VBUC vs VBUW

Visual Basic Upgrade Companion and Visual Basic Upgrade Wizard: Comparative Summary

VBUC vs VBUW: Summary

The Visual Basic Upgrade Wizard (VBUW) is a migration tool developed by Mobilize.Net that shipped with Microsoft Visual Studio until Visual Studio 2008. The tool was specifically designed to migrate applications from Visual Basic 6.0 to Visual Basic.NET. **There have been no improvements to VBUW since 2008**. Also, Visual Studio 2008 will no longer be supported by Microsoft as of April 10, 2018. Here's a summary of differences between VBUC and VBUW

Feature	VBUC	VBUW
Automated code migration		
VB.NET code output		
C# code output		
Number of extensions supported	7,000	250
Type inference		
ADO.NET support		
Structured error handling		
.NET native library support		
.NET enumerations		
Try…Catch error handling		
Code refactoring		
Code readability improvements		
Multiple project support		

The Visual Basic Upgrade Companion (VBUC) is an extended and more powerful version of the VBUW, dedicated to upgrading Visual Basic 6.0 applications to Visual

Basic.NET and C#. VBUW does not support C# In addition, VBUC is continuously upgraded and improved and has been used to convert thousands of projects and billions of lines of code. VBUC gets better each time it is used since new mappings and extensions are being discovered. There have been hundreds of upgrades to VBUC since 2008.

VBUW has a number of deficiencies that make it a less appealing tool than VBUC. For example, while VBUC can be customized according to your needs, increasing the percentage of automation of your VB migration project, VBUW has no extensibility option.

This paper highlights some of the productivity enhancements that the VBUC has over the VBUW that simplify the Visual Basic 6.0 to .NET migration process.

Extension capability:

VBUC can be extended to meet your specific needs. New extensions augment and extend the translation dictionary.

The VBUC extension capability enables you to automatically migrate your specific programming patterns, add new functionality to the migrated application and even migrate the ActiveX controls that you have in the original application to .NET Framework components or newer versions of the specific third-party controls, saving manual effort, time and money.

Here's a list of the ActiveX controls that VBUC automatically migrates, and their specific functionality coverage.

Original Component/Library	From	Target Component/Library	Vendor
COMSVCSLib	Microsoft	.NET intrinsic	Microsoft
CSTextLib	Crescent Software	C1Input	ComponentOne
FPSpread	FarPoint	Spread	FarPoint
MAPI	Microsoft	.NET intrinsic	Microsoft
Mh3dlblLib	MicroHelp	.NET intrinsic	Microsoft
MSACAL	Microsoft	.NET intrinsic	Microsoft
MSComCtl2	Microsoft	.NET intrinsic	Microsoft
MSComCtlLib	Microsoft	.NET intrinsic	Microsoft
MSComDlg	Microsoft	.NET intrinsic	Microsoft
MSDataGridLib	Microsoft	TrueDBGrid	ComponentOne
MSDBGridLib	Microsoft	TrueDBGrid	ComponentOne
MSFlexGridLib	Microsoft	FlexGrid	ComponentOne
MSMask	Microsoft	.NET intrinsic	Microsoft
MSWLess	Microsoft	.NET intrinsic	Microsoft
MSXML2	Microsoft	.NET intrinsic	Microsoft
MTxAS	Microsoft	.NET intrinsic	Microsoft

vb.Printer	Microsoft	Helper class	ArtinSoft
RichTextBox	Microsoft	.NET equivalents	Microsoft
Scripting	Microsoft	.NET intrinsic	Microsoft
SHDocVw	Microsoft	.NET intrinsic	Microsoft
SSActiveTreeView	Sheridan	.NET TreeView	Microsoft
SSCalendarWidgets	Sheridan	.NET equivalents	Microsoft
SSDataWidgets_B	Sheridan	TrueDBGrid	ComponentOne
SSDataWidgets_B	Sheridan	UltraWinGrid	Infragistics
SSDesignerWidgets	Sheridan	.NET TabControl	Microsoft
SSListBar	Sheridan	UltraWinListBar	Infragistics
SSSpliter	Sheridan	.NET SplitContainer	Microsoft
Threed	Sheridan	.NET TabControl	Microsoft
TrueDBGrid50Lib	APEX	TrueDBGrid	ComponentOne
TrueDBGrid60Lib	APEX	TrueDBGrid	ComponentOne
TrueDBGrid70Lib	APEX	TrueDBGrid	ComponentOne
VSFlex7Ctl	VideoSoft	C1FlexGrid	ComponentOne
VSFlex7LCtl	VideoSoft	C1FlexGrid	ComponentOne
VSOcxLib	Sheridan	.NET intrinsic	Microsoft
XArrayCustom	APEX	Helper class	Mobilize
XArrayObject	APEX	.NET intrinsic	Microsoft

This list is growing all the time.

Type Inference

An Artificial Intelligence-based type inference engine has been incorporated into the Visual Basic Upgrade Companion, which can infer the most appropriate data types for variables parameters and return values, avoiding the use of "generic" data types (i.e., Object). When an Object or Variant variable is found, the Visual Basic Upgrade Companion declares the variable with the appropriate type and avoids unnecessary migration errors, warnings and issues (EWIs).

By following the type inference approach, the amount of manual work that is required to check for Upgrade Warnings is drastically reduced.

Example Source Code

```
Private Sub loadini()


  Dim lngResult As Long
  Dim strFileName
  Dim strResult As String * 50
  strFileName = App.Path & "\createDsn.ini" 'Declare your ini file !
  lngResult = GetPrivateProfileString(KeySection, _KeyKey, strFileName,
strResult,Len(strResult), _strFileName)
```

```
…
End Sub
```

Code Generated by VBUW

```
Private Sub loadini()
 Dim lngResult As Integer
 Dim strFileName As Object
 Dim strResult As New VB6.FixedLengthString(50)
 'UPGRADE_WARNING: Couldn't resolve default property of object
 strFileName.
 strFileName = My.Application.Info.DirectoryPath & "\createDsn.ini"
 'Declare your ini file !
 'UPGRADE_WARNING: Couldn't resolve default property of object
 strFileName.
 lngResult = GetPrivateProfileString(KeySection, KeyKey, strFileName,
 strResult.Value, Len(strResult.Value), strFileName)
…
End Sub
```

Code Generated by VBUC

```
Private Sub loadini()
 Dim strResult As New VB6.FixedLengthString(50)
 Dim strFileName As String = My.Application.Info.DirectoryPath &
"\createDsn.ini" 'Declare your ini file !
 Dim lngResult As Long = GetPrivateProfileString(KeySection, KeyKey,
 strFileName, strResult.Value, Strings.Len(strResult.Value), strFileName)
End Sub
```

ADO to ADO.NET

VBUC upgrades the data model from ADO to ADO.NET as opposed to VBUW, which generates a target application that still uses ADO technology to communicate with the database via COM Interop wrapper calls. Visual Studio .NET offers a new and completely redesigned collection of classes for data access, which take into consideration modern application requirements of distribution, reliability and scalability. This new data access model is ADO.NET and, in addition to the ADO features, provides the following advantages:

- **Interoperability:** All data in ADO.NET is transported in XML format. The data is provided as a structured text document that can be read by anyone on any platform.
- **Scalability:** ADO.NET promotes the use of disconnected datasets, with automatic connection pooling bundled as part of the package.
- **Productivity:** ADO.NET can improve overall development time. For example, typed DataSets help you work more quickly and allow you to produce more bug-free code.
- **Performance:** Because ADO.NET provides disconnected datasets, the database server is no longer a bottleneck and application performance is improved.

Example Source Code

```
Private Sub Form_Load()
      createDsn
      Set db = New ADODB.Connection
      Set rs = New ADODB.Recordset
      db.Open "Provider=MSDASQL;DSN=TikkisDb;Password=1515151515;"
      rs.Open "SELECT * FROM WestSide", db, adOpenKeyset,
      adLockPessimistic
End Sub
```

Code Generated by VBUW

```
Private Sub frmLogin_Load(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles MyBase.Load
      createDsn()
      db = New ADODB.Connection
      rs = New ADODB.Recordset
```

```
        db.Open("Provider=MSDASQL;DSN=TikkisDb;Password=1515151515;")
        rs.Open("SELECT * FROM WestSide", db,
        ADODB.CursorTypeEnum.adOpenKeyset,
        ADODB.LockTypeEnum.adLockPessimistic)
End Sub
```

Code Generated by VBUC

```
Private Sub frmLogin_Load(ByVal eventSender As Object, ByVal eventArgs As
EventArgs) Handles MyBase.Load
        createDsn()
        db = New SqlConnection
        rs = New DataSet
        db.Open("Provider=MSDASQL;DSN=TikkisDb;Password=1515151515;")
        Dim com As SqlCommand = New SqlCommand()
        com.Connection = db
        com.CommandText = "SELECT * FROM WestSide"
        Dim adap As SqlDataAdapter = New SqlDataAdapter(com.CommandText,
        com.Connection)
        rs = New DataSet("dsl")
        adap.Fill(rs)
End Sub
```

C# Generation

The Visual Basic Upgrade Companion is able to generate C# directly from the Visual Basic 6.0 source code as an alternative to Visual Basic .NET. Since there are several Visual Basic 6.0 features not available in C# (i.e. Optional Parameters, Modules, ReDim, With), the Visual Basic Upgrade Companion performs special conversions for these features This functionality is not available on VBUW.

Example Source Code

```
Private Sub loadini()
  Dim lngResult As Long
  Dim strFileName
  Dim strResult As String * 50
  strFileName = App.Path & "\createDsn.ini" 'Declare your ini file !
  lngResult = GetPrivateProfileString(KeySection, _KeyKey, strFileName,
strResult,Len(strResult), _strFileName)
```

…

```
End Sub
```

Code Generated by VBUC

```
private void  loadini()

{

      FixedLengthString strResult = new FixedLengthString(50);

      string strFileName =
      Path.GetDirectoryName(Application.ExecutablePath) +
      "\\createDsn.ini"; // Declare your ini file !

      long lngResult = Module1.GetPrivateProfileString(KeySection,
      KeyKey, strFileName, strResult.Value, Strings.Len(strResult.Value),
      strFileName);
```

…

```
}
```

Structured Error Handling

The Visual Basic Upgrade Companion recognizes most *"On Error GoTo"* patterns and replaces them with the .NET *"Try … Catch"* error handling preferred constructs. WBUW upgrades the application using the same *"On Error GoTo"* pattern that Visual Basic 6.0 uses for handling errors.

Using VBUC, the generated code is easier to understand and conforms to the coding standards used when programming with .NET languages.

Example Source Code

```
Private Sub Button7_Click()
      On Error GoTo error12
      If List1.Text = "" Then
        MsgBox "Please Select from List", vbCritical
        Exit Sub
      End If
      …
      error12:
        MsgBox Err.Description, vbCritical
        Exit Sub
End Sub
```

Code Generated by VBUW

```
Private Sub Button7_Click(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles Button7.Click
      On Error GoTo error12
      If List1.Text = "" Then
         MsgBox("Please Select from List", MsgBoxStyle.Critical)
         Exit Sub
      End If
```

```
…

error12:

      MsgBox(Err.Description, MsgBoxStyle.Critical)

      Exit Sub

End Sub
```

Code Generated by VBUC

```
Private Sub Button7_Click(ByVal eventSender As Object, ByVal eventArgs As
EventArgs) Handles Button7.Click

      Try

      If List1.Text = "" Then

         MessageBox.Show("Please Select from List", String.Empty,
         MessageBoxButtons.OK, MessageBoxIcon.Error)

         Exit Sub

      End If

      …

      Catch excep As System.Exception

         MessageBox.Show(excep.Message, String.Empty,
         MessageBoxButtons.OK, MessageBoxIcon.Error)

         Exit Sub

      End Try

End Sub
```

.NET Native Libraries:

Instead of upgrading VB6 code using the Visual Basic Compatibility Libraries like the VBUW does, the VBUC promotes the use of .NET native libraries whenever possible.

There are several functions that when upgraded, still rely on the Visual Basic compatibility library. Once again, this does not mean that your code will not compile; however, your code will be better off using the native libraries that the .NET framework offers. By using native libraries, you are making your code easier to read, easier to maintain, and in some cases you are improving the performance of the

application. In the following table you can compare the end result of upgrading the Left, InStr and Len functions with VBUW and VBUC:

VB6 Code	VB.NET Code with UW	VB.NET Code with VB Companion
Left(strvar, 1)	VB.Left(strvar,1)	strvar.Chars(0)
Left(strvar, cant)	VB.Left(strvar,cant)	strvar.Substring(0, cant)
InStr(strvar1,strvar2)	InStr(strvar1, strvar2)	strvar1.IndexOf(strvar2) >= 0
Len(strvar)	Len(strvar)	strvar.Length

The code that is upgraded with the Upgrade Wizard relies on the same functions that were used in Visual Basic 6.0 and therefore, uses the Visual Basic compatibility libraries. On the other hand, the VBUC migrates functions such as Len to the Length property of the .NET String class. The table also shows the result of migrating the Left and InStr functions using the Wizard and the VBUC. VBUC uses properties from native classes, improving speed of the application by eliminating the overhead involved with interoperability.

.NET Enumerations:

Another important Visual Basic Upgrade Companion feature is that it replaces numeric literals assigned to several control properties with .NET enumeration equivalents when possible, so that the generated Visual Basic .NET code is more legible and maintainable.

Example Source Code

```
Sub Foo()

      num = vbArrow

      Me.MousePointer = num

      Me.MousePointer = 11

      Me.MousePointer = vbArrow

End Sub
```

Code Generated by VBUW

```
Sub Foo()

      Dim num As Object

      'UPGRADE_WARNING: Couldn't resolve default property …

      num = System.Windows.Forms.Cursors.Arrow

      'UPGRADE_WARNING: Couldn't resolve default property …

      'UPGRADE_ISSUE: Form property Form1.MousePointer does not support
      custom mouse pointers.

      Me.Cursor = num

      Me.Cursor = 11

      Me.Cursor = System.Windows.Forms.Cursors.Arrow

End Sub
```

Code Generated by VBUC

```
Sub Foo()

      Dim num As System.Windows.Forms.Cursor =

      System.Windows.Forms.Cursors.Arrow
```

```
        Me.Cursor = num

        Me.Cursor = System.Windows.Forms.Cursors.WaitCursor

        Me.Cursor = System.Windows.Forms.Cursors.Arrow
End Sub
```

The code that was upgraded with VBUW presents several compilation and runtime problems. For instance, the literal '11' assigned to the Cursor property should be converted to its respective enumeration to make the code compile.

VBUC has made various improvements to the upgraded code. First, it can be observed in the first line in the function that the variable 'num' was correctly identified as type Cursor, as opposed to the type Object that was defined by the Upgrade Wizard. If you look at the line corresponding to the assignment of a constant to the Cursor property, you can also identify that the VBUC has converted the value of '11' to the corresponding enumeration value.

Comparison between VBUW and VBUC in terms of quality of code

Another important difference between VBUC and VBUW is the quality of the code generated by the tools, which is the main point of this section. We are going to show some of the code quality improvements that the VBUC has over the VBUW.

VBUC does further code analysis to detect patterns that can be upgraded to more .NET-like, native structures. These aspects make the output code more readable and maintainable.

These code improvements include:

- Long "If..ElseIf" constructs are upgraded to the "switch" construct ("Select Case" in VB.NET) in order to improve performance and use better programming practices.
- VBUC uses the "Return" keyword instead of the function name to set the return value within a function.

Example Source Code

```
Public Function TrimSpaces(Text As String) As String

    Dim Loop1 As Long, SpaceCheck As String
```

```
    Dim FullString As String
      …
    TrimSpaces = FullString$
End Function
```

Code Generated by the VBUW

```
Public Function TrimSpaces(ByRef Text As String) As String
      Dim Loop1 As Integer
      Dim SpaceCheck As String
Dim FullString As String
…
TrimSpaces = FullString
End Function
```

Code Generated by the VBUC

```
Public Function TrimSpaces(ByRef Text As String) As String
Dim SpaceCheck, FullString As String
      …
Return FullString
End Function
```

- Collections are upgraded to ArrayList or HashTable depending on their usage.
- "For...Each" blocks are used instead of an cycles that use iteration variables.

Example Source Code

```
Private Sub GetAuthorList()
      …
      While (Not rs.EOF)
        List1.AddItem rs.Fields("Author").Value
        rs.MoveNext
```

```
        …
End Sub
```

Code Generated by the VBUW

```
Private Sub GetAuthorList()
        …
        While (Not rs.EOF)
                List1.Items.Add(rs.Fields("Author").Value)
                rs.MoveNext()
        End While
        …
End Sub
```

Code Generated by the VBUC

```
Private Sub GetAuthorList()
        …
        For Each iteration_row As DataRow In
                rs.Tables(0).Rows
                List1.Items.Add(iteration_row.Item("Author"))
        Next iteration_row
        …
End Sub
```

- Initialization values for variables are moved to the variable declaration.

Example Source Code

```
Private Sub Command2_Click()

    Dim year1 As Integer

    Dim year2 As Integer

    year1 = Format(SSDateCombo1.Date, "yyyy")

    year2 = Format(SSDateCombo2.Date, "yyyy")

    GetTitlesByPublishedYear year1, year2

End Sub
```

Code Generated by the VBUW

```
Private Sub Command2_Click(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
Command2.Click

    Dim year1 As Short

    Dim year2 As Short

    year1 = CShort(VB6.Format(SSDateCombo1.Date, "yyyy"))

    year2 = CShort(VB6.Format(SSDateCombo2.Date, "yyyy"))

    GetTitlesByPublishedYear(year1, year2)

End Sub
```

Code Generated by the VBUC

```
Private Sub Command2_Click(ByVal eventSender As Object, ByVal
eventArgs As EventArgs) Handles Command2.Click

    Dim year1 As Integer =
    CInt(CDate(SSDateCombo1.Value.Date).ToString("yyyy"))

    Dim year2 As Integer =
    CInt(CDate(SSDateCombo2.Value.Date).ToString("yyyy"))

    GetTitlesByPublishedYear(year1, year2)

End Sub
```

- Typical nested "If" statements used in VB6 to produce short circuit evaluation are upgraded to a single "If" statement with short circuit evaluation operators (AndAlso and OrElse).

Mobilize your 90's desktop app.