parameters into ByVal parameters.
Option Explicit missing	✓	VBMP doesn't take any specific action if Option Explicit is missing; however, you can use a pragma to automatically create VB.NET variables for all VB6 variables that weren't explicitly declared.
Optional parameter missing default value	✓	VBMP automatically add the property default value for Optional parameters if necessary.
Variable/Parameter with generic type	✓	VBMP provides statistics about Variant variables and allows you to use the SetType pragma to change the type of a variable or parameter during the conversion to VB.NET, without affecting the existing VB6 code.

To recap, VB Migration Partner can fully or partially handle 44 of the 49 compatibility issues that are left unresolved by the Upgrade Wizard. The remaining 5 unresolved issues are related to features that just don't make sense under VB.NET - such as OLE-related properties, the IDE extensibility object model, and WebClasses components.

Even more important, **VB Migration Partner fixes many more compatibility issues than those listed in this page**. As a matter of fact, VBMP solves many problems that even VB Project Analyzer fails to detect. (Read here and here for a more exhaustive list of migration problems.)