VB6 vs .NET controls

This whitepaper describes the differences between VB6 and .NET controls and the problems you can find in migrating VB6 applications with user-interface. The differences that are common to most controls are described in the All Controls (general) section.

Contents:

All Controls (general)

Align property

A few VB6 controls – including the PictureBox and Data control – expose the Align property, which permit to automatically dock the control to one of the sides of the form border. Under .NET this property is replaced by the Dock property, which is exposed by all the controls in the System.Windows.Forms namespace.

Appearance property

VB6 vs .NET controls

Many VB6 controls support the Appearance property, which enables to select between flat and 3D appearance. .NET doesn't support this property, because the control's appearance is dictated by the operating system.

Controls in VB Migration Partner's support library expose the Appearance property, which always returns the value 1-3D; any attempt to assign a different value is ignored (or throws an exception if **VB6Config.ThrowOnUnsupportedMembers** property is set to True).

AutoRedraw property

.NET doesn't support the AutoRedraw property, therefore implementing double-buffered graphics requires manual code edits.

VB Migration Partner supports and replicates the exact behavior of the AutoRedraw property, thus you can easily convert VB6 applications that use double-buffering techniques and persistent graphic output. As in VB6, no Paint event is raised when AutoRedraw is True.

BackColor property

When you drop a VB6 Label control on a form, the initial value of its BackColor property is set to ButtonFace; when you drop a .NET Label control on a form, the initial value of its BackColor property is equal to the background color of the container control. This behavior is common to other .NET controls, such as Button.

Caption property

No .NET control exposes the Caption property, which has been converted to the Text property. Notice that assigning the Text property a string equal to Nothing causes a runtime exception.

VB Migration Partner support the Caption property, to ensure that converted code works as intended even if the control is accessed via late binding. Strings equal to Nothing are converted to empty strings before being assigned to the Text property, thus avoiding unexpected runtime exceptions.

ClipControls property

The VB6 Form, UserControl, and PictureBox objects expose the ClipControls property, which allows to developers to speed up the user interface in some circumstances. .NET doesn't expose such a property.

Controls in VB Migration Partner's support library expose this property, but it always returns False; attempts to assign a different value are ignored (or throw an exception if the **VB6Config.ThrowOnUnsupportedMembers** property is True).

Color values

VB6 assigns and manipulates color values and properties – such as ForeColor and BackColor - by means of 32-bit integers; .NET and all .NET Framework languages represent color values by means of instances of the System.Drawing.Color type. In most circumstances both the source and the destination of a color assignment become Color values after the migration to .NET, therefore you don't need any specific fix to have the code compile correctly.

In some cases, however, you might have to convert a 32-bit integer to a color value, or vice versa. This explicit conversion is necessary, for example, if your application reads color values from a data file or a database, or if you dynamically calculate a color value by means of a method (e.g. when converting an image to gray scale). You can perform such conversions by means of the ToOle, FromOle, and FromWin32 methods of the System.Drawing.ColorTranslator type:

```
txtName.BackColor = ColorTranslator.FromOle(integerColorValue)

integerColorValue = ColorTranslator.ToOle(txtName.BackColor)
```

For readability's sake, VB Migration Partner generates calls to **FromOleColor6** and **ToOleColor6** methods, which in turn invoke the methods of the ColorTranslator type.

Container property

All visible VB6 controls expose the Container property, which returns a reference to the form or control that contains the current control. Interestingly, the Container property is writable under VB6, a feature that allows developers to move controls from one container to another container (on the same form) at runtime, as in this code:

```
' move the txtName control into the picFrame container (a PictureBox container)

Set txtName.Container = picFrame
```

The Container property corresponds to the Parent property under .NET, but the two properties aren't equivalent because the Parent property is readonly. You can change the container of a .NET control only by removing the control from the Controls collection of current container and adding it to the Controls collection of a different container:

```
' move the txtName control into the picFrame container (a PictureBox container)

txtName.Parent.Controls.Remove(txtName)  ' remove from current container

picFrame.Controls.Add(txtName)           ' add to new container
```

VB Migration Partner fully supports the Container property, including the ability to assign it for moving the control to another control.

Control accelerator keys

Both in VB6 and .NET, you assign a control accelerator key by inserting an & (ampersand) character in the control Caption (VB6) or Text (.NET) property:

```
lblName.Caption = "&Name"
```

However, .NET controls running under Windows XP or later versions don't display the accelerator key when the form loads. It becomes visible when the end user presses the Alt key.

The only documented way to always display control accelerators is by sending a WM_CHANGEUISTATE to the parent form, using the SendMessage API method. However, things are quite intricate because you must send this message after the form has initialized but before the form becomes visible. You must do this for each form in your application.

VB Migration Partner makes things easier, in that you only need to assign the **VB6Form.ShowAccelerators** property to True when the program starts. Then you can forget about the whole thing.

Control arrays

.NET doesn't support control arrays, therefore it is required that replace them with native .NET code. A VB6 developer typically use control arrays for two reasons: to create a new control at runtime and to centralize event handling, so that a single method can handle events coming from multiple controls.

.NET offers solutions for both tasks: you can create a new control simply by using the New operator and you can centralize event handling by means of the AddHandler operator. However, such techniques are quite different from the original control array concept and prevent from using an automatic code translator to migrate existing VB6 code.

Control variables

The Upgrade Wizard and other converters based on its engine convert VB6 Control variables to System.Windows.Forms.Control variables. However, the VB6 Control type is actually an IDispatch variable; in other words, it supports late binding and it is more akin to an Object variable than to a System.Windows.Forms.Control object. In fact, consider the following code snippet:

```
Dim ctrl As Control

For Each ctrl In Form1.Controls

    If Type ctrl Is CheckBox Then ctrl.Value = 0
```

```
Next
```

If the *ctrl* variable is converted to a .NET Control variable, then the ctrl.Value reference causes a compilation error, because the System.Windows.Forms.Control class doesn't expose a member named Value. Additionally, VB6 allows you to store invisible controls to a Control variable – for example, Timer and ImageList controls. These controls are translated to .NET components and can't be assigned to a System.Windows.Forms.Control variable.

For all the above mentioned reasons, VB Migration Partner converts Control variables to Object variables. For the same reason, the ActiveControl property of the Form and Screen classes return an Object value instead of a Control reference.

Coordinate systems in forms with menus

The (0,0) point in a VB6 form that contains a visible menu bar corresponds to the left-most point immediately below the menu bar; if the menu bar becomes invisible, the coordinate original is shifted accordingly and all controls are automatically moved up so that their Left and Top properties don't change. By contrast, .NET system coordinates ignore the presence of a menu bar; if the menu becomes invisible, controls aren't shifted.

VB Migration Studio accounts for this important detail, both during the migration process and when the converted .NET program is running. Forms migrated with the Upgrade Wizard and other converters based on the same engine do not.

CurrentX, CurrentY properties

GDI+ doesn't support the notion of the "current point" and the methods that draw lines and polygons must always specify the starting point and the ending point. For this reason, no .NET control supports the CurrentX and CurrentY properties.

VB Migration Partner supports all the graphics methods as well as the CurrentX and CurrentY properties, which are correctly scaled according to the current ScaleMode setting.

Data binding

Both VB6 and .NET support data-binding for their controls, but the actual mechanism differs greatly between the two languages. VB6 supports data binding to different data sources, namely DAO Data controls, RDO Data controls, ADO Data control (ADODC), DataEnvironment objects, ADO Recordsets, and ADO data source objects. Data-binding can be further refined by means of the StdDataFormat object and its Parse and Format events.

By contrast, .NET supports data binding with *any* object, because the actual binding capabilities are offered by the System.Windows.Forms.Binding object; this mechanism isn't compatible with the VB6 way of doing data binding.

Controls in VB Migration Partner's support library expose all the usual data-binding properties – that is, DataField, DataSource, DataMember, DataFormant, and DataChanged – and can be bound to the

same data sources that VB6 supports, including the various flavors of Data controls, DataEnvironment objects, ADO Recordsets, and ADO data source classes. VB Migration Partner even supports UserControls that work as ADO data sources, therefore it can migrate custom Data controls written in VB6. StdDataFormat objects and their Format and Parse events are also supported.

Default form instances

Both VB6 and .NET support the so-called default form instances: in other words, you don't need to explicitly create a form object and can reference the form by means of a special global variable that is named after the form itself:

```
Form1.Left = 200

Form1.Top = 300
```

However, .NET doesn't support such references inside the form itself. In other words, previous code causes a compilation error if it is located inside the Form1 class.

VB Migration Partner solves this problem by replacing the form reference with a reference to the Me object:

```
Me.Left = 200

Me.Top = 300
```

A special case occurs when the current form is unloaded and then set to Nothing:

```
Unload Form1

Set Form1 = Nothing
```

However, if the form being referenced is the current form then the default form reference must be replaced by the Me (VB.NET) or this (C#) keyword, which in turn causes the generation of an invalid statement. For this reason, VB Migration Partner generates a remarked assignment:

```
'EXCLUDED:  Set Me = Nothing
```

Default form instances (assignments)

VB6 allows you to assign Nothing to the default form instance as well a fresh new instance:

```
Set Form1 = Nothing

Set Form1 = New Form1
```

Interestingly, the latter syntax is equivalent to the former syntax: in fact, if you set a default form instance to Nothing and then reference again the variable, a new form is created automatically. In

.NET only the former assignment is valid; assigning a non-Nothing object to a default form instance causes a compilation error.

VB Migration Partner takes advantage of the abovementioned equivalence and converts the latter syntax to the former one.

Drag-and-drop features

VB6 offers two flavors of drag-and-drop features: the "classic" VB3-style drag-and-drop and the more modern OLE drag-and-drop. The former allows you to drag and drop items within a VB application only and is based on the DragMode property, Drag method, and DragOver and DragDrop events. The latter was introduced in VB5, allows to drag items from and to other Windows applications (including Windows Explorer), and is based on the OLEDragMode and OLEDropMode properties, the OLEDrag method, and the OLEStartDrag, OLESetData, OLEDragOver, OLEDragDrop, OLECompleteDrag, and OLEGiveFeedback events.

Both models support "automatic" and "manual" drag-and-drop: in automatic mode a control can start a drag-and-drop operation autonomously when the user drags the mouse over the control itself, whereas in manual mode the drag-and-drop operation begins when the developer invokes either the Drag or OLEDrag method.

.NET drag-and-drop programming model differs from both "classic" and OLE drag-and-drop model. In general, converting drag-and-drop code from VB6 to .NET isn't a trivial task. In fact, the Upgrade Wizard tool and other converters based on the same engine don't even attempt to convert drag-and-drop properties, methods, and event handlers.

VB Migration Partner manages to successfully convert both "classic" and OLE drag-and-drop code, in automatic and manual mode.

Dynamic Data Exchange (DDE)

.NET doesn't support DDE and the Upgrade Wizard (or other converters based on the same engine) can't migrate any member that is related to DDE, namely LinkMode, LinkTopic, LinkItem and LinkTimeout properties; LinkExecute, LinkPoke, LinkRequest, and LinkSend methods; LinkOpen, LinkClose, LinkNotify, LinkExecute, and LinkError events.

VB Migration Partner fully supports DDE communications between two or more converted .NET applications, but not between a .NET application and another (non migrated) application, such as Microsoft Excel. Also, the LinkEvent event and LinkTimeout property aren't supported.

Font inheritance

Both VB6 and .NET controls inherit their default font settings from their container (e.g. the parent form), however "inheritance" works differently in the two cases. A VB6 control that doesn't use a specific value for its Font property is assigned a *copy* of the font used by its container; in .NET a

control that doesn't define a custom font is assigned a *reference* to the font used by its container. It seems a minor difference, but it isn't.

In VB6 you can later change a font attribute – name, size, bold, italics, etc. – of the container and nothing else happens; in .NET changing a font attribute of the container affects all the child controls that have inherited the Font property from the container.

VB Migration Partner offers a special method named **FreezeControlsFont6** which forces .NET controls to receive a copy of their container's font and therefore to behave as in VB6.

Font object

VB6 applications can use the StdFont object – defined in the stdole2 type library. This object broadly corresponds to the System.Drawing.Font object in the .NET Framework, but the two objects aren't perfectly equivalent. For example, the VB6 Strikethrough property must be translated to the .NET Strikeout property; the VB6 Weight property can only approximated by means of the Bold property. Finally, the VB6 FontChanged event has no equivalent in .NET, because .NET fonts are immutable objects and can't be changed.

The fact that .NET font are immutable means that a VB6 statement such as:

```
txtName.Font.Bold = True
```

can't be translated directly to .NET, because the Bold property is readonly, as are all the other properties of the System.Drawing.Font class.

VB Migration Partner solves this problem by automatically generating a call to the **FontChangeBold6** special method:

```
FontChangeBold6(txtName.Font, True)
```

Similar support methods exist for the Name, Size, Italic, Strikeout, and Underline properties.

Support for the Weight property is ensured by the **GetFontWeight6** and **SetFontWeight6** methods. For example, the following VB6 statement:

```
txtName.Font.Weight = txtName.Font.Weight + 200
```

is translated to this VB.NET code:

```
SetFontWeight6(txtName.Font,  GetFontWeight6(txtName.Font) + 200)
```

VB Migration Partner emits a warning if the VB6 application being converted handles the FontChanged event of a Font object. As mentioned above, no simple workaround for this problem exists.

Font property (assignments)

The VB6 StdFont type maps to the System.Drawing.Font type under .NET. However, the two types differ for an important detail: the .NET Font type is *immutable*, which means that you must set all its properties when you create the font and can't re-assign them later. In other words, the following VB6 statement is illegal under VB.NET:

```
txtName.Font.Bold = True
```

The Upgrade Wizard fixes this problem by performing the assignment by means of helper methods defined in the Microsoft.VisualBasic.Compatibility.dll assembly:

```
FontChangeBold(txtName.Font, True)
```

The VB Migration Partner adopts a similar approach, except it uses methods in the proprietary support library.

Another problem you might face is that the .NET Font object has no property that corresponds to the StdFont.Weight property. VB Migration Partner (but not the Upgrade Wizard or other converters based on the same engine) solves this minor problem by means of the **GetFontWeight6** and **SetFontWeight6** helper methods.

Font-related properties

VB6 controls expose several font-related properties - including FontName, FontSize, FontBold, FontItalic, FontStrikethru, and FontUnderline – in addition to the Font property that takes a StdFont object. .NET controls only support the Font property.

Form chaining

VB6 supports *form chaining*, a fancy term that means that any VB6 form – including the startup form - can unload itself and load another form. For example, VB6 applications often use form chaining techniques to display a splash screen

.NET supports this mechanism only for forms that aren't the startup form. You should implement splash screens under .NET by enabling the related option in the Application tab of the My Project property page.

VB Migration Partner allows you to load a special **VB6HiddenStartupForm** object that works as the startup form, so that all other forms can use form chaining without any restriction. You can leverage this hidden form by inserting the **InitializeFormChaining6** method at the top of the Sub Main method.

Form reuse

VB6 allows you to reuse a variable that references a form that has been unloaded:

```
Unload Form1
   …
Load Form1
```

No similar sequence is available in .NET: if a form is closed, the variable can't be reused, because all associated Windows resources have been disposed of.

VB Migration Partner keeps track of whether a form has been closed and correctly re-initializes if it is loaded. Code generated by the Upgrade Wizard and other converters based on the same engine must be manually revised to ensure that it loads correctly.

Forms collection

The VB6 Forms collection contains all the loaded forms, including those that are loaded but not visible yet (or that have been hidden). By contrast, the .NET OpenForms collection contains only the forms that are visible.

Code generated by the Upgrade Wizard and other converters based on the same engine might deliver incorrect results at runtime.VB Migration Partner supports a **Forms6** collection that behaves exactly like the original VB6 collection.

GotFocus and LostFocus events

.NET controls support the GotFocus and LostFocus events, however Microsoft recommends that you use the Enter and Leave events instead.

Moreover, there are minor differences in the sequence in which the LostFocus, GotFocus, and Validating event fire. In .NET the actual sequence depends on whether the focus is moved by means of the Tab key, an Alt+hotkey combination, or the mouse. In most cases, these differences are really negligible. If they aren't, however, under VB Migration Partner you can set the **VB6Config.FocusEventSupport** to True for full emulation of the VB6 behavior.

Graphic-related properties and methods

You can dynamically produce graphic output on a VB6 form, usercontrol, or PictureBox control by means of methods such as Cls, PSet, Line, Circle, Print, and PaintPicture. The output of such methods can be affected by means of many properties, such as AutoRedraw, CurrentX, CurrentY, DrawMode, DrawStyle, DrawWidth, FillColor, and FillStyle. VB6 supports a few other graphic-related methods, such as Point, TextWidth, and TextHeight.

None of these properties and methods is exposed by .NET forms or controls. A .NET Framework application produces graphic output by means of GDI+ objects and methods, and the two programming models are completely different. In general, translating a graphic-intensive piece of

VB6 code to .NET is a major effort that requires in-depth knowledge of the two languages. Bear in mind, for example, that VB6 graphic methods are affected by the current ScaleMode setting and that experienced VB6 developers can leverage the ScaleLeft, ScaleTop, ScaleWidth, and ScaleHeight properties to create custom coordinate systems so that the graphic output appears mirrored along the X- or Y-axis, or even rotated by 180 degrees.

VB Migration Partner fully supports all the graphic properties and methods, with the only exception of the DrawMode property. (The reason: GDI+ doesn't support an equivalent feature.) Custom ScaleMode settings are fully supported, as are the various fill modes with hatched brushes, arcs, pies, and so forth.

hDC property

The Form, PictureBox, and UserControl classes expose the hDC readonly property, which returns the handle of the control's device context. No similar property exists for .NET Framework. If you really need a device context handle to be passed to a Windows API method, you should use the following code sequence:

```
Dim gr As Graphics = txtName.CreateGraphics()

Dim handle As IntPtr = gr.GetHdc()

Dim hDc As Integer = handle.ToInt32()

' use the hDC value here ...

…

' release the handle and destroy the device context

gr.ReleaseHdc()

gr.Dispose()
```

It is essential that you release both the handle (with the ReleaseHdc method) before doing any other operation on the control, including a simple refresh.

Controls in VB Migration Partner's library expose the hDC property and don't require you to go through the previous sequence, except that you should invoke the **ReleaseHdc** method that the Form, PictureBox, and UserControl classes expose.

Help formats

VB6 applications can use help files in two different formats: .hlp files and .chm files. .NET doesn't support the .hlp format, though.

VB Migration Partner solves this problem by using Windows API methods to display help pages.

Help-related properties and methods

Most VB6 controls can be associated to a help page by means of the HelpContextID and WhatsThisHelpID properties, and can use the ShowWhatsThis method to display help. None of these properties are supported by .NET controls, which in fact can display help only by means of an HelpProvider control.

VB Migration Partner correctly supports all the help-related properties and methods in converted applications.

hWnd property

The VB6 hWnd property maps to the Handle property under .NET. However, the Handle property returns an IntPtr value, therefore you might need to convert it to an integer during the migration process.

Image and Picture properties

A few VB6 objects – namely, the Form, UserControl, and PictureBox objects – support both the Picture property and the Image property. The difference between the two is subtle: the Picture property is writeable and allows you to load a bitmap on object's background, for example by means of a LoadPicture method; the Image property is readonly and returns the current image contained in the object. (It may differ from the Picture property if you've used graphic methods to draw lines and circles on the object's surface.)

To make matters more complicated, the actual behavior of these properties is further influenced by the AutoRedraw property. For example, consider the following sequence of actions under VB6:

1. set AutoRedraw to True
2. produce graphic output by means of Circle and Line methods
3. reset AutoRedraw to False
4. produce more graphic output with Circle and Line methods
5. invoke the Cls method.

As surprising as it may sound, the Cls method doesn't really clear all graphic output, because the lines and circles produced in step 2 have become part of the persistent background image: the Cls method simply restores such background image and has therefore the effect to delete only the lines and circles produced at step 4.

VB Migration Partner fully supports both the Image and Picture properties, including their relation with the AutoRedraw property. In fact, the previous sequence is translated correctly to .NET and behaves as intended.

Index property

.NET doesn't support control arrays and consequently doesn't support the Index property.

VB Migration Partner supports both control arrays and the Index property. If a .NET control was originally part of a VB6 control array, its name embeds the Index value in its last three characters. For example, the control belonging to the *txtFields* control array and whose Index property is 10 is named *txtFields_010*.

KepPress event

.NET supports the KeyPress event, but the mechanism for ignoring a key press is different: in VB6 you can ignore a key by setting the KeyAscii argument to zero; in .NET you have to set the Handled property of the KeyPressEventArgs object to True:

```
Private Sub txtName_KeyPress(ByVal sender As Object, ByVal e As KeyPressEventArgs) _
    Handles txtName.KeyPress
    ' ignore spaces
    If e.KeyChar = " "c Then e.Handled = True
End Sub
```

The Upgrade Wizard and other converters based on the same engine convert correctly KeyPress event handlers, but generate a lot of additional code to correctly set the Handle property on exiting the event; VB Migration Partner produces readable and maintainable or C# code that preserves the structure and simplicity of the original VB6 code.

Left, Top, Height, and Width properties

The position and size of VB6 controls is determined by their Left, Top, Width, and Height properties; these properties are supported by .NET controls, but the form designer expects that these values be expressed by means of the Position and Size properties, which take a Point and a Size object, respectively.

In addition to using different properties, VB6 and .NET differ in how position and size values are calculated. By default, in VB6 you specify these values as twips (1 pixel = 15 twips) but the actual unit being used depends on the form's ScaleMode property. This property can take seven values: Twips, Point, Pixel, Character, Inch, Millimeter, Centimeter, plus a special value User; when User is specified, the coordinate system is user-defined and is affected by the ScaleLeft, ScaleTop, ScaleWidth, and ScaleHeight properties. (In VB6 you can set all these properties in one step by means of the Scale method.)

VB Migration Partner fully supports all the ScaleMode variants, including the special User value, the ScaleLeft, ScaleTop, ScaleWidth, and ScaleHeight properties, and the Scale method. More precisely,

the Left, Top, Width, and Height properties exposed by all the controls in the support library return a value that accounts for the current ScaleMode. Likewise, VB Migration Partner ensures that all coordinates passed as parameters or returned by methods – for example the Move, PSet, Line, Circle, TextWidth, TextHeight, ScaleX, and ScaleY methods – are interpreted correctly and work as in VB6.

Locked property

A few VB6 controls – namely the TextBox, ComboBox, RichTextBox, ImageCombo, DataCombo, and DataList – expose the Locked property. While this name isn't reserved under VB.NET, it interferes with Visual Studio's form designer, and in fact if a .NET control exposes such a property, then the property is shadowed by the Locked property that the form designer adds to all .NET controls. For this reason, .NET controls expose the ReadOnly instead of the Locked property.

MouseDown, MouseMove, and MouseUp events

.NET supports the Click, DblClick, MouseDown, MouseMove, and MouseUp events. However, the DblClick event has been renamed as DoubleClick and the coordinates passed to the MouseDown, MouseMove, and MouseUp events are always in pixels. (In VB6 these coordinates are in the current ScaleMode coordinate system.)

The Upgrade Wizard and other converters based on the same engine convert correctly mouse event handlers, but generate additional code to extract information out of the MouseEventArgs object passed to these events; VB Migration Partner produces readable and maintainable .NET code that preserves the structure and simplicity of the original VB6 code.

MousePointer and MouseIcon properties

.NET controls doesn't support the MousePointer and MouseIcon properties, which have been replaced by the Cursor property.

Paint event

Form, PictureBox, and UserControl objects raise the Paint event under VB6 only if the AutoRedraw is set to False; .NET doesn't support the AutoRedraw property, therefore these objects always raise the Paint event.

Controls in VB Migration Partner's library raise the Paint property only if AutoRedraw is set to False, as in VB6.

Parent property

All visible VB6 controls expose the Parent property, which returns a reference to the form or the UserControl that contains the control, either directly or through a chain of container controls. .NET controls do expose a property named Parent, but it has a different meaning and returns the control's

direct container. You can often replace the Parent property with a call to the FindForm method under .NET, but the FindForm method returns the top-level form even when the control is hosted inside a UserControl, therefore it isn't perfectly equivalent to the Parent property.

VB Migration Partner fully support the Parent property, which works as the original VB6 property also when the current control is hosted inside a UserControl.

Print method

.NET doesn't support this method. Instead, you should use the DrawString method of the System.Drawing.Graphics object. This latter method only works with pixels, so you should convert from the current ScaleMode settings.

VB Migration Partner fully supports the Print method, including all its variations and separators (commas, semicolons, TAB() and SPC() functions, and accounts for the current ScaleMode value.

RightToLeft property

Both VB6 and .NET support the RightToLeft property; however, this property is a Boolean under VB6 and an enumerated value under .NET.

Scale, ScaleX, and ScaleY methods.

.NET only supports control size and position expressed in pixels, therefore these methods aren't supported.

VB Migration Partner fully supports these methods and all the VB6 ScaleMode values, including user-defined coordinate systems, both a design-time and at runtime.

ScaleMode, ScaleLeft, ScaleTop, ScaleWidth, and ScaleHeight properties

.NET only supports control size and position expressed in pixels, therefore these properties aren't supported.

VB Migration Partner fully supports these properties and all the VB6 ScaleMode values, including user-defined coordinate systems, both a design-time and at runtime. It even supports negative values for the ScaleWidth and ScaleHeight properties, so that you can create mirrored graphic motifs as you do in VB6.

SetFocus method

The SetFocus method has been renamed as Focus under .NET; when applied to forms, the SetFocus method maps to the Activate method.

TabIndex property

All controls on a VB6 form have a unique TabIndex value: if you assign a TabIndex value that is already in use, the Visual Basic runtime automatically adjust the TabIndex of other controls so that no duplicate value exists.

The TabIndex property works in a different manner under .NET forms. First, the TabIndex value of a control is relative to the TabIndex value of all other controls inside the same container (e.g. a Panel control). Second, two controls can have the same TabIndex value even if they are located in the same container. This difference in behavior can be confusing and can cause malfunctioning at runtime. For example, a VB6 developer can give a control the input focus simply by assigning zero to its TabIndex property; such code doesn't work after the migration to .NET.

VB Migration Partner's support library exposes the **SetTabIndex6** method, which correctly replicates VB6's behavior.

ToolTipText property

All VB6 controls expose the ToolTipText property. .NET controls don't expose such a property; developers must manually add a ToolTip extender control to the form and then use the ToolTip property that the extender control adds to all controls.

Transparent controls (Label and Image)

Under VB6 you can use a Label or an Image control with a transparent background to create a rectangular "hot spot", so that you can detect mouse activity over a portion of a larger image. Such controls are fully transparent, yet they capture all the mouse events. Many expert VB6 developers have used this feature to design forms with non-standard shapes and appearance, or irregularly-shaped controls.

This feature isn't easily achievable under .NET, because you can't create a control with transparent background. Nevertheless, VB Migration Partner manages to replicate this behavior. All the VB6 code that uses transparent controls to define hot spots works correctly after the conversion to .NET.

Validate event

.NET doesn't support the Validate event, which has been replaced by the Validating and Validated pair of events. In practice, you should replace the VB6 Validate event handler with the .NET Validating event handler.

There is also a subtler behavioral difference, though: when a VB6 form is being closed – either by the end user or programmatically - no Validate event fires for the control that has the input focus. Conversely, the .NET form always fires a Validating event on the control that has the focus when its form is being closed.

VB Migration Partner exactly duplicates the VB6 behavior under .NET.

ZOrder method

All VB6 controls expose the ZOrder method. Under .NET, the ZOrder method is replaced by the BringToFront and SendToBack methods.

Form object

Activate and Deactivate events

Not only does .NET rename these events to Activated and Deactivated, there is also another behavioral difference: the VB6 Activate and Deactivate events fire when moving to another form of the same application, but don't fire if the end user gives the input focus to a different application.

Another significant difference is that VB6 fires no event when a MsgBox or InputBox is executed, whereas .NET fires a Deactivated event for the form that was current, and then an Activated event when the message box is closed.

BackColor property

When you set the BackColor property in VB6 you indirectly clear the form's background, as if you had executed a Cls method.

VB Migration Partner correctly replicates this behavior in converted .NET applications.

BorderStyle property

The VB6 BorderStyle property maps to the .NET FormBorderStyle property. The two properties behave similarly in the two languages, but the correspondence isn't perfect. For example, a VB6 form

has no caption Caption is an empty string and ControlBox is False, or if BorderStyle=vbSBNone; a .NET form has no caption only if FormBorderStyle=None.

Controls collection

The VB6 Controls collection includes all the controls hosted on the form, including controls that are contained in a child control (e.g. the radio buttons contained in a Frame control) and controls that are invisible at runtime (e.g. menus, timers, and common dialogs). By contrast, the .NET Controls collection contains only the visible controls that sit directly on the form's surface.

The VB6Form class in VB Migration Partner's support library exposes the Controls6 collection that behaves like the original VB6 object. This approach guarantees that For Each loops that iterate over the Controls collection delivers the same results as in the original VB6 application.

Controls.Add method

VB6 developers can dynamically create new controls by means of the Add method exposed by the Controls collection of the Form and the UserControl class, as in this example:

```
' Create a new TextBox control named "txtName"

Dim tb As Text

Set tb  = Controls.Add("VB.Text", "txtName")
```

The .NET Controls collection has an Add method, but it has a different meaning and a different syntax. This is the VB.NET code that is roughly equivalent to the above listing:

```
Dim tb As New TextBox

tb.Name = "txtName"

Controls.Add(tb)
```

The Upgrade Wizard isn't capable to convert the Controls.Add method, therefore equivalent code must be inserted manually. VB Migration Partner fully supports the Controls.Add method.

FontTransparent property

.NET forms don't support the FontTransparent property; background text and graphic on a .NET form are always visible and there is no simple way to simulate the VB6 FontTransparent property in a .NET Framework application.

VB6 vs .NET controls

Forms in the VB Migration Partner's support library expose the FontTransparent property, but it always returns True; attempts to assign a different value are ignored (or throw a runtime exception if the VB6Config.ThrowOnUnsupportedMembers property is set to True). This property is marked as obsolete and a migration warning is emitted when the property is referenced in code.

ForeColor property

Assigning the ForeColor property in VB6 simply selects the color that will be used for subsequent Line, Circle, or Print methods. Assigning the ForeColor of a .NET form can actually affect the form's appearance - for example, it can change the color of the border around push buttons.

VB Migration Partner fully duplicates the original VB6 behavior under .NET.

Hide methods

The Hide method fires the FormClosing event under .NET, but it fires no events under VB6, therefore the converted .NET might receive these events at the wrong time.

VB Migration Partner ensures that no events are fired when the Hide method is called, thus preserving the VB6 behavior.

MaxButton, MinButton properties

These properties have been renamed MaximizedBox and MinimizeBox, respectively. There is also another behavioral difference: no maximize or minimize buttons are showed if the BorderStyle property is set to 3-vbFixedDialog under VB6.

MdiChild property

This property has been renamed IsMdiChild under .NET. Moreover, under VB6 this property can be assigned to transform a standard form into an MDI child form, or vice versa. This isn't possible under .NET, because the IsMdiChild property is read-only. (Under .NET you achieve this effect by assigning the form's MdiParent property.)

VB Migration Partner fully supports the writable MdiChild property.

Moveable property

VB6 forms support the Moveable property: if this property is set to True then the end user can't move the form. No similar property exists in the .NET Framework.

VB Migration Partner fully supports this property.

Palette and PaletteMode properties

.NET forms don't support the Palette and PaletteMode properties, and there is no direct way to simulate these VB6 properties.

Forms in the VB Migration Partner's support library expose the Palette and PaletteMode properties, but they always returns Nothing and 0-Halftone, respectively; attempts to assign different value are ignored (or throw a runtime exception if the VB6Config.ThrowOnUnsupportedMembers property is set to True). These properties are marked as obsolete and a migration warning is emitted when these properties are referenced in code.

Popup method (context menu)

.NET doesn't support the PopupMenu method; instead, you must assign a ContextMenuStrip object to the ContextMenuStrip property of the form or the control on whose surface the context menu must appear. You don't need to trap the MouseDown event if the menu has to appear when the end user right-clicks on the control, because this logic is built in the ContextMenuStrip control.

In spite of the deep differences between the VB6 and .NET programming models, VB Migration Partner fully supports the PopupMenu method, except that the flags and boldCommand parameters are ignored: all context menus are left-aligned and no command is shown in boldface. X and Y coordinates, if provided, are correctly converted from twips or whatever the current ScaleMode is.

PrintForm method

.NET doesn't support the PrintForm method.

VB Migration Partner fully supports this method when migrating to either VB 2005 and 2008; no dependence from the Power Pack library is added.

QueryUnload event

This event isn't supported. The closes equivalent under .NET is the FormClosing event. The event handler can detect the reason why the form is being closed by inspecting the CloseReason property of the FormClosingEventArgs object passed to the event handler; however, not all the possible reasons are supported directly.

VB Migration Partner supports the QueryUnload event and deduces the correct value for the reason argument passed to the event handler.

Resize event

.NET supports this event, which can be fired even when the form is still invisible. Conversely, VB6 never fires this event when the form is invisible, therefore a converted .NET application might behave differently from the original VB6 code and a runtime error might occur.

VB Migration Partner fires this event only when the form is visible, hence it guarantees that the original behavior is implemented.

Show method

VB6 can display both modal and modeless forms by means of the Show method (if the method's first argument is nonzero, the form is displayed modally):

```
Dim frm As New Form1

frm.Show  1                    ' displays modal form

frm.Show                       ' displays modeless form
```

.NET uses two different methods for these task: Show for modeless forms and ShowDialog for modal forms:

```
Dim frm As New Form1()

frm.ShowDialog()               ' displays modal form

frm.Show()                     ' displays modeless form
```

The second argument of VB6's Show method allows to assign the owner of the form being displayed; an owned for is automatically closed or minimized when the owner form is closed and minimized, and an owned window is always displayed in front of its owner. (For these reasons, owned forms are often used to implement floating palettes of icons and commands.):

```
Dim frm As New Form1

frm.Show ,Me            ' display a modeless form, make the current form own it
```

In .NET you reach the same effect by means of the AddOwnedForm method:

```
Dim frm As New Form1

Me.AddOwnedForm(frm)           ' make the current form own the new form

frm.Show()                     ' display the form
```

VB Migration Partner fully supports the Show method and all its optional arguments.

Show method (MDI child forms)

If you use the Show method to display an MDI child form under VB6, the method automatically loads and displays the MDI parent form if necessary. This feature allows VB6 developers to select an MDI child form as the application's startup form. Under .NET you must ensure that the MDI parent form is already visible before trying to show an MDI child form.

Unlike other conversion tools, VB Migration Partner takes care of this detail and correctly converts MDI applications.

StartUpPosition property

This property has been renamed StartPosition under .NET. The enumerated values that you can assign to this properties have different values.

TextWidth and TextHeight methods

These methods aren't supported by .NET. Instead, you should use the MeasureString method of the System.Drawing.Graphics object.

VB Migration Partner fully supports these methods and accounts for the current ScaleMode.

ValidateControls method

The VB6's ValidateControls method doesn't return any value, but raises Error 380 if any control fails validation. Under .NET this method has been replaced by the Validate function, which returns False if validation fails (but never raises any error).

Visible property

.NET supports this property, which works as in VB6, except that it is ignored if it assigned the True value and the form is the application's startup form. Under .NET the startup form can be made visible only by means of the Application.Run method.

VB Migration Partner fully supports the VB6 behavior for the Visible property.

MDIForm object

Arrange method

The VB6 Arrange method corresponds to the .NET LayoutMdi method.

AutoShowChildren property

In VB6 you can set the AutoShowChildren property of an MDI form to True, to ensure that all MDI child forms become visible as soon as they are loaded. No such a property exists in .NET and you have to implement this behavior programmatically.

VB Migration Partner fully supports this property.

ActiveForm property

The ActiveForm property isn't supported and corresponds to the ActiveMdiChild property under .NET.

VB Migration Partner fully supports the ActiveForm property; instead of returning a Form object, it returns an Object value, so that statements that rely on late binding work as in the original VB6 application, for example:

```
If TypeOf ActiveForm Is frmDocument Then
    ActiveForm.DisplayDocument    ' this relies on late binding
End If
```

BackColor property

The .NET form exposes the BackColor property, but this property isn't functional if the form works as an MDI container.

The reason for the different behavior is that .NET MDI forms contain an additional child control of type MdiClient. This control fills the visible portion of the form background, thus hiding the "true" form background. As a matter of fact, you can change the background of a .NET MDI form by setting the BackColor property of such an MdiClient control. The following .NET code shows how to implement this technique:

```
Dim mdiClient As MdiClient
For Each ctrl As Control in Me.Controls
    mdiClient = TryCast(ctrl, MdiClient)
    If mdiClient IsNot Nothing Then Exit For
```

```
Next

If mdiClient IsNot Nothing Then mdiClient.BackColor = Color.Red
```

VB Migration Partner fully supports the BackColor property in MDI forms.

Click, DblClick, MouseDown, MouseMove, MouseUp

A VB6 MDI form raises mouse events when the user clicks or moves the mouse on the form's background area. Conversely, no mouse event is triggered by a .NET MDI form.

The reason for the different behavior is that .NET MDI forms contain an additional child control of type MdiClient. This control fills the visible portion of the form background, thus hiding the "true" form background. Mouse activity is routed to this MdiClient control, not the parent form, therefore you should listen to events coming from the MdiClient control. The following code shows a simple technique to solve this problem:

```
Dim mdiClient As MdiClient

For Each ctrl As Control in Me.Controls

    mdiClient = TryCast(ctrl, MdiClient)

    If mdiClient IsNot Nothing Then Exit For

Next

If mdiClient IsNot Nothing Then

    ' ClickHandler and MouseMoveHandler methods must abide by .NET event syntax

    AddHandler mdiClient.Click, AddressOf ClickHandler

    AddHandler mdiClient.MouseMove, AddressOf MouseMoveHandler

End If
```

VB Migration Partner fully supports all mouse events in MDI forms, without the need to manually wire the event handlers to the hidden MdiClient control.

Picture property

The Picture property of a VB6 form broadly maps to the BackgroundImage property of a .NET. However, setting the BackgroundImage property of an MDI Form has no effect under .NET.

The reason for the different behavior is that .NET MDI forms contain an additional child control of type MdiClient. This control fills the visible portion of the form background, thus hiding the "true" form background. As a matter of fact, you can change the background of a .NET MDI form by setting

the BackgroundImage property of such an MdiClient control. The following .NET code shows how to implement this technique:

```
Dim mdiClient As MdiClient

For Each ctrl As Control in Me.Controls

    mdiClient = TryCast(ctrl, MdiClient)

    If mdiClient IsNot Nothing Then Exit For

Next

If mdiClient IsNot Nothing Then mdiClient.BackgroundImage = theImageToBeLoaded
```

VB Migration Partner fully supports the Picture property in MDI forms.

ScrollBars property

The ScrollBars property isn't supported and broadly corresponds to the AutoSize property of a .NET form.

However, notice that there are two important differences you should account for. First, the default value of the ScrollBars property is True in VB6, whereas the default value of the AutoSize property is False under .NET. Second, any assignment to the AutoSize property resets the IsMdiContainer property, therefore you should always save and restore the value of the IsMdiContainer property when setting the AutoSize property, as in this code:

```
Dim saveValue As Boolean = Me.IsMdiContainer

Me.AutoSize = True

Me.IsMdiContainer = saveValue
```

VB Migration Partner fully supports the ScrollBars property, so that the migrated code is guaranteed to work correctly even if the form is accessed in late-bound code.

UserControl object

Many of the differences among the VB6 and .NET UserControl classes are similar to the differences between the VB6 and .NET Form classes. They are explained in depth in the section devoted to the **Form object**. The current section deals with the properties, methods, and events that are specific to the UserControl object.

AccessKeys property, AccessKeyPress event

Under VB6 you can assign the AccessKey property to specify which hotkeys the end user can use to move the input focus to the UserControl; when one of the specified hotkeys is actually pressed, the UserControl object receives the AccessKeyPress event. This mechanism is used by controls that are rendered exclusively by means of graphic and that contain no Label, Button, or other controls that support the Caption property. .NET UserControls don't support a built-in mechanism for trapping hotkeys.

VB Migration Partner partially supports this rarely-used feature; the property retains its value between assignments but has no effect on the runtime behavior. The AccessKeyPress event is also implemented but is never actually fired. In practice, its only purpose is to allow the UserControl to compile correctly.

Ambient property and AmbientChanged event

.NET controls expose neither the Ambient property nor the AmbientChanged event.

When migrating VB6 code that relies on the Ambient property, in most cases you can use the corresponding property of the object returned by the Container property. For example, the Ambient.BackColor and Ambient.Font VB6 properties map to Container.BackColor and Container.Font .NET properties.

This holds true also for the Ambient.ForeColor and Ambient.RightToLeft properties, which map to Container.ForeColor and Container.RightToLeft, respectively. In some cases, the name of the container property is different. For example, the Ambient.DisplayName property maps to Container.Name.

The Ambient.LocaleID VB6 property can be replaced by the CurrentInfo.CurrentUICulture.LCID property under .NET. A few VB6 ambient properties have no corresponding value in the .NET Framework, namely MessageReflect, Palette, ShowGrabHandles, ShowHatching, and UIDead.

VB Migration Partner partially supports the Ambient property, which returns an instance of the VB6Ambient class. This class exposes all the properties of the VB6 Ambient object, so that no compilation errors occurs. However, only a subset of the Ambient properties are actually supported, namely: BackColor, DisplayAsDefault, DisplayName, ForeColor, Font, ForeColor, LocaleID, RightToLeft, ScaleUnits, and UserMode.

In addition, the AmbientChanged event is supported only for the BackColor, ForeColor, and Font properties.

AsyncRead and CancelAsyncRead methods, AsyncReadComplete and AsyncReadProgress events

Under VB6 a UserControl can read a property asynchronously using the AsyncRead method, or cancel an asynchronous read operation by means of the CancelAsyncRead method. When the read

operation is progressing one or more AsyncReadProgress events are fired; when the read operation is complete, an AsyncReadComplete event is fired. This feature is especially useful for UserControls that are meant to be hosted inside an HTML page, where the actual value of the property – for example, an image – must be read via the Internet.

.NET doesn't directly support asynchronous properties. It is indeed possible to perform any read operation in asynchronous way, but many manual edits should be performed.

VB Migration Partner supports these members, but no asynchronous behavior is implemented. The AsyncRead method reads a property synchronously and then fires the AsyncReadComplete event. The CancelAsyncRead method does nothing. The AsyncReadProgress event is never fired.

CanPropertyChange and PropertyChanged methods

Under VB6 a UserControl should call the CanPropertyChange method before assigning a different value to a property, and the PropertyChanged method after changing the value of a property. These methods are intercepted by the VB6 runtime to implement data-bound UserControls, among other things.

.NET doesn't support anything similar to these methods. Implementing a .NET UserControl that supports data-binding requires many manual changes.

VB Migration Partner supports these methods. In current implementation, the CanPropertyChange method always return True, whereas the PropertyChanged method does nothing. The main purpose of these methods is to avoid compilation and runtime errors in migrated .NET applications.

ContainedControls collection

In .NET it isn't possible to discern between controls that were placed on the usercontrol's surface at design-time (when the usercontrol is defined) and those that were added after placing the usercontrol on a .NET form's surface.

VB Migration Partner partially supports the ContainedControls collection; however, this collection always contains the same items as the standard Controls collection.

ContainerHwnd property

This property isn't supported under .NET. You can replace it with code that returns the Handle property of the container, something like

```
Dim handle As Integer = DirectCast(Me.Parent, Control).Handle.ToInt32()
```

DataMember property

This property isn't supported under .NET.

VB Migration Partner implementation of this property as an empty member that always returns Nothing.

EnterFocus, ExitFocus events

Under VB6 these events fire when the input focus enters and exits the UserControl, respectively. These events correspond to the Enter and Leave events in .NET.

EventsFrozen method

Code running inside a VB6 UserControl can use the EventsFrozen method to ensure that it is safe to trigger an event. .NET doesn't support this method.

VB Migration Partner partially supports the EventsFrozen method. This method returns True when the UserControl is being loaded and in a few other occasions. However, it isn't guaranteed to be perfectly equivalent to the original VB6 method.

Extender property

Under VB6 the Extender property returned a reference to a special "extender" wrapper that was built the first time an ActiveX was dropped on a form's surface. The Extender object had typically all the members of the original ActiveX plus all the members added by the container (i.e. the VB6 IDE), such as Left and Top. The definition of the extender object was stored in a file with .OCA extension.

The Extender property isn't supported under .NET, because no wrapper is created when a UserControl is dropped on a Windows Forms surface. You can usually delete the reference to the Extender property. For example, the following VB6 code:

```
Me.Extender.Left = Me.Extender.Left + 100
```

can be translated into .NET as

```
Me.Left = Me.Left + 100
```

VB Migration Partner supports the Extender property, which simply returns a reference to the UserControl itself.

HitBehavior property, HitTest event

VB6 developers can use the HitBehavior property and the HitTest event to implement non-rectangular UserControls or controls with "holes" inside them. .NET doesn't support this property

and requires completely different programming techniques to implement such irregular UserControls, for example using Windows region objects.

VB Migration Partner supports this property; it always returns the value 1; attempts to assign a different value are ignored (or throw an exception if the **VB6Config.ThrowOnUnsupportedMembers** property is True). The HitTest event is implemented but is never fired.

HyperLink property

VB6 developers can use the Hyperlink property to manage jumps to a different HTML page, when the UserControl is hosted inside the browser. This functionality isn't available under .NET.

VB Migration Partner implements this property but it always returns Nothing. In practice, it only serves to avoid errors when compiling the UserControl.

KeyPreview property

This property isn't available under .NET. The easiest way to simulate its behavior is to intercept the KeyDown, KeyUp, and KeyPress events of all the controls on the UserControl, using code like this:

```
For Each ctrl As Control In Me.Controls

    AddHandler ctrl.KeyDown, AddressOf Control_KeyDown

    AddHandler ctrl.KeyUp, AddressOf Control_KeyUp

    AddHandler ctrl.KeyPress, AddressOf Control_KeyPress

Next
```

VB Migration Partner fully supports the KeyPreview property; if the input focus is currently on a UserControl's child control this property is True, any keyboard operation fires an event in the UserControl class first, and then in the child control that currently has the focus.

Locked and Size properties

If a VB6 user control exposes a property named Locked, such property should be renamed when the user control is converted to VB6, because it would interfere with Visual Studio 2005's designer. Similarly, if the user control exposes a member named Size, it should be renamed because it would confuse Visual Studio 2005's designer.

VB Migration Partner renames such members as **Locked6** and **Size6**, respectively.

Hide and Show events

The .NET UserControl class doesn't support these events. Instead, you should trap the VisibleChanged event to detect the moment when the control becomes visible or invisible.

VB Migration Partner fully supports the Hide and Show events.

InvisibleAtRuntime property

VB6 UserControl support this property; by setting this property to True you make the control always invisible at runtime, as is the case of components such as the Timer.

.NET doesn't directly support this property. A UserControl that is always invisible at runtime should be implemented as a class that inherits from System.ComponentModel.Component instead of System.Windows.Forms.UserControl.

VB Migration Partner supports this property and generates a UserControl whose Visible property is always False. Any attempt to set this property to True – either at design-time or runtime – throws an exception.

Palette and PaletteMode properties

.NET user controls don't support the Palette and PaletteMode properties, and there is no direct way to simulate these VB6 properties.

The VB6UserControl class in the VB Migration Partner's support library exposes the Palette and PaletteMode properties, but they always returns Nothing and 0-Halftone, respectively; attempts to assign different value are ignored (or throw a runtime exception if the **VB6Config.ThrowOnUnsupportedMembers** property is set to True). These properties are marked as obsolete and a migration warning is emitted when these properties are referenced in code.

ParentControls collection

Under VB6 this property returns the controls contained in the form that contains the current UserControl. The ParentControlsType property exposed by this collection permits to determine whether the actual controls or the Extender wrappers of those controls should be returned.

.NET doesn't support anything similar to the ParentControls collection. VB Migration Partner returns a collection that includes all the controls contained in the parent form, including the form itself. The returned collection exposes the ParentControlsType property, but assigning a value to this property has no effect on the items in the collection itself.

Picture property

.NET doesn't support the Picture property. You can achieve the same effect by using the BackgroundImage property.

PropertyPages property

VB6 developers can use the PropertyPages to associate a given PropertyPage object to a UserControl. The .NET Framework doesn't support property pages, hence this property isn't available under .NET.

VB Migration Partner supports the PropertyPages property, but it always returns an empty strings and all assignments to it are ignored. In practice, it only serves to ensure that the UserControl compiles with no errors.

Runtime-only properties

A VB6 user control can include one or more properties that are available only at runtime and that shouldn't appear in the property window. By convention, such a property should raise error 382 or 387 in the Property Get procedure, as in:

```
Property Get hWnd() As Long

    If Not Ambient.UserMode Then Err.Raise 382

    ' …

End Property
```

During the conversion to .NET, such properties should be marked with the Browsable(False) attribute to hide them in the property window, and with the DesignSerializationVisibility(DesignerSerializationVisibility.Hidden) attribute to avoid persistence in the designer. VB Migration Partner correctly detects this situation and generates these attributes as needed.

Menu control

Top-level menus, drop-down menus, and separators

.NET uses three different objects to render the VB6 Menu object.

- Top-level menus are translated as MenuStrip objects; there is only one MenuStrip object in each form, whose child controls are drop-down menus.

- Drop-down menus are translated as ToolStripMenuItem objects.
- Separator bars are translated as ToolStripSeparator objects. (A VB6 separator is a plain menu object whose Caption property is set to the "-" character.)

VB Migration Partner handles this important difference internally and generates .NET code that behaves like the original VB6 code in virtually all cases.

Caption property

The Caption property maps to the Text property under .NET, but there is an important difference. If you assign a "-" (dash) character to the Caption property, you actually transform the menu element into a menu separator under VB6, even at runtime. Nothing similar happens under .NET.

VB Migration Partner can correctly handle these assignments and correctly transforms a menu element into a separator (or vice versa), depending on the value being assigned to the Caption property.

Click event

.NET menus support the Click event; however, in VB6 the Click event of dropdown menu fires immediately after the menu becomes visible, whereas .NET fires the Click event before the dropdown menus becomes visible. For this reason, event handlers generated by the Upgrade Wizard and other converters based on the same engine might be executed at the wrong time and cause runtime errors or bogus results. If you need to handle the Click event of dropdown menus you should use the DropDownOpened event instead.

VB Migration Partner is aware of this difference and fires the Click event at the right time even for dropdown menus.

NegotiatePosition property

The NegotiatePosition property isn't supported by .NET menus.

VB Migration Partner supports this property, so that the .NET code always compiles correctly. However, this property doesn't affect the menu's appearance or behavior.

Shortcut property

The VB6 Shortcut property maps to the ShortcutKeys property under .NET.

Visible property

The .NET MenuStrip, ToolStripMenuItem, and ToolStripSeparator objects all support the Visible property. However, there are two important differences. First, when a top-level menu becomes visible all the controls on the form are shifted down to make room for the menu; likewise, when a top-level menu becomes invisible, all controls on the form are shifted up. Second, the Visible property of ToolStripMenuItem and ToolStripSeparator objects return False both if you set it to False or if you set the parent menu's Visible property to False.

VB Migration Partner automatically accounts for all these minor differences, so that the generated .NET is guaranteed to work like the original VB6 code.

Label control

Alignment property

The Alignment property has been renamed as TextAlign. The VB6 control always aligns text to the top and you can only decide whether it's left justified, right justified, or centered. The .NET control also allows to defined vertical alignment.

Appearance and BorderStyle properties

The VB6 Appearance property has been dropped and its effect has been integrated into the VB.NET BorderStyle property, according the following rule: if BorderStyle is equal to 0-None under VB6 then the control has no border and the Appearance property is ignored; if BorderStyle is equal to 1-FixedSingle and Appearance is equal to 0-Flat under VB6, then the .NET BorderStyle is set to 1-Flat; if BorderStyle is equal to 1-FixedSingle and Appearance is equal to 1-ThreeD under VB6, then the .NET BorderStyle is set to 2-Fixed3D.

VB Migration Partner supports all the combinations of Appearance and BorderStyle properties.

AutoSize property

The default value for the AutoSize property is False under VB6; the value of the AutoSize property of a .NET Label control dropped on a form is True.

BackStyle property

.NET Label controls don't support the BackStyle property and there is no simple way to implement it. The background portion of a .NET Label control is always opaque.

If the Label control has a nonempty caption, VB Migration Partner simulates the BackStyle property by setting the BackColor property to the background color of the container controls when the BackStyle property is set to 0-Transparent. If the Label control has an empty caption, VB Migration Partner improves the simulation by making the control truly transparent while it continues to receive mouse events.

Caption property

The Caption property is renamed as Text.

There is also an additional minor difference: under VB6, any carriage-return (ASCII 13) character splits text on two distinct lines; under .NET you need a full CR-LF pair (that is, ASCII 13 + ASCII 10) to split text on multiple lines. VB Migration Partner correctly accounts for this detail.

Change event

The Change event has been renamed as TextChanged.

WordWrap property

.NET Label controls don't support the WordWrap property, because they always wrap long lines of text and there is no way to avoid it. However, you can simulate the VB6 behavior by changing the Height property, so that the wrapped lines aren't visible.

The Label control in VB Migration Partner's support library exposes the WordWrap control, but it always returns True. Any attempt to assign a different value is ignored (or throws an exception if the **VB6Config.ThrowOnUnsupportedMembers** property is True.)

TextBox control

Alignment property

The Alignment property has been renamed as TextAlign.

Appearance and BorderStyle properties

The VB6 Appearance property has been dropped and its effect has been integrated into the .NET BorderStyle property, according the following rule: if BorderStyle is equal to 0-None under VB6 then the control has no border and the Appearance property is ignored; if BorderStyle is equal to 1-FixedSingle and Appearance is equal to 0-Flat under VB6, then the .NET BorderStyle is set to 1-Flat; if BorderStyle is equal to 1-FixedSingle and Appearance is equal to 1-ThreeD under VB6, then the .NET BorderStyle is set to 2-Fixed3D.

VB Migration Partner supports all the combinations of Appearance and BorderStyle properties.

Change event

The Change event has been renamed as TextChanged.

Locked property

This property has been renamed as ReadOnly, because the Locked property is used by the Visual Studio's form designer.

VB Migration Partner supports the Locked property, so that code that uses this property works after the migration even if the control is accessed in late-bound mode.

PasswordChar property

This property is of String type under VB6 and of Char type under .NET. This causes a minor problem because you can't assign an "empty char" to the PasswordChar property in .NET; to disable the password feature in .NET you must assign an ASCII 0 char to this property.

For the highest compatibility, VB Migration Partner supports a PasswordChar property of String type. If a multi-char string is assigned to this property, only the first character is used to define the password char, as in VB6. Notice that a different property – named **PasswordChar_**, with a trailing underscore) is used to assign the property in the Properties window.

ScrollBars property

The VB6 ScrollBars property indirectly affects word wrapping: if ScrollBars is equal to 2-Vertical, then long text lines are automatically wrapped to the next line. To achieve the same behavior under .NET you have to explicitly set the WordWrap property to True.

VB Migration Partner correctly replicates the VB6 behavior under .NET.

SelStart, SelLength, and SetText properties

The VB6's SetStart, SelLength, and SetText properties have been renamed as SelectionStart, SelectionLength, and SelectionText under .NET, respectively.

There is also a minor difference in the behavior: the SelStart property indirectly resets the SelLength property to 0 and the SelectionText to an empty string, and ensures that the insertion point is visible inside the TextBox control. In .NET you have to invoke the **ScrollToCaret** method to bring the insertion point into the control's visible area.

VB Migration Partner correctly replicates the VB6 behavior under .NET.

Text property

If you assign a string that is longer than the current value of the MaxLenght property, VB6 automatically trims the value to MaxLength characters, whereas .NET doesn't.

VB Migration Partner correctly replicates the VB6 behavior under .NET.

CheckBox and OptionButton controls

Alignment property

The VB6 Alignment property is missing in .NET and is replaced by the CheckAlign and TextAlign properties.

Appearance property

The Appearance property has been renamed as FlatStyle.

Click event

The Click event is supported under .NET; however this event doesn't fire when the control's Value is changed via code. Also, it is advisable not to use the MouseClick event, because such an event doesn't fire if the control's value is changed via the keyboard. The most reliable way to replicate the original VB6 behavior is using the CheckStateChange event.

VB Migration Partner correctly handles all these differences. Converted .NET code works exactly as in the original VB6 program.

Picture and Style properties

The VB6 Picture property maps to the .NET Image and Appearance properties, but the Style property has no .NET counterpart. (In VB6 the Picture property is used only if Style=1-vbButtonGraphical.) Therefore, you should assign the .NET Image property only if you really mean to display an image.

In addition to displaying an image, the VB6 Style property affects the position of the control's caption: in standard buttons the caption is centered vertically, whereas in graphical controls it is near to the bottom border. You can reach the same effect by assigning the .NET TextAlign property.

VB Migration Partner fully supports all the possible combinations of these two properties.

TabStop behavior (OptionButton only)

VB6 and .NET differ in how a radio button is given the focus when the end users presses the Tab key. Under VB6, when the end user tabs into a group of radio button, the button whose Value is True receives the focus. Conversely, under .NET the radio button with the lowest value of the TabIndex property receives the focus, regardless of whether it is checked or not.

VB Migration Partner accounts for this difference and correctly mimics the VB6 behavior.

Value property

This property isn't supported and maps to the Checked property.

In addition, there is a minor difference in what happens when you set the property to False in a radio button that is currently checked. In a VB6 OptionButton control, setting this property to False fires no Click events; in a .NET RadioButton control this assignment causes a Click event. (Note that you typically set the Value property to False when you need to display a group of unchecked buttons.)

To fully mimic the VB6 behavior, VB Migration Partner provides a ResetValue method, which changes the value of the Value property without firing any event:

```
optChoice.ResetValue(False)
```

CommandButton control

Appearance property

The Appearance property has been renamed as FlatStyle.

VB Migration Partner still supports the Appearance property, so that migrated code works as expected even if the control is accessed in late-bound mode.

Cancel and Default properties

The .NET Button control doesn't support the Cancel or Default properties. Instead, you select which the default and cancel buttons on a form are by setting the form's AcceptButton and CancelButton properties. Therefore, the following VB6 code:

```
btnCancel.Cancel = True

btnOK.Default = True
```

can be converted to .NET as follows:

```
Me.CancelButton = btnCancel

Me.AcceptButton = btnOK
```

All VB6 converter tools on the marker correctly migrate forms containing default and cancel buttons. However, only VB Migration Partner supports these properties when assigned in late-bound mode or when the button is located inside a usercontrol.

Click event

.NET exposes both a Click and a MouseClick event. You should always use the Click event in migrated applications, because the MouseClick event doesn't fire if the button is operated with the Space key.

DownPicture and DisabledPicture properties

These properties have no .NET counterpart. You can achieve the same effect in .NET by monitoring the mouse activity on the button (for DownPicture) and changes to the Enabled property (for DisabledPicture).

VB Migration Partner fully supports these properties and correctly display buttons in down or disable state.

MaskColor and UseMaskColor properties

These properties have no .NET counterpart and can't be easily implemented.

VB Migration Partner supports both properties. If UseMaskColor is equal to True, the button displays an image where all pixels whose color is equal to MaskColor are rendered as transparent pixels.

Picture and Style properties

The VB6 Picture property maps to the .NET Image property, but the Style property has no .NET counterpart. (In VB6 the Picture property is used only if Style=1-vbButtonGraphical.) Therefore, you should assign the .NET Image property only if you really mean to display an image.

In addition to displaying an image, the VB6 Style property affects the position of the button's caption: in standard buttons the caption is centered vertically, whereas in graphical buttons it is near to the bottom border. You can reach the same effect by assigning the .NET TextAlign property.

VB Migration Partner fully supports all the possible combinations of these two properties.

Value property

In VB6 you can programmatically set the Value property of a CommandButton control to indirectly fire the control's Click event. You reach the same effect under .NET by invoking the PerformClick method.

VB Migration Partner fully supports the Value property.

ListBox control

AddItem method

The AddItem method isn't supported and can be rendered using the Add or Insert methods of the Items collection, depending on whether the method receives an index argument.

Clear method

The Clear method isn't supported under .NET; you can use the Items.Clear method instead.

Click event

The Click event in the .NET control doesn't fire when a new element is selected via code or by using the keyboard. The actual .NET counterpart of this event is the SelectedIndexChanged event.

Columns property

The Columns property isn't supported. You can approximate its effect by opportunely assigning the MultiColumn and ColumnWidth properties, which requires that you correctly calculate the width of each column.

VB Migration Partner fully supports this property.

DblClick event

The DblClick event maps to the DoubleClick event. However, you should also take into account that under VB6 no event fires if the mouse isn't clicked on an element.

VB Migration Partner fully supports the DblClick event; as in VB6 no event fires if the mouse isn't clicked on an element.

ItemCheck event

The ItemCheck event is supported but has a different syntax. Moreover, under VB6 this event fires when the checked state has been already assigned, whereas in .NET it fires before the assignment occurs. The code inside a VB6 event handler can reset the checked state of an element by simply assigning it, whereas code in a .NET event handler must assign the new state to the NewValue property of the ItemCheckEventArgs object passed as an argument to the event handler.

VB Migration Partner fully supports this event and automatically accounts for all the behavioral differences between VB6 and .NET.

ItemData property

The ItemData property isn't supported by the .NET ListBox control. Code migrated by the Upgrade Wizard and other converters based on the same engine uses helper methods to render this property.

Interestingly, the items of a .NET ListBox control can be objects of any type, not just strings. If an object is used, the ListBox control displays whatever the object's ToString method returns. If you need to associate data with an item, you can just create a class that contains two items, as in this example:

```
Class ListItem
```

```
        Public ItemText As String    ' what is displayed
        Public ItemData As Object    ' associated data

        Public Overrides Function ToString() As String
            Return ItemText
        End Function
    End Class
```

VB Migration Partner fully supports this property; migrated code works correctly even if the control is accessed in late-bound mode.

List property

The List property isn't supported; it can be rendered using the Items collection. Notice that the VB6 List property returns an empty string when the index is out of valid range, whereas you get an exception if you attempt to access a nonexisting element of the .NET Items collection. Along the same line, in VB6 you can create a new element by assigning the List property of the first available index:

```
List1.List(List1.ListCount) = "new item"
```

This feature isn't supported by the Items collection.

VB Migration Partner fully support this property, including all its quirks and undocumented behaviors.

ListCount property

The ListCount property isn't supported under .NET; you can use the Items.Count property instead.

ListIndex property

The ListIndex property isn't supported under .NET; you can use the SelectedIndex property instead.

MultiSelect property

The MultiSelect property isn't supported and must be translated using the SelectionMode property.

NewIndex property

The NewIndex property isn't supported; it broadly corresponds to the value returned by the Add method of the Items collection.

VB Migration Partner fully support this property, so that migrated code is guaranteed to work as intended.

RemoveItem method

The RemoveItem method isn't supported and can be rendered using the RemoveAt method of the Items collection. There are other differences in behavior to take into account, though. For example, under VB6 the ListIndex property points to the element prior to the one being removed if the ListBox supports multiple selections, whereas in .NET the ListIndex property is always set to -1.

VB Migration Partner support this method and perfectly replicates all its quirks, thus no adjustments are necessary after the migration.

Scroll event

The Scroll event isn't supported under .NET and it can't easily simulated. For example, you can approximate it by hooking the SelectedIndexChanged event, but you wouldn't trap the cases when the user scrolls the listbox contents without changing the current element. The only reliable way to get a notification when the ListBox control is scrolled is by means of Windows subclassing.

VB Migration Partner fully supports this event.

SelCount property

The SelCount readonly property isn't supported; it can be rendered by means of the SelectedItems.Count property.

Selected property

The Selected property isn't supported; it can be rendered by means of the GetSelected method for simple ListBox controls, or the GetItemChecked and SetItemChecked methods for ListBox controls with checkboxes.

Style property

The .NET ListBox control doesn't support the Style property. ListBox controls with the Style property set to 1-vbListBoxCheckbox must be migrated to the CheckedListBox control instead of the ListBox control.

VB Migration Partner migrate a ListBox with Style=1 to the most appropriate .NET control.

ComboBox control

AddItem method

The AddItem method isn't supported and can be rendered using the Add or Insert methods of the Items collection, depending on whether the method receives an index argument.

Change event

The Change event maps to the TextChanged event, but in the case of the ComboBox controls there are several differences between VB6 and .NET. The VB6 Change event fires only when the text in the edit area is modified, whereas the .NET TextChanged event also fires when a different item is selected in the list area, when the AddItem is called, or when the value of the current element is assigned programmatically via the List property.

For this reason, the TextChanged event handler generated by the Upgrade Wizard and other converters based on the same engine may fire at the wrong time and cause bogus results or runtime errors. This never happens with VB Migration Partner, which perfectly mimics the VB6 behavior.

Clear method

The Clear method isn't supported under .NET; you can use the Items.Clear method instead.

Click event

The Click event in the .NET control doesn't fire when a new element is selected via code or by using the keyboard. The actual .NET counterpart of this event is the SelectedIndexChanged event.

ItemData property

The ItemData property isn't supported by the .NET ComboBox control.

Interestingly, the items of a .NET ComboBox control can be objects of any type, not just strings. If an object is used, the ComboBox control displays whatever the object's ToString method returns. If you need to associate data with an item, you can just create a class that contains two items, as in this example:

```
Class ListBoxItem

    Public ItemText As String    ' what is displayed

    Public ItemData As Object    ' associated data


    Public Overrides Function ToString() As String

        Return ItemText

    End Function

End Class
```

VB Migration Partner fully supports this property; migrated code works correctly even if the control is accessed in late-bound mode.

List property

The List property isn't supported; it can be rendered using the Items collection. Notice that the VB6 List property returns an empty string when the index is out of valid range, whereas you get an exception if you attempt to access a nonexisting element of the .NET Items collection. Along the same line, in VB6 you can create a new element by assigning the List property of the first available index:

```
List1.List(List1.ListCount) = "new item"
```

This feature isn't supported by the Items collection.

VB Migration Partner fully support this property, including all its quirks and undocumented behaviors.

ListCount property

The ListCount property isn't supported under .NET; you can use the Items.Count property instead.

ListIndex property

The ListIndex property isn't supported under .NET; you can use the SelectedIndex property instead.

Locked property

Under VB6 the Locked property allows you to create a ComboBox with an edit area whose contents can't be modified. This feature can't be easily replicated under .NET and can only be approximated by creating a ComboBox control with DropDownList style.

VB Migration Partner supports the Locked property using the technique just described.

NewIndex property

The NewIndex property isn't supported; it broadly corresponds to the value returned by the Add method of the Items collection.

VB Migration Partner fully support this property, so that migrated code is guaranteed to work as intended.

RemoveItem method

The RemoveItem method isn't supported and can be rendered using the RemoveAt method of the Items collection. There are other differences in behavior to take into account, though. For example, under VB6 the ListIndex property points to the element prior to the one being removed if the ComboBox supports multiple selections, whereas in .NET the ListIndex property is always set to -1.

VB Migration Partner support this method and perfectly replicates all its quirks, thus no adjustments are necessary after the migration.

Scroll event

The Scroll event isn't supported under .NET and it can only be approximated by trapping the SelectedIndexChanged event and by subclassing the window that is created when the dropdown list appears.

VB Migration Partner fully supports this event.

SelStart, SelLength, and SetText properties

The VB6's SetStart, SelLength, and SetText properties have been renamed as SelectionStart, SelectionLength, and SelectionText under .NET, respectively.

VB Migration Partner fully supports the SelStart, SelLength, and SelText properties, so that migrated code works as intended even if the control is accessed in late-bound mode.

SetFocus method

The VB6 SetFocus method maps to the .NET Focus method. In addition, when used on a ComboBox control, the SetFocus method selects the entire contents of the edit area.

Text property

There are a few minor differences in how VB6 and .NET handle the Text property. Under VB6, assigning the Text property raises a runtime error if the control is a dropdown list and you attempt to assign a value different from the value of the current element. No error ever occurs in .NET.

VB Migration Partner perfectly mimics the VB6 behavior.

TopIndex property

The .NET ComboBox control doesn't support the TopIndex property. You can read and modify the index of the topmost element by sending a CB_GETTOPINDEX and CB_SETTOPINDEX message to the control using the DefWndProc protected method.

VB Migration Partner fully supports this property. No manual edits of the migrated code are needed.

Frame control

BorderStyle property

Likewise, VB Migration Partner converts a Frame control into a VB6Frame control (which inherits from GroupBox) or a VB6FrameNoBorder control (which inherits from Panel).

The problem is that the .NET GroupBox control always has a visible border, therefore you can't later hide the border by assigning 0 to the BorderStyle property. In .NET projects created by the Upgrade Wizard and other converters based on the same engine this assignment causes a compilation error; in .NET projects created by VB Migration Partner this assignment compiles fine but generates a warning coming from the Obsolete attribute associated with the BorderStyle property.

If the Frame control is borderless at design-time – and is therefore converted into a Panel or a VB6FrameNoBorder control, which inherits from Panel – then assignments to the BorderStyle compile and run fine. However, the visual effect is slightly different from VB6, because the border runs along the control's edge (in VB6 it runs a few pixels away from the edge). Moreover, the Panel control and the VB6FrameNoBorder control can't display a caption, therefore any assignment to the Text or Caption property is ignored.

PictureBox control

Appearance and BorderStyle properties

The VB6 Appearance property has been dropped and its effect has been integrated into the .NET BorderStyle property, according the following rule: if BorderStyle is equal to 0-None under VB6 then the control has no border and the Appearance property is ignored; if BorderStyle is equal to 1-FixedSingle and Appearance is equal to 0-Flat under VB6, then the .NET BorderStyle is set to 1-Flat; if BorderStyle is equal to 1-FixedSingle and Appearance is equal to 1-ThreeD under VB6, then the .NET BorderStyle is set to 2-Fixed3D.

AutoSize property

The .NET PictureBox control doesn't expose this property and you must manually resize the image. Notice that, if AutoSize is True and you are assigning an image in Metafile format, you can reproduce the VB6 behavior by assigning the Image to the BackgroundImage property and by assigning ImageLayout.Stretch to the BackgroundImageLayout property.

VB Migration Partner fully supports this property. No manual adjustments of other properties are necessary.

Change event

The Change event isn't supported by the .NET PictureBox. If you display an image by assigning it to the BackgroundImage property, you can handle the BackgroundImageChanged event instead.

VB Migration Partner fully supports the Change event. No modifications in code are necessary.

Container functionality

The main difference between the VB6 and .NET versions of the PictureBox control is that the latter can't work as a container for other controls. If the original application used the PictureBox as a control container, the .NET code generated by the Upgrade Wizard and other converters based on the same engine needs to be heavily modified by hand.

VB Migration Partner overcomes this limitation and correctly migrates PictureBox controls that work as control container.

KeyDown and KeyUp events

These events are supported under VB, however there is a minor difference in how arrow keys are processed. In VB6 these keys are treated like any other keys and are notified to the KeyDown and KeyUp events. Under .NET these keys can move the input focus to a different control and might not notified to the events.

VB Migration Partner ensures that arrow keys be processed exactly as in VB6, thus no code adjustment is necessary.

Picture property

The Picture property isn't supported by the .NET PictureBox control. It must be replaced by the Image property or the BackgroundImage property.

Image control

Picture property

The Picture property isn't supported and maps to the Image property.

Stretch property

The Stretch property isn't supported and maps to the SizeMode property.

HScrollBar and VScrollBar controls

Change event

The Change event isn't supported and has been replaced by the ValueChanged event.

LargeChange property

.NET scrollbars deal with this property in a very special way: if the value of this property is higher than 1 and isn't a divisor of the Maximum value, then the end user can't click below or above the thumb indicator to scroll the bar to a value higher than Max-LargeChange+1. For example, if Maximum=255 and LargeChange=10 then the scrollbar can't be scrolled to a value other than 250. No range limit is enforced if clicking on the scrollbar arrows, though.

This limitation is inherent to the .NET Framework and VB Migration Partner can only attempt to make the problem less serious. The easiest way to ensure that the converted .NET application works well is forcing the LargeChange property to be equal to 1 and ignoring all assignments to it. The neat result is that the end user can move the thumb indicator up to its maximum position, even if the scrollbar will react more slowly.

This feature can be controlled by means of the **IgnoreLargeChange** property, whose default value is True. If you set this value to False, values assigned to the LargeChange property become immediately effective and the scrollbar will behave similarly to the original VB6 program.

Min and Max properties

The Min and Max properties aren't supported and map to Minimum and Maximum properties, respectively.

Even more important is that VB6 supports scrollbars whose minimum value is higher than its maximum value, a setting that is useful to implement scrollbars that work in nonstandard way (for example, a vertical scrollbar whose bottommost position corresponds to the minimum value). This arrangement isn't available for .NET scrollbars, as this code demonstrates:

```
HScroll1.Minimum = 1

HScroll1.Maximum = 100

HScroll1.Minimum = 200              ' higher than Maximum!

Debug.WriteLine(HScroll1.Maximum)   ' displays "200"
```

Converting such scrollbars to .NET using the Upgrade Wizard and other converters based on the same engine can be quite cumbersome. VB Migration Partner fully supports the Min and Max properties and, above all, supports the VB6 behavior and guarantees that functional equivalence is preserved.

Scroll event

The Scroll event is supported, with a caveat: in VB6 the Value property has been already updated when the event fires, whereas in .NET the Value property contains the previous value when the event fires. You can learn the updated Value property by querying the NewValue property of the ScrollEventArg object passed as an argument to the event handler.

VB Migration Partner fully supports this event and automatically accounts for this minor difference: when the Scroll event fires, the Value property has been already updated with the new value.

DriveListBox control

Click event

The Click event in the .NET control doesn't fire when a new element is selected via code or by using the keyboard. The actual .NET counterpart of this event is the SelectedIndexChanged event.

Scroll event

The Scroll event isn't supported under .NET and it can only be approximated by trapping the SelectedIndexChanged event and by subclassing the window that is created when the dropdown list appears.

VB Migration Partner fully supports this event.

TopIndex property

The .NET DriveListBox control doesn't support the TopIndex property. You can read and modify the index of the topmost element by sending a CB_GETTOPINDEX and CB_SETTOPINDEX message to the control using the DefWndProc protected method.

VB Migration Partner fully supports this property. No manual edits of the migrated code are needed.

DirListBox control

List, ListCount, and ListIndex properties

The Upgrade Wizard converts these properties into the DirList, DirListCount, and DirListIndex properties of the Microsoft.VisualBasic.Compatibility.VB6.DirListBox control.

VB Migration Partner supports the original List, ListCount, and ListIndex properties, to ensure that migrated code works as intended even if the control is accessed in late-bound mode.

Scroll event

The Scroll event isn't supported under .NET and it can only be approximated by trapping the SelectedIndexChanged event and by subclassing the window that is created when the dropdown list appears.

VB Migration Partner fully supports this event.

FileListBox control

Scroll event

The Scroll event isn't supported under .NET and it can only be approximated by trapping the SelectedIndexChanged event and by subclassing the window that is created when the dropdown list appears.

VB Migration Partner fully supports this event.

Timer control

Interval property

Under VB6 you can assign zero to this property to disable the timer; under .NET such an assignment throws an exception.

VB Migration Partner fully supports the Interval property and replicates the VB6 behavior when zero is assigned to it.

Timer event

This event has been renamed as Tick in .NET.