17 Reasons for Using a Support Library in Migration Scenarios

Unlike the majority of the language conversion products available on the market – which mainly attempt to convert code from one language to another – VB Migration Partner both converts the code *and* uses a support library to decrease the gap between VB6 and VB.NET and to ensure that the converted code truly behaves as expected.

This white paper illustrates why we took this approach and the many benefits a support library for migrated VB.NET applications can provide. It is meant to be used by both developers and IT managers, regardless of their familiarity with .NET Framework programming. In some cases, we couldn't help showing short code snippets to prove a point, but you don't need to understand what each line does to follow the line of reasoning.

Summary

In this article we will analyse each topic in a very detailed manner, with code samples and technical considerations. This is a short summary of what you can find:

1. Language impedance: a support library can fill the gap between VB6 and VB.NET and support more keywords than by any other means.
2. User interface impedance: only a control library can perfectly mimic all the VB6 controls, their object models, and their features.
3. Integration with the code generation engine: when coupled with a very sophisticated code generation engine, the support library can solve most of the issues that must be manually fixed if working with other, conversion software that use a more traditional approach.
4. Preliminary work on the VB6 code: a support library can transparently support more VB6 features and can therefore eliminate (or significantly reduce) the need to prepare the VB6 code for migration, which in turn permits to keep the VB6 and VB.NET code in sync.
5. Compilation errors: a support library exposes classes and methods with a VB6-like syntax, which help to reduce the number of compilation errors in the VB.NET code produced by the migration software.
6. Runtime errors: by mimicking the VB6 behavior as closely as possible, a support library greatly reduces the efforts required to reach functional equivalence.

17 Reasons for Using a Support Library in Migration Scenarios

7. Concise, readable, and maintainable code: a migration software that uses a support library doesn't require to generate additional code to work around the differences between VB6 and.NET object models.
8. Application advancement: a support library is by no means an obstacle to improving and extending the VB.NET application after the migration process has been completed.
9. Performance optimizations: VB Migration Partner's library exposes objects that can improve performance with very little effort on your part.
10. Consulting and training costs: a support library makes VB.NET and C# languages more familiar to VB6 developers and reduces the urge for specific .NET training during the migration process.
11. Embedded debug features: VB Migration Partner's library includes special functions that make tracing and debugging easier.
12. Interoperability with VB6: by reproducing VB6 behavior as closely as possible, a support library allows you to reuse existing data files and/or exchange data with VB6 apps you don't want to migrate yet.
13. Bug fixing: if you use a traditional conversion tool to migrate to VB.NET and the resulting application that has bugs (what software hasn't?), all the bug fixing activity is on you. If you deploy a migrated application with bugs that depend on our support library, Code Architects takes care of them.
14. Runtime royalties: registered users can freely redistribute VB Migration Partner' library with converted VB.NET apps, with no runtime royalties whatsoever.
15. Future upgrades: you can freely download future versions of our support library, with no additional or hidden costs.
16. Source code: we can license the library's source code.
17. After-migration library improvements: VB Migration Partner users can benefit from future extensions to its support library.

Conclusions

1. Language impedance

17 Reasons for Using a Support Library in Migration Scenarios

Writing a VB6 to VB.NET code converter isn't for the faint of heart, but isn't overly complex either, because the two languages are very similar, or at least VB.NET offers a high degree of backward compatibility with its predecessor. The truth is, however, that VB.NET is more akin to C# and other .NET languages than to VB6. A support library is the best option to fill the gap.

We would like to claim that Code Architects invented the concept of support libraries to reduce language impedance, but of course we didn't. In fact, VB.NET comes with two support libraries: Microsoft.VisualBasic.dll and Microsoft.VisualBasic.Compatibility.dll. **Microsoft.VisualBasic.dll** aims at making the VB.NET experience more pleasant for VB6 developers (who can find more familiar names such as MsgBox and Hex), whereas **Microsoft.VisualBasic.Compatibility.dll** provides additional support for .NET application, including support for controls such as DriveListBox and FileListBox, or for VB6 features that are missing in the .NET Framework, such as control arrays.

The main problem with these libraries is that they weren't built with background compatibility in mind. As a matter of fact, in most cases they only vaguely approximate the original VB6 behavior.

Take the Get# and Put# file methods, for example. The Upgrade Wizard (included in Visual Studio 2010 and previous versions) and similar tools map these methods to the FileGet and FilePut methods in Microsoft.VisualBasic.dll. Unfortunately these methods are far from being equivalent to the original VB6 methods: they don't work well with UDTs and with values stored inside Object variables; they don't behave in a consistent way if the file is opened for binary or random access; they have problems with arrays and fixed-length strings, and so forth. In short, you can't trust the file methods in Microsoft.VisualBasic.dll for a robust migration, except in very simple cases.

While developing VB Migration Partner, we quickly realized that the file methods in the "official" Microsoft libraries were so crippled as to be nearly useless. This is why VB Migration Partner maps the Get# and Put# keywords to the FileGet6 and FilePut6 methods in our support library. We put a lot of work in these methods to ensure that they work as their VB6 counterparts as closely as possible.

File methods are just one single example. To get a more complete picture, just browse the Resource section of this website. VB Migration Partner solves all these issues by combining a smart code generator with a comprehensive support library.

Interestingly, all the conversion tools we are aware of also use a support library to reduce the gap between VB6 and VB.NET. Only, their library isn't as complete and powerful as ours and can only solve a small subset of the immense set of differences between the two languages.

2. User interface impedance

As serious as it may sound, language impedance accounts for less than 10-15% of the problems you face when migrating a UI-intensive VB6 application, in which case the real obstacle in reaching functional equivalence is the set of differences between VB6 and VB.NET controls. The number of these differences is so high that we can't list all of them here (we are documenting them in the Resources section of this website). Here are just a few:

Missing controls: a few VB6 controls have no .NET counterparts, including ADO/DAO/RDO data controls, DataList, DataCombo, Animation, ImageCombo, UpDown, PictureClip, and SysInfo. A migration software that doesn't use a support library can't do much other than reusing the ActiveX control on the .NET form, which results in a less-than-robust application.

Missing features: even if a VB6 control has a .NET equivalent, many features aren't available in the .NET world. For example, the VB.NET PictureBox can't work as a container and the ListImage object doesn't support anything similar to the Draw or the ExtractIcon methods, just to name a few common problems.

Graphics: You have to carefully remap all your Line, Circle, PSet, Point, and PaintPicture commands to GDI+ objects. Also, if you used GDI methods for advanced graphics, you'll soon discover that .NET controls don't have an hDC property that returns the handle to their device context.

Drag-and-drop: The .NET way to implement drag-and-drop is so different from the VB6 way that no migration tool even attempts to convert existing D&D code, hoping that you are skilled enough in both VB6 and .NET to do the conversion yourself.

Incompatible object models: even if a .NET control provides the same features as the original VB6 control, in some cases a 100% correct translation is impossible because the two controls have

significantly different object models. The most obvious case is the VB6 CommonDialog control, which maps to as many as six .NET controls, some of which are incompatible with the original control.

To get an idea of how incompatible object models can make migration more difficult, try the following experiment. Drop a Toolbar and an ImageList control on a VB6 form, bind the Toolbar to the ImageList control, and then write this code to add a few items to the Toolbar when the form loads:

```
Private Sub Command1_Click()
    ' add a plain button
    Me.Toolbar1.Buttons.Add , "New", "New file", tbrStandard
    ' add a button menu
    Dim btn As Button
    Set btn = Me.Toolbar1.Buttons.Add(, "first", "a menu", tbrDropdown)
    btn.ButtonMenus.Add , "Open", "Open file"
    btn.ButtonMenus.Add , "Save", "Save file"
End Sub

Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
    ' react to clicks on the toolbar
    MsgBox Button.Caption & " has been clicked"
End Sub

Private Sub Toolbar1_ButtonMenuClick(ByVal ButtonMenu As MSComctlLib.ButtonMenu)
    ' react to clicks on the button menu
    MsgBox ButtonMenu.Text & " menu has been selected"
End Sub
```

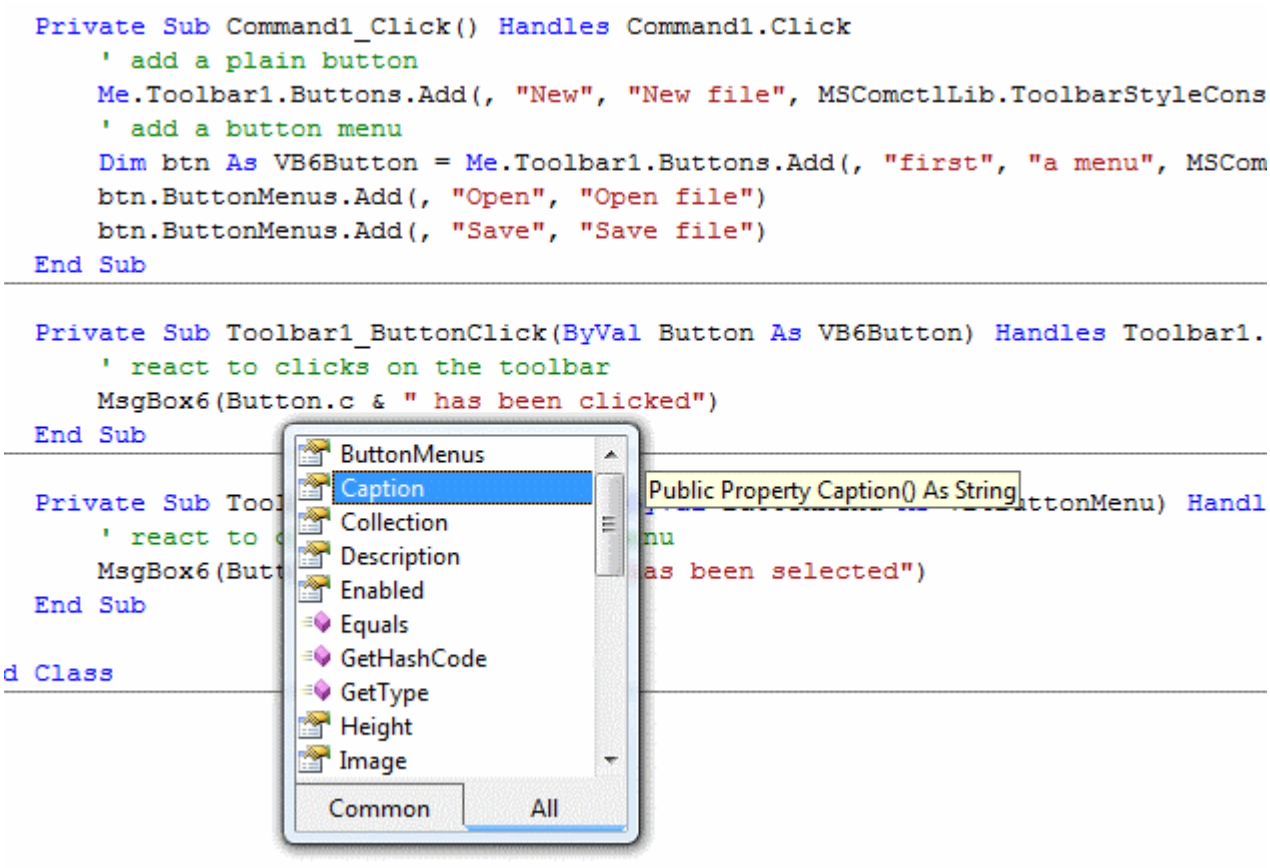
This is the simplest code you can build with a Toolbar, yet it delivers as many as 12 compilation errors and 4 upgrade warnings if converted with the Upgrade Wizard that comes with previous versions of Visual Studio. It takes a while to get rid of all these compilation errors but, above all, you have to completely re-write all the VB.NET code, because .NET Buttons.Add method works differently and the ButtonMenus collection doesn't even exist. You also have to manually rewire the Click events to the right .NET object.

A question arises quite naturally: why should you use a migration software if you then need to manually fix all the code manually? As a matter of fact, re-creating a working .NET form by hand takes less time, in this and many other cases.

You can expect that a more sophisticated (and expensive) migration software can handle this simple case more efficiently, but no tool that relies exclusively on code transformation can avoid all

compilation and runtime errors in all cases. Keep in mind that a remarkable amount of manual work is always required.

By comparison, thanks to its support library approach, **VB Migration Partner can handle virtually all the differences between VB6 and .NET** related to controls and their object model. This feature alone can save you weeks or months in a complex migration initiative. (...and yes, the above code compiles and runs fine at the first attempt, if you use VB Migration Partner.)



3. Integration with the code generation engine

You might believe that VB Migration Partner can achieve its higher success ratio only thanks to its support library, but this is not correct. **VB Migration Partner is above all a very sophisticated code translator** that solves most of the common migration issues by means of advanced code generation techniques exclusively. Consider the following migration issues:

- Array with nonzero LBound index
- Preservation of auto-instancing (As New) semantics
- Refactoring of Gosubs calls into distinct methods
- Conversion of On...Goto and On Gosub methods
- Automatic initialization of Type...End Type blocks that contain arrays, fixed-length strings, and As New objects
- As Any parameters and AddressOf keyword used with Declare statements
- Correct cleanup of recordsets, database connections, and other IDisposable objects
- Conversion of On Error statements into Try-Catch blocks
- Type inference to detect the best data type for Variant/Object variables
- Merging of nested IF statements, using short-circuiting and the AndAlso or && operators
- Refactoring of ByRef arguments into ByVal arguments if possible, or "out" parameters when converting to C#.

We can solve all these migration issues thanks to our sophisticated code generation engine, with no aid from our support library. As of this writing, **our competitors don't support most of the features in this list**.

4. Preliminary work on the VB6 code

All migration tools we are aware of – including Upgrade Wizard and its derivatives – require that you "fix" the VB6 code before attempting the migration. This is necessary to get rid of those language constructs that the tool doesn't know how to migrate, for example Gosub, On Goto/Gosub, As Any parameters, arrays with nonzero LBound.

As you can imagine, preparing the VB6 code for migration is easier said than done. Changing the bounds of an array involves more than just editing a Dim statement, because you also have to check all the places where the array is used, where the array is passed to other methods, where the array is written to disk, etc. Just as important, you must run all your QA tests to ensure that your edits didn't introduce subtle bugs in the VB6 code. This task alone can take one or two weeks in a large VB6 application.

VB Migration Partner eliminates – or dramatically reduces, even in the worst cases – the need for this preliminary step thanks to its code generator engine and its approach based on the support library. Even better, you can enhance the original VB6 code by adding **migration pragmas**. Being just special remarks, pragmas are guaranteed not to change the way the VB6 code works, thus you can continue to distribute and evolve the VB6 code with confidence, because remarks don't change its behavior. No expensive and time consuming QA tests are required.

5. Compilation errors

In addition to providing features that couldn't be implemented otherwise, a support library offers the benefit to reaching the "zero compilation error" sooner than by any other means. Our support library provides "do-nothing" stub methods that have been added only to make the compiler happy. For example, like all other tools in this area, VB Migration Partner doesn't support a few properties of the App object, such as OleRequestPendingMsgText or OleServerBusyTimeout. However, these members have been implemented in the library and don't cause a compilation error. They do cause a compilation warning, though, which allow you to quickly analyze all those occurrences while being able to launch the migrated project.

When the application is up and running, you can set the VB6Config.ThrowOnUnsupportedMembers global flag to True. By doing so, all calls to stub methods now throw an exception and can't go unnoticed, to ensure that you will get rid of all these calls before releasing the product to your customers.

6. Runtime errors

Interestingly, the effectiveness of a code migration software is usually measured in terms of how many compilation errors can be found in the generated code, as if reaching the "*zero compilation errors*" stage was the main goal of a migration initiative. If you browse our competitors' literature, a

relatively little space is devoted to the time and effort required to move from there to the "*zero runtime errors*" stage, which is clearly what really matters.

We can demonstrate that **VB Migration Partner converts to VB.NET with at least 5x fewer compilation errors than the Upgrade Wizard** on the average, even before adding migration pragmas. We ran our tests on a mix of open source VB6 projects that represents a good mix of the challenges you face in migrating a business app (database access, data-binding, lots of controls, graphics, drag-and-drop, etc.).

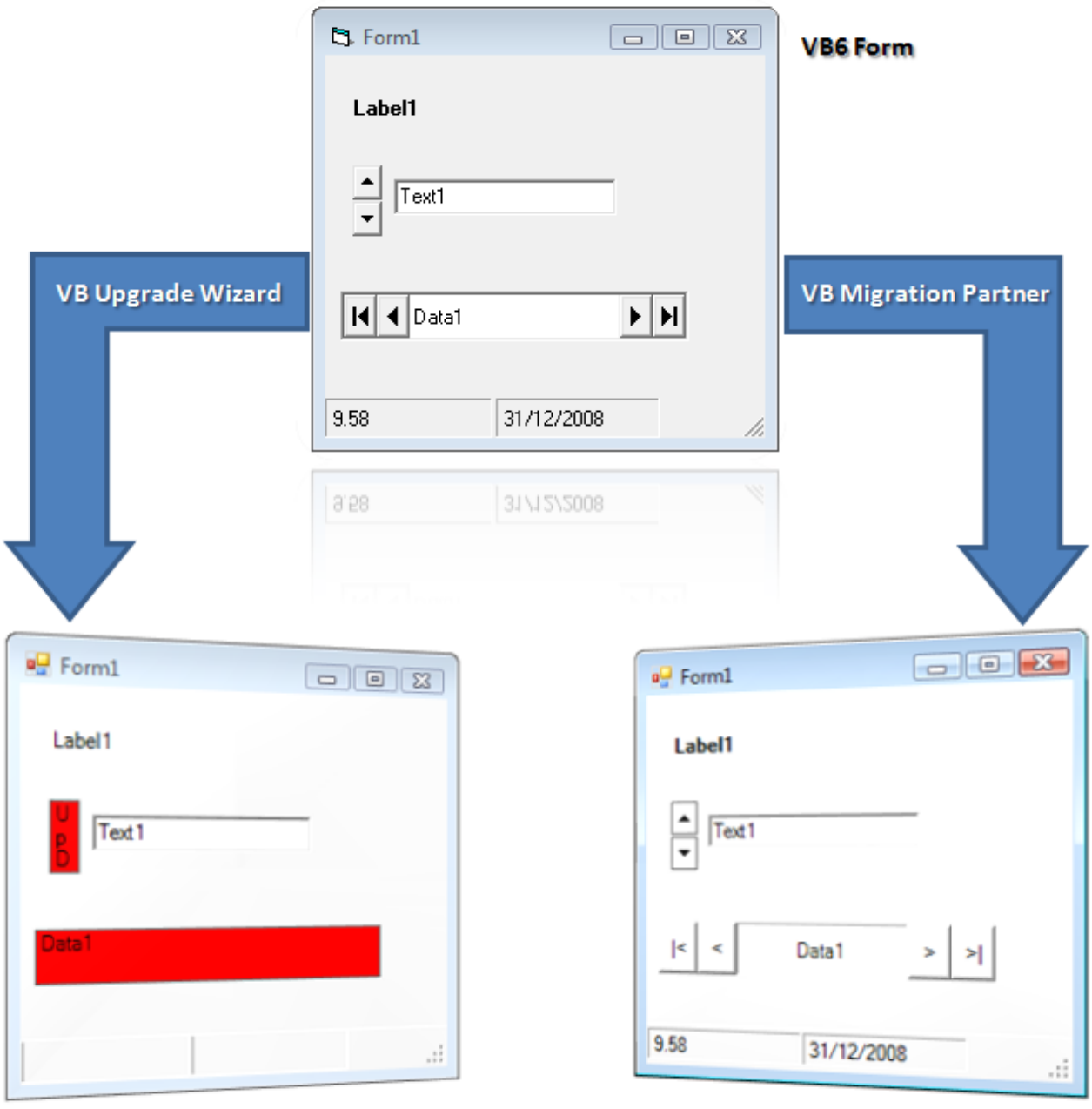
Avoiding compilation errors is only the first problem to solve in a migration project, and is also the simplest one. After all, you can fix most compilation errors by just looking at the messages in Visual Studio's Error List window and using commands such search-and-replace and code refactoring. The only important goal is delivering a .NET application that runs without errors and that is 100% functionally equivalent to the original VB6 project.

Unfortunately, reaching functional equivalence with the original VB6 code is the most complex step in a migration project. In our experience, when using a traditional code converter such as Upgrade Wizard the effort required to get rid of all runtime errors and reach true functional equivalence is at least 2-3 times more expensive than just fixing compilation errors. In many real-world cases, it can be up to 10x more expensive.

A support library can significantly reduce the time needed to achieve functional equivalence. This is the most important benefit of a support library.

Thanks to our support library, **VB Migration Partner generates VB.NET code that behaves exactly like the original application**, or at least as closely as possible. Our support library accounts for virtually 99% of the behavioral differences between the two languages. In fact, it's easier for us to document the minor differences that we ***do not*** support than the other way around. For example, read the KB articles that explain the few differences between VB6 and VB.NET versions of the StatusBar control, the Toolbar control, and the UpDown control. Not bad if you consider that other conversion tools don't support any of the styles of the StatusBar (date, time, capslock, etc.) and doesn't even attempt to convert the UpDown control (see image below) .

17 Reasons for Using a Support Library in Migration Scenarios



7. Concise, readable, and maintainable code

Some developers claim support libraries is that they make it more difficult to maintain the generated .NET code, because it is less readable than it should. To break this false myth, let's consider a very simple VB6 code fragment:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    ' convert the pressed key to uppercase, but ignore spaces
    If KeyAscii = 32 Then KeyAscii = 0: Exit Sub
    KeyAscii = Asc(Chr$(KeyAscii))
End Sub
```

Here is the code converted by VB Migration Partner. Thanks to the library approach, the syntax and semantics of the KeyPress event is preserved. The generated VB.NET code is as concise, readable, and maintainable as the original method:

```
Private Sub Text1_KeyPress(ByRef KeyAscii As Short) Handles Text1.KeyPress
    ' convert the pressed key to uppercase, but ignore spaces
    If KeyAscii = 32 Then KeyAscii = 0: Exit Sub
    KeyAscii = Asc(Chr(KeyAscii))
End Sub
```

This is the code produced by the Upgrade Wizard and other tools based on the same conversion engine:

```
Private Sub Combo1_KeyPress(ByVal eventSender As System.Object, _
        ByVal eventArgs As KeyPressEventArgs) Handles Combo1.KeyPress
    Dim KeyAscii As Short = Asc(eventArgs.KeyChar)
    If KeyAscii = 32 Then KeyAscii = 0 : GoTo EventExitSub
    KeyAscii = Asc(Chr(KeyAscii))
EventExitSub:
    eventArgs.KeyChar = Chr(KeyAscii)
    If KeyAscii = 0 Then
        eventArgs.Handled = True
    End If
End Sub
```

It's hard to consider this code as concise, readable, or maintainable – unless, of course, you consider a 2x increase in size as an example of "conciseness" and you think of the GoTo keyword as "readable and maintainable." ☺

For more examples of "conciseness" and "readability", just try the Upgrade Wizard and its siblings with other common event handlers, such as a MouseDown or MouseMove.

8. Application advancement

Typically, you migrate a VB6 app to .NET because you want to later extend it with new features.

The marketing literature from one of our competitors claims a support library makes future extensions difficult if not impossible, because the .NET application is tied to a library over which you have no control.

Let's see what the reality is and why .NET applications generated by VB Migration Partner are in no way limited in future expansions:

- You can add standard .NET forms to migrated applications, or standard .NET controls to migrated forms
- The controls in our library inherit from the corresponding .NET controls, thus you can easily access all the features of the .NET Framework, including the most powerful properties, methods, and events.
- If you later want to add features that are missing in the original VB6 app – say, drag-and-drop or data-binding – you can implement them using the standard .NET syntax. The controls in the library are full-fledged .NET controls and can behave as such.
- You can easily revert to standard .NET objects, if you don't like using methods in our library.

The bottom line is: **a support library doesn't hamper future application advancement in any way**. Read the "Interoperability with VB6" section for an example of this concept.

9. Performance optimizations

In some cases, a support library can offer you dramatic performance improvement with very little work on your part. For example, many .NET developers know that it's often possible to optimize a piece of code that uses string concatenation, by replacing the string variable with a StringBuilder object, as in this piece of VB.NET code that extracts data from an ADODB recordset:

```
Dim text As String = ""
Do Until rs.EOF
    text = text & rs("FirstName").Value & " "
    text = text & rs("LastName").Value & ControlChars.CrLf
    rs.MoveNext()
Loop
text = UCase(text)
```

Using a StringBuilder object instead of a plain String for the text variable requires major changes in the generated code, because you have to replace the & operator with the Append method and use the ToString method any time you pass the StringBuilder to a method that expects a string.

To make life easier for developers, VB Migration Partner comes with a special class named **StringBuilder6**, which is basically an enhanced StringBuilder class that supports the & operator and implicit conversions to/from the String type. In fact, you can optimize the above piece of VB.NET code by modifying just ***one line*** of code:

```
Dim text As StringBuilder6 = ""
```

The speed improvement can astonishing, even 100x or more.

VB Migration Partner's code analyser is enough sophisticated as to automatically suggest where using a StringBuilder6 object can positively affect performance. Our software offers similar speed improvements in other fields as well, for example when working with collections.

10. Consulting and training costs

Developers working at a migration project should be adequately skilled in both VB6 and .NET. If most developers in your company aren't very familiar with .NET these are your options:

a. you hire one or more .NET skilled developers
b. you hire a consulting company that specializes on .NET
c. you train existing VB6 developers to use VB.NET instead
d. you do nothing of the above, and just hope for the best

All these options are expensive. Even the last option is going to cost you a lot, because less skilled developers typically need more time to complete the migration to .NET, which indirectly increases the cost of the migration process.

A support library isn't a magic wand, yet it can contribute to reducing training and consulting costs. Language commands and control members have same name, same syntax and – more important – same behavior as in VB6. Our customers feel themselves immediately at ease when working with VB Migration Partner and its library.

As a result, in most cases you don't need to hire .NET developers or external consultants. As a matter of fact, **our customers are making the best of VB Migration Partner without any support from Code Architects** other than our standard tech support. In most cases the online documentation and the huge mass of KB articles available on our web site is enough for our customers to solve most common migration issues without even asking our tech support.

Training costs can't be avoided in the long run, though. If you plan to extend your migrated app with new features your developers must become familiar with .NET, its immense power, and its many idiosyncrasies. However, you can focus on delivering a working .NET application first, and later use your consulting budget for extending the application with new features.

11. Embedded debug features

VB Migration Partner's library comes with a few test and debug features already built-in. For example, if one of your customers reports a fatal bug, you can produce a detailed log by just changing one line in your code:

```
VB6Config.LogFatalErrors = True
```

You don't even need to recompile the application, because you can assign the LogFatalErrors from a configuration file. Once you receive the log file from your customer, you know exactly where the fatal error occurred. With some luck, you might be able to fix the problem very quickly and suggest a fix in a few hours.

In short, **a support library can help you track down your bugs more easily and in less time**.

12. Interoperability with VB6

A support library is also helpful to solve interoperability issues with VB6 apps that you don't want to migrate right away. Consider the following examples:

- The VB6 application uses binary or random files to store data
- The VB6 application uses text files to store user preferences, such as color, window position, etc.

VB Migration Partner handles both cases quite nicely. All binary and random files are accessed by means of the FileGet6 and FilePut6 methods in our support library (see the "Language Impedance" section, earlier in this article), which guarantee a very high degree of compatibility with files created by VB6.

All configuration data stored in text files also works correctly, because .NET apps generated by our migration software understand twips and all the ScaleMode settings (including user-defined modes). Also, the numeric value of enum types is the same as under VB6, therefore you can read an integer from a file, assign it to a property, and achieve the same result as in VB6:

```
' next line works because the Move method understands
```

```
        VB6 coordinate systems
txtName.Move( GetControlLeft("txtName"), GetControlTop("txtName") )
' next line works because the Alignment property understands VB6
    enum values
txtName.Alignment = GetControlAlignment("txtName")
```

As mentioned in the "Application advancement" section above, you aren't cornered in VB6 syntax, though. VB Migration Partner's controls are true .NET controls, therefore you can use .NET methods and properties instead, if you later extend the form with more features:

```
txtName.Location = New Point(100, 200)
txtName.TextAlignment = HorizontalAlignment.Center
```

13. Bug fixing

You are in the software industry; you know that a bug-free program hasn't been invented yet. Migration products are no exceptions, including VB Migration Partner.

In fact, we have no problem in conceding that the support library that comes with VB Migration Partner might have some quirks. On the other hand, the short yet intense history of our product proves that we can fix our bugs very quickly. Our customers know that we often provide a bug fix within a few hours, or a couple days in the worst cases. Many of them praised our tech support team too, as you can read here.

Consider two migration programs, one that uses a support library (VB Migration Partner), another that relies on code transformations exclusively (a competitor's tool). Let's now see what consequences a bug in these programs has on their respective customers.

Say that you are a Code Architects' customer who found a problem in how the ComboBox control reacts to data-binding. This is what is going to happen:

1. You contact our tech support and provide enough details to let us reproduce the problem, for example a fatal error log automatically produced by VB Migration Partner's support library (see the "Embedded debug features" section).
2. We fix the bug and send you a new version of the library.

3. You distribute the new library to your customers, who just overwrite the existing DLL and restart the application.

You might wait a few hours or a couple days. In either case, we do the bug fixing, not you.

Let's see now what happens if you are using a competing tool and one of your customers reports of a malfunctioning caused by an orphaned delegate object or the missing cleanup code for an IDisposable object. You must go through these steps:

1. You have to understand what went wrong. (Unfortunately, other tools give you no built-in debugging features to give you a hint...)
2. You have to work out a solution, which may require a lot of .NET expertise.
3. Once you have devised a robust solution, you have to apply it to your application. Odds are that you have myriads of objects that aren't correctly disposed of. This step can take a loooong time...
4. Finally, you can recompile all your EXEs and DLLs, make a setup procedure, and ship it to your customers.

Not surprisingly, 9 out of 10 developers would rather use a library to simplify tech support. The bottom line: **Fixing a bug caused by a library requires less time and effort on your part.**

Using a support library isn't different from using a custom control from any other company, be it Microsoft or a control vendor such as Infragistics or Component One. If you use a 3rd-party control you become dependent, to a degree, on the company who authored the control. If you bump into a bug in the .NET Framework or in a 3rd-party control, you have to wait until the manufacturer fixes the bug. Microsoft might take months or even years to fix even serious bugs in the .NET Framework; component vendors and tool makers such as Code Architects are typically more reactive, and our customers can attest it.

14. Runtime royalties

We know that a few vendors charge for their runtime libraries. This isn't the case with Code Architects. **VB Migration Partner users don't pay one cent for the ability to distribute its runtime library**.

15. Future upgrades

In addition to not paying for royalties, **VB Migration Partner users don't pay for future upgrades of the support library**. If you purchased VB Migration Partner you will be allowed to download all future releases of the support library. If we fix more bugs, your automatically benefit from those fixes. If we add minor features, you get them for free.

It's like a life-time insurance on your migrated code.

16. Source code

Our library isn't a "black box" and we can license its source code, under certain conditions:

a. Large companies and government agencies need the source code of any 3rd-party library or control they use, therefore we are willing to comply with these requirements.
b. Should we discontinue VB Migration Partner, **the library's source code will be provided free of charge to all registered customers**.

Before applying for the library's source code, consider that:

- You can modify the behavior of a control by inheriting from our class and overriding its methods and properties. You don't need the library's source code for doing that.
- Even if we sell you the source code (see case A above), we don't encourage you to modify it. In fact, we don't provide any form of support for doing that, including paid consulting.
- We don't offer tech support for applications that use a support library that has been recompiled by our customers.

As explained in the "Bug fixing" section, using a support library is conceptually similar to using a 3rd-party custom control. You might have used ActiveX controls for years without owning their source code. Or maybe you had purchased the source code but never even opened those files.

17. After-migration library improvements

A runtime library offers a few non-obvious advantages too. For example, not only it simplifies bug fixing: it even solves bugs before they occur. In fact, we periodically publish new versions of the library, which contain all the cumulative bug fixes implemented so far. If a bug is especially serious, we will publish a fix immediately and alert our users via our newsletter or blog. **You indirectly benefit from the experience of all VB Migration Partner's users all over the world**. You and your customers.

An example of these advantages has been obvious in VB Migration Partner version 1.20. In this version we added support for "classic" (VB3-style) drag-and-drop and for DDE-related members. These two features were the only major VB6 features that were missing in previous VB Migration Partner, thus we decided to fill this gap. When we released version 1.20, all VB Migration Partner customers just had to overwrite the old support library with the new one and – presto! – their migrated apps magically supported both drag-and-drop and DDE. (Only minor adjustments had to be performed manually in some cases.)

Conclusions

It took over 5,000 words to explain all the benefits of the support library approach in a migration project. Some benefits might be very important to you, some might be negligible. In all cases, we suggest that you carefully weigh all the options available and compare VB Migration Partner with migration tools that rely exclusively on code transformations.