VB Migration Partner Support Library

This document summarizes all the known functional differences between VB Migration Partner's support library and the original Visual Basic 6's runtime library.

Accelerators

Migrated projects support accelerators (underlined characters in the caption of buttons and label controls) with a different behavior at run-time: in .NET these characters aren't underlined until the Alt key is pressed. TabStrip and SSTab controls partially support this feature.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=364
http://www.vbmigration.com/detknowledgebase.aspx?Id=90
http://www.vbmigration.com/detknowledgebase.aspx?Id=624

ActiveTreedPlus controls

VBLibrary hides most of the differences – but not all of them - between the VB6 and the .NET versions of the controls included in Infragistics' ActiveThreedPlus library. All ActiveTreedPlus controls are implemented, with these differences:

- **All controls:** the following properties are not implemented and are marked as obsolete: ActiveColors, AutoSize, BevelInner, BevelOuter, BevelWidth, BorderWidth, CaptionStyle, DataBindings, FloodFillStyle, Font3D, MarqueeDelay, MarqueeDirection, MarqueeScrollAmount, MarqueeStyle, Object, Outline, Picture, PictureAlignment, PictureAnimationCount, PictureAnimationDelay, PictureAnimationEnabled, PictureFrame, PictureFrames, PictureBackgroundUseMask, RoundCorners, ShadowStyle, ShadowColor, WindowLess.
- **All controls:** the following events aren't supported and are marked as obsolete: MarqueCycleBegin, MarqueeCycleEnd, PictureFrameChanged.
- **SSCheck and SSOption controls :** the BackStyle, CheckBoxGraphics, CheckBoxMaskColor, CheckBoxUseMask, OptionBtnGraphics, OptionBtnMaskColor, and OptionBtnUseMask properties are not implemented and are marked as obsolete.
- **SSCommand control:** the AutoRepeat, BackStyle, ButtonStyle, Outline, PictureDisabled, PictureDisabledFrames, PictureDn, PictureDnFrames, Shape, and ShapeSize properties are not implemented and are marked as obsolete.
- **SSFrame control:** the Alignment, BackStyle, and PictureBackgroundUseMask properties are not implemented and are marked as obsolete.
- **SSPanel control:** the Outline, Picture, PictureAlignment, and PictureBackgroundUseMask properties are not implemented and are marked as obsolete. Also, this control cannot be used as a container for other controls. During the migration all child controls are moved to the form's surface.
- **SSRibbon control:** the BackStyle, ButtonStyle, Outline, PictureDisabledFrames, PictureDn, PictureDnFrames, PictureUp, PictureDnChange, Shape, and ShapeSize properties are not implemented and are marked as obsolete.
- **SSScroll control:** the ScrollStyle property is not implemented and is marked as obsolete.
- **SSTransition control:** the ClipControls property is not implemented and is marked as obsolete. Also, the transitionType and duration arguments of the Transition methods are ignored.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=670
http://www.vbmigration.com/detknowledgebase.aspx?Id=647

ADO

All VB6 applications that have a reference to any version of the ActiveX Data Object library are migrated into a VB.NET application that has a reference to the ADODB Primary Interop Assembly (PIA). The Microsoft ADODB PIA doesn't include the definition of the **adDBFileTime** enumerated value: VB Migration Partner solves this issue by converting references to this value into references to **ADODB_DataTypeEnum_adDBFileTime** constant (this constant is defined in the control support library).

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=109

Array

A VB6 array is converted into a .NET array. This is a list of differences:

a) VB6 can declare arrays whose lower index is nonzero. The .NET array is always 0-based.
b) VBLibrary supports ReDim, ReDim Preserve, and Erase keywords to a scalar (non-array) Variant variable using **Redim6** or **RedimPreserve6** helper methods. These two methods work exactly like the VB6 keywords, except for one detail: if the array being dimensioned holds an array of UDTs (i.e. .NET structures), then the structure might not be initialized correctly.
c) The Erase statement works differently in VB6, depending on the array type (static or dynamic). The same behavior is supported by VBLibrary, thanks to **Erase6** and **ClearArray6** methods.
d) Under VB6, array assignments copy all the elements of the source array into the destination array; under VB.NET array assignments just copy the array pointer. VBLibrary exposes the **CloneArray6** that performs a *shallow* copy of the array by default, but can also perform a *deep* copy if True is passed to the second argument.
e) VB6 supports auto-instancing arrays, that is, array of objects declared with the As New. The same behavior is supported by VBLibrary, thanks to **CreateArray6** method.
f) The IsArray method differs from VB6 because it's unable to recognize uninitialized arrays. The **IsArray6** method replicates the same VB6 behavior.

The VB6 array type can be converted also as **VB6Array** object that supports arrays with nonzero lower index. Alternatively the VB6 array can be converted as **VB6ArrayNew** which adds support for auto-instancing arrays.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=32
http://www.vbmigration.com/detknowledgebase.aspx?Id=87
http://www.vbmigration.com/detknowledgebase.aspx?Id=616
http://www.vbmigration.com/detknowledgebase.aspx?Id=617
http://www.vbmigration.com/detknowledgebase.aspx?Id=53
http://www.vbmigration.com/detknowledgebase.aspx?Id=362
http://www.vbmigration.com/detknowledgebase.aspx?Id=629

Byte-oriented string functions

VB.NET doesn't support byte-oriented string functions, such as LeftB, RightB, and MidB. VBLibrary keeps this same behavior using special **LeftB6**, **RightB6**, and **MidB6** methods. These replacement methods mimic their VB6 counterparts, but they may fail to reproduce perfectly the original VB6 behavior.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=599

Cls

In some cases the Cls method fails to correctly refresh other child controls on the form. If you find that a Cls method on the form affects other controls, you should refresh those manually after the Cls method, or use the **RefreshChildControls** method exposed by the VB6Form class.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=653

Collection objects

VB6 collections are converted as .NET collections or, alternatively, they could be converted as **VB6CollectionVariant** objects, which use items of the **VB6Variant** type. The main difference between VB6 and .NET collection is that the latter can't be modified while it is inside a For Each loop.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=89
http://www.vbmigration.com/detknowledgebase.aspx?Id=628
http://www.vbmigration.com/detknowledgebase.aspx?Id=15

Colors

VBLibrary exposes two methods (**FromOleColor6** and **ToOleColor6**) that convert a 32-bit integer into a .NET Color value and vice versa.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=18

Command control

The VB6 command control is fully supported. The only exception is related to the order of some events.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=376

CommonDialog control

This control is fully supported, except for the *cdlOFNShareAware* bit of the Flags property.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=88

Control arrays

All control array features are supported, including dynamic loading and events. Support is provided by means the **VB6ControlArray(Of T)** type. The only differences are:

a) This class only supports controls of the same type. In some cases ListBox or Frame controls are translated as different controls (depending on Style or BorderStyle properties). For this reason, the ListBox or Frame control arrays have the same Style or BorderStyle property.
b) For the same reason, an array of menu controls can't contain separators.
c) In VB6 is legal to reference a control array element before actually creating it. In .NET this is not true.
d) VB6 allows you to dynamically create a submenu or a context menu immediately before displaying it, and to destroy it immediately afterwards. VB.NET requires that the menu elements exist at the instant when the Click event is processed.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=601
http://www.vbmigration.com/detknowledgebase.aspx?Id=626
http://www.vbmigration.com/detknowledgebase.aspx?Id=26
http://www.vbmigration.com/detknowledgebase.aspx?Id=36

Controls collection

The VB6 Controls collection is supported. The only difference is that the Add method doesn't work when adding a UserControl that has a Friend scope or isn't decorated with an VB6Object attribute.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=311
http://www.vbmigration.com/detknowledgebase.aspx?Id=605

DAO-RDO controls

DAO and RDO data controls are supported with these limitations:

a) The DAO Data control exposes the Recordset property, which returns a reference to the inner DAO.Recordset object. Similarly, the MSRDC control exposes the Resultset property, which returns a reference to the inner RDO.rdoResultset object. If the current position is changed using these objects, no event is raised and all bound controls aren't updated.
b) In some cases the Validate event may not fire.

The other differences in RDO data control are:

c) When the Refresh method is invoked, the VB6 RemoteData control fires the QueryCompleted event, whereas the .NET control fires the QueryCompleted and then the Reposition event.
d) When you reach the BOF or EOF condition, the VB.NET control disables the Previous or Next button, respectively, whereas the original VB6 control never disables any button.
e) When the parent form is loaded, the .NET RemoteData control fires the QueryCompleted and the Reposition events, whereas no event is fired by the original VB6 control.
f) The .NET control can raise the MouseDown, MouseMove, and MouseUp events, whereas the original VB6 control never fires these events.
g) When you change the DataSourceName property via code and invoke the Refresh method, the VB6 control raises the QueryCompleted event, whereas the .NET control fires the QueryCompleted and the Reposition event.
h) If the BOFAction property is set to 0, the first record is the current record, and the end user clicks on the Previous button, then the .NET control fires the Validate and Reposition event. No event is fired by the original VB6 control in the same circumstances.

i) If the EOFAction property is set to 0, the last record is the current record, and the end user clicks on the Next button, then the .NET control fires a Validate and a Reposition event. Under the same circumstances, the VB6 control fires the following sequence of events: Validate, Reposition, Validate, Reposition. However, the VB6 control fires this sequence only the very first time the button is clicked since the form is loaded; all subsequent clicks don't fire any event. (This is probably a VB6 quirk.)
j) If you modify the contents of a TextBox (or another similar control) bound to a VB6 RemoteData control and the move to a different record, the following events are fired: Validate (with Action=13), Validate (with Action=3), and Reposition. In the same circumstances the .NET control fires only two events: Validate (with Action=3) and Reposition.
k) In case of composite errors, the VB6 RemoteData control can fire multiple Error events. Conversely, the VB.NET control always fires a single Error event.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=328
http://www.vbmigration.com/detknowledgebase.aspx?Id=97
http://www.vbmigration.com/detknowledgebase.aspx?Id=102
http://www.vbmigration.com/detknowledgebase.aspx?Id=646

Data binding

The control library supports binding with the Data, RDO Data, and ADODC controls, and perfectly reproduces the VB6 behavior. In some cases the data source of a bound control isn't updated if the control's value is modified via code. To work around, the **VB6Form** and **VB6UserControl** classes expose a custom **UpdateDataSource** method.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=612

DataGrid

The .NET control wraps the original DataGrid control, but in some cases the behavior is different:

a) Changing properties of the data source associated with a DataGrid control can throw an exception.
b) The DataGrid's SelColChange event may fire unexpectedly.
c) Assigning a reference to an ADO Data control to the DataSource property causes the control to recreate the Columns collection so that their captions match the fields' names exactly.
d) Some properties of VB6DataGrid control can't be modified from inside Visual Studio's property window.
e) In VB6, assignments to the DefColWidth property are ignored if the VB6 developer had used the "Retrieve Fields" command to populate the grid's column. In migrated VB.NET applications, however, assigning a value to this property resizes all the columns to the specified width.
f) In a VB6 application you typically need to invoke the DataGrid.ReBind method after adding a new column to the grid, in order to fill the new column with values from the database. In VB.NET applications the ReBind method has the effect of restoring the original set of columns, therefore in practice you can't add a column at runtime.
g) Modifying the RowHeight property at runtime has no effect if the parent form has its ScaleMode property set to 0-User.
h) Under VB.NET, the DataGrid control fires a few spurious GotFocus and LostFocus events. For example, when the user clicks on another cell on the same grid, the following events are fired under VB6: MouseDown, MouseUp, Click, RowColChange, RowColChange. The same action fires the following events under VB.NET: LostFocus, GotFocus, MouseDown, GotFocus, MouseUp, Click, RowColChange, RowColChange, LostFocus.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=334
http://www.vbmigration.com/detknowledgebase.aspx?Id=110
http://www.vbmigration.com/detknowledgebase.aspx?Id=111
http://www.vbmigration.com/detknowledgebase.aspx?Id=113
http://www.vbmigration.com/detknowledgebase.aspx?Id=114
http://www.vbmigration.com/detknowledgebase.aspx?Id=116

DataCombo control

The DataCombo control is supported with these differences:

a) The Locked property can't be directly implemented in VB.NET and its behavior can only be approximated. When the Locked property is True then the combobox's style is set equal to DropDownList, so that the end user can only select one of the values in the list area.
b) The DataBindings property isn't supported.
c) The VB.NET control never fires the DblClick event.
d) When you click the VB6 control, the following events are fired: MouseDown, MouseUp, Click. The VB.NET control raises the Click event immediately after MouseDown and the sequence is MouseDown, Click, MouseUp.
e) When the end user moves the input focus to the VB6 control by clicking on it, the following events are raised: MouseDown, Click, MouseUp, GotFocus. The VB.NET raises the GotFocus before the MouseDown event and the actual sequence is: GotFocus, MouseDown, Click, MouseUp.
f) If a VB6 DataCombo control is the first control that gets the input focus when the form loads, it fires the following event sequence: Change, Change, GotFocus (two Change events followed by GotFocus); in the same circumstances the VB.NET control fires the following events: Change, LostFocus, GotFocus.
g) If the end user opens the dropdown list and selects a different item, the VB6 control raises the following events: MouseDown, GotFocus, MouseUp, Click, MouseDown, Change, MouseUp, Click. The VB.NET control raises fewer events: LostFocus, GotFocus, MouseDown, Click, MouseUp, Change.
h) If the end user double-clicks an element of the VB6 DataList control, the following event sequence takes place: MouseDown, MouseUp, Click, DblClick, MouseUp. Under VB.NET the sequence is slightly different: MouseDown, Click, MouseUp, MouseDown, DblClick, MouseUp.
i) If you add, remove, or change a record in the data source that is associated to the list area of the DataCombo control, you need to manually invoke the control's Refresh method.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=322
http://www.vbmigration.com/detknowledgebase.aspx?Id=611

DataList control

The DataList control is supported with these differences:

a) When the end user moves the input focus to the VB6 control by clicking on it, the following events are raised: MouseDown, Click, MouseUp, GotFocus. The VB.NET control raises the GotFocus before the MouseDown event and the actual sequence is: GotFocus, MouseDown, Click, MouseUp.
b) The DataBindings property isn't supported.

c) When the end user double-clicks on an item of the list, the VB6 DataList control fires the following events: MouseDown, MouseUp, Click, DblClick, MouseUp. The VB.NET control raises a slightly different sequence: MouseDown, Click, MouseUp, MouseDown, DblClick, MouseUp.
d) If you add, remove, or change a record in the data source that is associated to the list area of the DataList control, you need to manually invoke the control's Refresh method.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=321
http://www.vbmigration.com/detknowledgebase.aspx?Id=611

DDE

DDE support is implemented by *simulating* the VB6 behavior, but without actually using any native DDE feature offered by Windows. This detail has an important consequence: DDE communications only work between VB.NET applications that have been converted by VB Migration Partner and that use VB Migration Partner's support library. If your original VB6 code uses DDE to communicate with Microsoft Excel or any other compiled DDE server application, it won't be possible to establish the communication.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=613

Default members

VB6Library offers two methods (**GetDefaultMember6** and **SetDefaultMember6**) that discover and resolve at runtime the default member reference of an object and work correctly also if the default member takes one or more arguments. These methods don't work correctly if the argument is a COM object and intentionally throw an exception to ensure that the developer doesn't overlook the problem.
VB6Library also contains **IIf6** and **Choose6** functions that use internally the GetDefaultMember6 method and are able to evaluate the default member of objects passed to them.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=17
http://www.vbmigration.com/detknowledgebase.aspx?Id=600

Drag-and-drop

"Classic" and OLE drag-and-drop are supported by VBLibrary, with these differences:
a) An automatic drag-and-drop operation can be initiated by pressing the either the left or the right mouse button and then dragging (VB6 only supports the left button). Supporting the right mouse button is necessary for controls that can contain editable text (i.e. the TextBox control), because dragging with the left button would change the text selection.
b) The drag operation begins only when the mouse cursor is moved outside the source control's border. Only after this point all events that are related to drag-and-drop (i.e. OLEStartDrag) will fire.
c) In some rare cases the "classic" drag-and-drop might not work properly if a previous drag-and-drop operation has been concluded in an "irregular" way. **VB6Utils.StopDragDrop** method can be used to manually signal that a drag-and-drop operation has been concluded.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=609
http://www.vbmigration.com/detknowledgebase.aspx?Id=661
http://www.vbmigration.com/detknowledgebase.aspx?Id=662

DTPicker

The .NET DTPicker control is supported in .NET with only these differences:

a) The VB.NET DTPicker control doesn't support custom fields, therefore it doesn't support the *Format*, *FormatSize*, and *CallbackKeyDown* events. These events are marked as obsolete.
b) The DateChanged property is set to True when a new value is assigned to the Value property, as in VB6. However, in the converted VB.NET application the Value property is assigned behind the scenes more often than in VB6, therefore you might find that the DataChanged property becomes True under certain circumstances in VB.NET but not in the original VB6 application.
c) The VB6 DTPicker control accepts dates as early as 1/1/1601 and as late as 12/31/9999, and in fact these are the default values for the MinDate and MaxDate properties, respectively. Conversely, the .NET DateTimePicker control doesn't accept dates earlier than 1/1/1753 or later than 12/31/9998.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=315
http://www.vbmigration.com/detknowledgebase.aspx?Id=82
http://www.vbmigration.com/detknowledgebase.aspx?Id=83

Enabled

The Enabled property is fully supported for all controls. The only difference with VB6 is that disabled controls have a light gray background color and a dark gray foreground color. You can force controls on VB.NET forms to behave like their VB6 counterparts by calling the **ConvertSystemColors6** method on a single control or on its container control.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=45

End keyword

The VB6 End keyword ends the application with no other event raised. The VB.NET End keyword still causes the Form's Terminate event, therefore the equivalence with the original VB6 code isn't perfect.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=359

FileOpen, FileClose, FilePut, FileGet

FileOpen6, **FileClose6**, **FileGet6**, **FilePut6**, and all other file-oriented method read and write values and UDTs using the same format that VB6 uses. This is the list of features that VB Migration Partner doesn't handle correctly:

a) Variant elements in UDTs aren't handled correctly and prevent the entire UDT from being written to and read from file.
b) Dynamic arrays in UDTs aren't handled correctly and prevent the entire UDT from being written to and read from file.
c) FileGet6 can't read arrays that haven't been initialized in code or arrays that contain a number of elements that is different from the length stored in the file.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=368
http://www.vbmigration.com/detknowledgebase.aspx?Id=55

FlatScrollBar control

When the user moves indicator of a VB6's FlatScrollBar control, then the control fires several Scroll events, followed by a single Change event. In the same situation, the **VB6FlatScrollbar** control fires just one Scroll event, followed by a single Change event.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=91

Font

VB6's Font and StdFont objects are converted to .NET Font objects. The main difference is that the .NET Font object is immutable. VBLibrary includes the **FontChangeName6, FontChangeSize6, FontChangeBold6, FontChangeItalic6, FontChangeStrikeout6**, and **FontChangeUnderline6** methods which allow you to work around the read-only nature of the corresponding property of the .NET Font object.
Under VB6 you can change the Font property of a form, a PictureBox, or another container control without affecting the font of child controls. In migrated VB.NET this isn't true: any change in the Font property of a container affects the Font property of all child controls, unless you've specified a specific value for one or more font-related properties (FontName, FontSize, FontBold, and so forth).

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=35
http://www.vbmigration.com/detknowledgebase.aspx?Id=651

Form

The Form control is supported with these differences:

a) When the default instance is used, the .NET form works correctly the first time it is being showed, but behaves differently (or delivers wrong results) if it is re-opened.
b) In some rare cases, when closing or unloading a form from inside an event handler in the migrated VB.NET project (for example, the Click event of a button) you get an ObjectDisposedException error.
c) The.NET form ignores assignments to Left and Top properties also if StartupUpPosition is equal to 3-Windows Default.
d) In some cases a VB6 form has a border but the migrated .NET form hasn't.
e) In VB6, when you access a control located in a different form, a Load event raises in target form. In VB.NET the Load event raises only when you call the Show method.
f) In the VB6 Form_Initialize you can access to a form's property or one of the controls on the form, in .NET you can't.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=27
http://www.vbmigration.com/detknowledgebase.aspx?Id=637
http://www.vbmigration.com/detknowledgebase.aspx?Id=31
http://www.vbmigration.com/detknowledgebase.aspx?Id=38
http://www.vbmigration.com/detknowledgebase.aspx?Id=69
http://www.vbmigration.com/detknowledgebase.aspx?Id=323
http://www.vbmigration.com/detknowledgebase.aspx?Id=614

Forms collection

VBLibrary supports the form collection; it contains instances of the VB6Form and VB6MdiForm classes, and returns standard .NET forms created in Visual Studio only if the **Forms6.IncludeNetForms** property is set to True.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=22

GotFocus/LostFocus/Validate/MouseDown events

All these events are supported for all controls, with the following differences:

a) Setting the Visible=False for a VB6Frame may cause spurious GotFocus, LostFocus, and Validate events
b) The VB6 UserControl class receives a GotFocus or LostFocus event only if the user control contains no child controls that can take the input focus
c) A VB6UserControl fires GotFocus and LostFocus events only if it receives the focus only by means of the Tab key or a mouse click directly on the user control's surface.
d) The .NET MsgBox, InputBox, and common dialogs cause spurious LostFocus and GotFocus events
e) VB6 controls fire the Validate event first and then the LostFocus event. Conversely, migrated .NET controls fire these events in the same sequence only if end users move the input focus by means of the keyword; if they use the mouse, the control fires a LostFocus event, then the Validating event. You can force the controls in the support library to behave more closely to the original VB6 controls by setting the **VB6Config.FocusEventSupport** static property to True.
f) In such circumstances, VB6 controls fire the MouseDown event and then the GotFocus event, whereas .NET controls do exactly the opposite: they fire the GotFocus event and then the MouseDown event. You can force the support library to fire the MouseDown and GotFocus events in the same order as VB6, by setting the **VB6Config.FocusEventSupport** static property to True.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=335
http://www.vbmigration.com/detknowledgebase.aspx?Id=664
http://www.vbmigration.com/detknowledgebase.aspx?Id=331
http://www.vbmigration.com/detknowledgebase.aspx?Id=332
http://www.vbmigration.com/detknowledgebase.aspx?Id=675

Graphic methods (Line, Circle, PSet, Cls, PaintPicture, PrintForm)

All these methods are supported with the following differences:

a) The DrawMode property isn't supported when box filled parameter is set.
b) If graphic shapes are generated in Activate event and the AutoRedraw property is set to False, the the graphic output is cleared immediately after the form becomes visible
c) The library doesn't support mixing VB6-like graphic methods and GDI32-based methods (i.e. graphics created by means of direct Windows API calls) in the same control, because the two methods can interfere with each other.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=28

hDC

VBLibrary supports hDC property of forms, user controls, and regular controls. However, when a .NET program acquires the handle of a GDI device context, it becomes "locked" until you explicitly release it. If the application attempts to display any user interface or graphic element on the control, the .NET runtime throws an InvalidOperationException error whose message reads "Object is currently in use elsewhere". **ReleaseHdc6** method can be used to invoke the ReleaseHdc method on each control passed as argument.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=40

Image control

Image control is fully supported with the following difference: in VB6, an Image control whose Picture property is Nothing has a transparent background, i.e. you can see "through" it but the Image control still receives mouse events. VBLibrary supports this scenario by setting the Image control's Visible property to False and manually forwarding all mouse events to the Image control but it works only if the BorderStyle property is set to 0-None.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=120
http://www.vbmigration.com/detknowledgebase.aspx?Id=61

ImageCombo control

ImageCombo control is supported with these differences:

a) When the Locked property is set to True, the VB6 control allows the end user to select all or a portion of the string showed in the edit area, which can be copied to the clipboard. Conversely, when the Locked property is True, the VB.NET control behaves like a combobox whose style is set to DropDownList. This detail prevents the end user from selecting all or part of the text that appears in the edit area. Consequently, when Locked is True the SetText property always returns "" (empty string) and the SelStart and SelLength properties always return 0 (zero).
b) The VB6 ImageCombo control always display the image associated with the currently selected item when the list area is closed, regardless of the value of the Locked property. When the Locked property is False – in other words, when the end user can edit the text of the currently selected item – the VB6ImageCombo control used in converted VB.NET applications doesn't display the image associated to the currently selected control when the list area is closed.
c) When the Locked property is True and the ImageCombo control is bound to a data source, the VB6 control can still display any value from the database. Conversely, the VB.NET control displays a value from the database only if the value matches one of the items in its list area.
d) In VB6, the Change event may fire multiple times when you add a new item to the control. We consider this to be a bug and haven't replicated this behavior in the VB.NET, which fires the Change event just once.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=314

ImageList control

ImageList control is supported with these differences:

a) Under VB6, the Width and Height properties of the bitmap returned by the ListImage.Picture property are returned in HiMetric units, whereas under VB.NET these values are returned as pixels.

b) Under VB6, the Width and Height properties of the bitmap returned by the ListImage.Picture property reflect the original size of the bitmap; under VB.NET these properties reflect the original size of the bitmap only if the bitmap was added dynamically at runtime; if the bitmap was added at design time, these properties return the dimensions of the resized bitmap.
c) The VB6 ImageList control can contain images of different size (unless the ImageList is bound to a common control) and there is no limit to the image size; the .NET control only accepts images of same size and can't contain images wider or higher than 256 pixels.
d) The ListImage.Draw method of the VB.NET control doesn't support the imlSelected and imlFocus styles.
e) In an undocumented VB6 behavior, the ListImage.ExtractIcon method can affect the size of the bitmap returned by the ListImage.Picture property. This behavior hasn't been implemented in VB.NET.
f) If you modify the MaskColor property of the VB6 ImageList control the new setting affects all the images that were previously loaded inside the control. By contrast, if you assign the TransparentColor property of the .NET ImageList control the new setting affects only the images that are loaded *after* the assignment.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=610
http://www.vbmigration.com/detknowledgebase.aspx?Id=73

Label control

Label control is fully supported with the following differences:
a) Setting the BackColor property of a VB6 Label to Transparent control makes the label's background truly transparent, you can actually see the form's background "through" the label, even if clicks on the background are routed to the Label control. Conversely, setting the BackColor property of a VB.NET Label to Transparent is equivalent to setting it equal to the container's BackColor property, and no transparency effect is achieved.
b) Mouse event aren't received by transparent Label controls

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=120
http://www.vbmigration.com/detknowledgebase.aspx?Id=64

LargeChange property

All the scrollbar controls in the VBLibrary (**VB6HScrollBar**, **VB6VScrollBar**, **VB6WLHScroll**, **VB6WLVScroll**, and **VB6FlatScrollBar)** ignore any assignment to the LargeChange property. More precisely, the property correctly retains the value assigned to it, but the new value isn't assigned to the underlying .NET control. You can restore the standard .NET behavior by assigning False to the **IgnoreLargeChange** property that all VB6*** scrollbar controls expose.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=81

Len and Trim methods

Len6 works both with strings and User-Defined Types (UDTs), with these differences:
a) If the UDT contains fixed strings, the returned value is not the same as in VB6.
b) It only works with strings and aren't capable to deal with DBNull values arriving from the database (this is true also for Trim method).

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=649
http://www.vbmigration.com/detknowledgebase.aspx?Id=641
http://www.vbmigration.com/detknowledgebase.aspx?Id=635

Line and shape controls

Line and shape controls are supported with these differences:

a) Forms that contain dozens of such controls (for example, forms that use Shape and Line controls to create simple animations) are subject to serious flickering issues.
b) Under VB6, Shape and Line controls are updated immediately after you change one of their properties. In .NET applications, however, changing a property of a Shape or Line control simply invalidates its container (in order to reduce flickering). To reproduce the same VB6 behavior you can set the control's ImmediateUpdate property to True.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=46
http://www.vbmigration.com/detknowledgebase.aspx?Id=47

Listbox control

The Listbox control is supported with these differences:

a) Under .NET, removing an item from a multi-selectable ListBox resets ListIndex property.
b) Under .NET, removing an item from a single-selectable ListBox selects the previous item.
c) If the Style property is set to True, under VB6 the user can check or uncheck an item by clicking on the checkbox icon to the left of each item; this single click automatically selects the ListBox item, if it wasn't the currently selected item. Under .NET, two distinct clicks are necessary to check or uncheck an item that isn't the selected item: the first click is necessary to select the item, and the second click (right on the checkbox) is needed to check or uncheck it.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=80
http://www.vbmigration.com/detknowledgebase.aspx?Id=668
http://www.vbmigration.com/detknowledgebase.aspx?Id=660

ListView control

The ListView control is supported with these differences:

a) Values TopRight, BottomLeft, BottomRight, Center for PictureAlignment don't work in .NET
b) Assignments to the ListView.SelectedItem property are ignored if the control isn't visible
c) the VB6ListView control ignores icon indexes specified in the ColumnHeaders.Add method or assigned to the ColumnHeader.Icon property
d) The FlatScrollBar property isn't supported and is marked as obsolete.
e) The TextBackground property isn't supported and is marked as obsolete.
f) The Ghosted property of the ListItem object isn't supported and is marked as obsolete.
g) The ToolTipText property of the ListSubItem object isn't supported and is marked as obsolete.
h) The ReportIcon property of the ListSubItem object isn't supported and is marked as obsolete.
i) When the View property is set to LargeIcons or SmallIcons, the VB6 control allows you to arrange items with the mouse; conversely, the .NET ListView never allows you to move items.

j) When you click on the blank area in the .NET ListView control, the currently highlighted item is unselected; when you perform the same operation on the VB6 ListView control, the currently selected item stays selected. (In VB6 you can unselect the currently selected item via code, though).
k) The ListView's ToolTipText property is converted correctly, however by default the ShowItemToolTips property of the .NET control is set to True, which prevents the control's tooltip to appear in favor of the tooltip of individual items. You can force the display of the control's tooltip by setting the ShowItemToolTips property to False.
l) If the input focus is currently on the .NET ListView control and you then activate another application and finally go back to the .NET application, the control raises several LostFocus and GotFocus events. The VB6 ListView control doesn't raise all these extra events.
m) The VB6 ListView control raises a Click event also if you click on the control's blank area, whereas the .NET ListView control raises the Click event only if you click on an item.
n) When you click on an item, the VB6 ListView control raises the following sequence of events: MouseDown, ItemClick, MouseUp, Click. In the .NET control the event sequence is: MouseDown, Click, ItemClick, MouseUp.
o) When you edit an item, the .NET ListView control raises the following events: BeforeLabelEdit, LostFocus, AfterLabelEdit, GotFocus. The VB6 ListView control raises only the BeforeLabelEdit and AfterLabelEdit events.
p) When you move the input focus to a VB6 ListView control using the mouse, the event sequence is: MouseDown, GotFocus, MouseUp. The same operation causes the .NET control to raise the events in this order: GotFocus, MouseDown, MouseUp.
q) When you click on the checkbox associated with an item, the VB6 ListView control raises the following events: MouseDown, ItemCheck, MouseUp, Click. In the same circumstances the .NET control fires these events: MouseDown, Click, ItemClick, MouseUp, ItemCheck.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=307
http://www.vbmigration.com/detknowledgebase.aspx?Id=325
http://www.vbmigration.com/detknowledgebase.aspx?Id=74

MDIForm and MDI child forms

MDIForm and MDI child forms are supported with these differences:
a) Invoking the Show method of MDI child forms containing one or more ActiveX controls might not fire the Load event.
b) The Scrollbars property is only partially implemented in MDI forms, because setting this property to False doesn't ensure that the scrollbars are really hidden.
c) Under VB6 it is legal to invoke the Show method of an MDI child form from inside the Load event handler of its MDI parent form, that is when the MDI parent form isn't visible yet, or from inside the Load event handler of a splash screen. Under .NET, invoking the Show method of an MDI child form correctly displays its MDI parent form only if the MDI child form isn't the first form that the application displays.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=673
http://www.vbmigration.com/detknowledgebase.aspx?Id=627
http://www.vbmigration.com/detknowledgebase.aspx?Id=34

Menu

Standard and popup menus are fully supported, including shortcut keys and control arrays of menu items. The VB6MdiForm class exposes a Boolean property named **HideMainMenuOnChildFormActivate**. If this property is False (the default value), menus are merged as they were in previous versions. If this property is set to True, however, the menus on the MDI child form replace the toplevel menus in the MDI parent form, as it happens in VB6.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=650
http://www.vbmigration.com/detknowledgebase.aspx?Id=655

MonthView

The MonthView control is supported with these differences:

a) The BackColor property has no effect and has been made obsolete, even though it correctly retains the value that you've assigned to it via code.
b) The Appearance and BorderStyle properties have no effect and have been marked as obsolete.
c) The DateChanged property is set to True when a new value is assigned to the Value property, as in VB6. However, in the converted VB.NET application the Value property is assigned behind the scenes more often than in VB6, therefore you might find that the DataChanged property becomes True under certain circumstances in VB.NET but not in the original VB6 application.
d) When you click on the year number portion of the VB.NET MonthView control, no mouse event is fired.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=316

Null and Empty values

VB6 Null and Empty values are converted to the special **Empty6** and **Null6** values, respectively, and are correctly recognized by functions such as **IsEmpty6** and **IsNull6**.
In some cases there are some exceptions when trying to manage strings and null values together. VBLibrary exposes the **FixNullValue6** method, which converts Null and Empty values to the empty string and can therefore be used to solve this issue. VBLibrary defines also the **VB6Config.ReturnedNullValue** value, which is the value that some string functions (Chr6, CurDir6, Environ6…) return when they receive a null argument.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=19
http://www.vbmigration.com/detknowledgebase.aspx?Id=374
http://www.vbmigration.com/detknowledgebase.aspx?Id=339

OptionButton control

The VB6 OptionButton control is fully supported. It has only one difference: VB.NET control automatically sets the Checked property of the first control in a group of Option buttons, while VB6 allows you to define a group of Option controls and leave the Value property of all of them set to False.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=60

Paint event

VB6 and VB.NET forms greatly differ on how form refreshes are handled. **VB6Form**, **VB6PictureBox** or **VB6UserControl** controls could receive spurious or missing Paint events (i.e. when setting the Font property). We have added the **IgnoreNextPaintEvent** Boolean property to these controls which, if True, forces VB Migration Partner to ignore the next Paint event for a given form, PictureBox or UserControl.
If the form contains many child controls that must fire the Paint event, invoking the Refresh on each control can be an annoying, error-prone approach. For this reason, the VB6Form class exposes the **RefreshChildControls** method, which invokes the Refresh method on each child control.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=666
http://www.vbmigration.com/detknowledgebase.aspx?Id=667
http://www.vbmigration.com/detknowledgebase.aspx?Id=652
http://www.vbmigration.com/detknowledgebase.aspx?Id=49

RichTextBox control

The RichTextBox control is supported with these differences:

a) Invalid assignments to the Rtf property might not throw an exception.
b) The font of the text assigned to the Rtf property at design time might be different in the converted VB.NET program. This change is necessary because .NET controls don't accept System fonts such as MS Sans Serif or Courier.
c) Under VB6, the SelFontName, SelFontSize, SelBold, SelItalic, SelUnderline, and SelStrikeThru properties of the RichTextBox control return Null if the selection contains characters with different attributes. In these cases the VB6RichTextBox returns either False or Nothing.
d) The VB6 control stores newline characters as CR-LF pairs (ASCII 13 + ASCII 10), whereas the VB.NET control stores them as individual LF characters (ASCII 10).
e) The VB6 control ignores invalid font assignments, unlike the VB.NET control.
f) Reading back a value assigned to the SelHanging, SelIndent, and SelTabs properties might return a value different from the value assigned previously. The reason for this behavior is that these properties take or return values in twips (more precisely, in the container's ScaleMode) whereas the .NET control internally stores these values in pixels. When you assign a value to one of these properties the value is internally converted to pixel and then converted back to twips when the property is read back. Pixels values are stored internally as integers, therefore the double conversion might bring to a loss of precision.
g) In the VB6 control, the SelTabCount property is independent from the current selection. More precisely, this property returns Null if the selection spans paragraphs with different tab settings. However, if you then assign any value to the SetTabs(n) property, the SelTabCount property returns the highest of the SelTabCount values for all the paragraphs in the selection, and you can query the SelTabs(n) property for each value of N included between 0 and SelTabCount-1, even if you never actually assigned the N-th tab for a given paragraph.
Under VB.NET, the SelTabCount property behaves differently. If the current selection includes includes two or more paragraphs with different tab settings, this property returns 0 (zero). If you then assign a value to the SetTabs(n) property an exception occurs.
h) When the end user selects a piece of text with the mouse, when he or she releases the mouse button the VB6 control fires the following events: MouseUp, Click, SelChange. In the same circumstances, the VB.NET control fires the same events but in a different order: Click, SelChange, MouseUp.
i) If the OLEDragMode property is set to 2-Automatic, the VB6 RichTextBox control has an inconsistent behavior when a file is dropped on it, for example at the end of a drag-and-drop operation initiated from Windows Explorer: depending on the nature of the file, the RichTextBox

control either displays the file contents or the file icon and name. The VB.NET version of the RichTextBox always displays the file contents.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=319
http://www.vbmigration.com/detknowledgebase.aspx?Id=85
http://www.vbmigration.com/detknowledgebase.aspx?Id=86

ScaleHeight and ScaleWidth

VBLibrary supports the ScaleHeight and ScaleWidth properties of form objects. However, the design-time value of these properties is lost as soon as the form is modified in any way inside Visual Studio's designer. Another difference is related to negative values for ScaleWidth and ScaleHeight properties, that cause incorrect output in Microsoft Vista.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=674
http://www.vbmigration.com/detknowledgebase.aspx?Id=29

ScaleMode

VBLibrary supports all ScaleMode values, for forms, PictureBox and UserControl containers. However, if the application relies on user-defined coordinate systems (i.e. ScaleMode is set to the value 0-vbUser), then you might notice that the Left, Top, Width, and Height of child controls might not work exactly as they do under VB6.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=71

SSTab control

The SSTab control is supported with these differences:

a) In VB6, the Top property of controls hosted inside an SSTab control is relative to the top border. In .NET it is relative to TabPage border
b) All the tabs in the VB.NET control are visible and enabled. VB Migration Partner offers support for the TabVisible and TabEnabled properties by relying on the Visible and Enabled properties of the TabPage .NET control. However, both properties are deprecated aren't officially supported by Microsoft. Please ensure that you test your VB.NET under all possible circumstances, different operating system versions, and so on, before deploying the migrated application to your customers.
c) The BackColor and ForeColor properties aren't supported and are marked as obsolete.
d) The VB.NET control can't display pictures, therefore the Picture and TabPicture properties aren't supported and are marked as obsolete.
e) The VB.NET control can't wrap long captions in tabs, therefore the WordWrap property isn't supported and always returns False.
f) The VB.NET control can display tabs on multiple rows but you can't decide how many tabs must be displayed in each row, therefore the TabsPerRow property isn't supported and is marked as obsolete.
g) The Style property isn't directly supported and is marked as obsolete.
h) The VB.NET control always displays focus rectangles, therefore the ShowFocusRect property isn't supported and is marked as obsolete.

i) In VB6 you can move the focus to a control located on a tab that isn't currently visible by pressing the Alt+key combination corresponding to the hotkey associated with that control. This operation isn't directly supported by the .NET control, but you can enable it using ProcessHotKey method.
j) When you click on the control to give it the input focus, VB6 raises the following events: GotFocus, MouseDown, MouseUp. In VB.NET the order is slightly different: MouseDown, GotFocus, MouseUp.
k) When you double-click on a tab, VB6 raises the following events: MouseDown, GotFocus, MouseUp, DblClick, MouseUp. In VB.NET the event sequence is different: GotFocus, MouseDown, MouseUp, MouseDown, DblClick, MouseUp.
l) Under VB6, if a control is on a tab that isn't currently visible, then its events are fired anyway and can be trapped by the client application. For example, setting the Text property of a TextBox contained in *any* tab of an SSTab fires the corresponding Changed event. Conversely, under .NET controls that are on tabs other than the currently active tab don't raise any event. This behavior is inherent to the .NET TabControl and can't be changed.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=318
http://www.vbmigration.com/detknowledgebase.aspx?Id=671
http://www.vbmigration.com/detknowledgebase.aspx?Id=624

StatusBar control

The StatusBar control is supported with these differences:

a) The VB.NET version of the StatusBar control prevents from adding or removing panels at runtime while the Style property is set to 1-sbrSimple. If you invoke the Add, Remove, or Clear methods on the StatusBar's Panels collection while Style is equal to 1-sbrSimple, an ArgumentException error is thrown
b) When you click on panel, the VB6 control raises the following events: MouseDown, PanelClick, MouseUp, Click. The VB.NET control raises the same events, but in a different order: MouseDown, Click, PanelClick, MouseUp.
c) When you double-click a panel, the VB6 cntrol raises the following events: MouseDown, PanelClick, MouseUp, Click, DblClick, PanelDblClick, MouseUp. The VB.NET control raises the following events: MouseDown, Click, PanelClick, MouseUp, MouseDown, DblClick, PanelDblClick, MouseUp. (Notice the order is different and that there is an extra MouseDown event.)
d) The Style property of the Panel object doesn't support the value 7-sbrKana.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=313

String type

The VB6 string is converted into a .NET string. The only difference is related to the values contained when the variable is not initialized yet: the uninitialized VB6 string contains an empty string, but the .NET one contains a *null string*, i.e. Nothing. For this reason, VB Migration Partner automatically initializes a String variable to an empty string in all cases when this initialization is possible.

The VB6 string type can be converted also as **StringBuilder6** object, which is useful if the original VB6 code contains many string concatenations, but in some cases it raises an InvalidCast exception.

Most methods in the VB6 runtime (GetAttr, SetAttr…) automatically truncate a string to its first ASCII 0 character; this behavior is very convenient when working with strings returned from Windows API calls, for

example. Unfortunately, the corresponding VB.NET methods don't truncate the string and therefore throw an exception. VBLibrary comes with a set of replacement file-related methods that replicates this behavior and automatically truncate their string arguments at the first ASCII 0 char. The provided methods are: **ChDir6**, **ChDrive6**, **FileCopy6**, **FileLen6**, **MkDir6**, **RmDir6**, **GetAttr6**, **SetAttr6**, **Kill6.**

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=118
http://www.vbmigration.com/detknowledgebase.aspx?Id=302
http://www.vbmigration.com/detknowledgebase.aspx?Id=345
http://www.vbmigration.com/detknowledgebase.aspx?Id=639

TabStop and TabIndex properties

TabStop and TabIndex work as VB6, with these differences:

a) VB6PictureBox, VB6Frame and other container controls don't correctly process the Tab key if TabStop=False
b) Setting the TabIndex property in VB.NET doesn't affect the TabIndex of other controls. To overcome this issue we have included the **SetTabIndex6** method that correctly shifts the TabIndex property of all the controls in the same container

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=589
http://www.vbmigration.com/detknowledgebase.aspx?Id=42

TabStrip

The TabStrip control is supported with these differences:

a) If Placement property is set to any value other than 0-tabPlacementTop, then the Style property can't be set to 2-tabFlatButtons.
b) The TabFixedWidth and TabFixedHeight properties might cause the VB.NET control not to look like the VB6 control on forms whose ScaleMode is set to a value other than 1-Twips.
c) The MultiSelect property isn't supported and always returns False.
d) The Separators property isn't supported and always returns True.
e) The TabMinWidth property isn't supported and always returns 0.
f) The TabStyle property isn't supported and always returns 0-tabTabStandard.
g) The Left and Top properties of individual TabPage elements can return values that don't match exactly those returned in the VB6 application.
h) The Highlighted property of individual TabPage elements isn't supported and always returns False.
i) If the user clicks on the tab that is currently selected, VB6 fires the MouseDown, MouseUp, and Click events. In the same circumstances, VB.NET fires only the MouseDown and MouseUp events.
j) If the user clicks on a tab other than the currently selected tab, the VB6 control fires the following events: MouseDown, BeforeClick, GotFocus, MouseUp, Click. In the same circumstances, the VB.NET controls fires the events in a different order: GotFocus, BeforeClick, Click, MouseDown, MouseUp.
k) If you select a tab other than the currently selected tab, via code, the VB6 control fires the BeforeClick and Click events; in the same circumstances, the VB.NET control fires the LostFocus, Validate, GotFocus, BeforeClick, and Click events.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=341

TextBox control

VB6 TextBox control is fully supported with the following difference: in VB6 sometimes data entry forms automatically select the contents of a TextBox control when the control gets the input focus, this behavior is perfectly migrated only when the end user moves the input focus to the target control by means of the keyboard.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=324

Timer controls

VB6 Timer control is fully supported with the following minor difference: Timers in interpreted VB6 programs don't fire events when a message box or an input box is visible on the screen, whereas events are never disabled in compiled VB6 programs, regardless of whether they are compiled to p-code or native code. VB.NET has no "interpreted mode", therefore in this converted VB.NET applications always behave like compiled VB6 applications and never disable timers.

Related KB article:

http://www.vbmigration.com/detknowledgebase.aspx?Id=65

Toolbar control

The Toolbar control is supported with these differences:

a) The .NET Toolbar control can't contain other controls and you can't move a control over a toolbar by setting its Container property.
b) The Caption property of all placeholder buttons is always set to "" (empty string.)
c) When you click a VB6 Button, the event sequence is: MouseDown, MouseUp, Click, DoubleClick. In the same circumstances, the .NET control fires these events: MouseDown, Click, ButtonClick, MouseUp.
d) When you double click an area of the toolbar where there are no buttons, the event sequence is: MouseDown, MouseUp, Click, DblClick, MouseUp. (Notice that there is one MouseDown but two MouseUp.) In the same circumstances, the .NET control fires these events: MouseDown, Click, MouseUp, MouseDown, DblClick, MouseUp.
e) VB6 ignores the attempt to add a ButtonMenu element to a Button object whose style is different from tbrDropDown (more precisely, the Add method returns a ButtonMenu object but this element isn't visible). In the same circumstances, the Add method throws an exception under VB.NET

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=326
http://www.vbmigration.com/detknowledgebase.aspx?Id=77
http://www.vbmigration.com/detknowledgebase.aspx?Id=327

TreeView control

The TreeView control is supported with these differences:

a) HitTest method of the TreeView control can return different values in VB.NET
b) The FullRowSelect property has effect in the .NET TreeView control only if the Style property has a value in the range 0-3 – that is, the values tvwTextOnly, tvwPictureText, tvwPlusMinusText, and tvwPlusMinusPictureText. In other words, if the current style includes tree lines, the .NET control never highlight the currently selected node.

c) The StartLabelEdit method of the .NET control works only if the LabelEdit property is 0-tvwAutomatic, whereas it always works in the VB6 control. (VB Migration Partner inserts a warning just before the call to StartLabelEdit.)
d) The DropHighlight property is not supported, but it is simulated
e) The CreateDragImage method of the Node object isn't supported and is marked as obsolete.
f) If your application invokes the StartLabelEdit on the current node from inside the TreeView's Click event handler, the control doesn't enter edit mode if the user clicks right on the TreeView node; however, if he or she clicks on the control's blank area, the StartLabelEdit method works correctly. This weird behavior is caused by the fact that, when the node is clicked, the control raises a LostFocus event immediately after the Click event, and the LostFocus event immediately ends the editing mode. You can work around this issue by invoking the StartLabelEdit from inside the MouseUp event handler.
g) When you move the focus from the TreeView control to another control, the VB6 TreeView control raises the Validate event and then the LostFocus event; the .NET TreeView control generates the LostFocus event first, and then the Validate event.
h) If the input focus is corrently on the .NET TreeView control and you then activate another application and finally go back to the .NET application, the control raises several LostFocus and GotFocus events. The VB6 TreeView control doesn't raise all these extra events.
i) When you move the input focus to a VB6 TreeView control using the mouse, the event sequence is: MouseDown, GotFocus, MouseUp. The same operation causes the .NET control to raise the events in this order: GotFocus, MouseDown, MouseUp.
j) The VB6 TreeView control raises a Click event also if you click on the control's blank area, whereas the .NET TreeView control raises the Click event only if you click on a node.
k) When you edit a node's label, the .NET TreeView control raises the following events: BeforeLabelEdit, LostFocus, AfterLabelEdit, GotFocus. The VB6 ListView control raises only the BeforeLabelEdit and AfterLabelEdit events.
l) When you click on the checkbox associated with a node, the VB6 TreeView control raises the following events: MouseDown, NodeCheck, MouseUp, Click. In the same circumstances the .NET control fires these events: MouseDown, NodeCheck, NodeClick, Click, MouseUp.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=312
http://www.vbmigration.com/detknowledgebase.aspx?Id=125

UpDown control

The UpDown control is supported with these differences:
a) Values assigned to the BuddyProperty at design-time aren't converted and by default the .NET UpDown control consider the default property of the buddy control as the buddy property.
b) The AutoBuddy property isn't supported and is marked as obsolete.
c) In VB6 if you set the BuddyControl property to a non-Nothing value, the UpDown control automatically moves near its buddy control. If you later reset the BuddyControl property to Nothing or an empty string, the UpDown control moves back to its original location. The .NET UpDown control mimics the former behavior (it moves near to its buddy control) but doesn't move back if the BuddyControl property is set to Nothing or empty string.
d) In VB6, if the end user clicks on the arrows, the following sequence of events occurs: MouseDown, Change, MouseUp, DownClick (or UpClick). In .NET, the last two events are reversed and the sequence is: MouseDown. Change, DownClick (or UpClick), and MouseUp.
e) While the VB6UpDown control behaves like the original VB6 control under normal circumstances, the equivalence isn't always ensured under error conditions. In other words, it isn't guaranteed that the VB6UpDown control raises an error with the same number as under VB6 and it isn't guaranteed that, when an error occurs, property values are reset as they are in VB6.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=317
http://www.vbmigration.com/detknowledgebase.aspx?Id=78

UserControl

The UserControl is supported with these differences:

a) InitProperties, ReadProperties and WriteProperties events are never raised. The code in these event handlers is correctly migrated, however it must be invoked manually in the migrated program. (For example, the InitProperties handler could be invoked manually from inside the constructor.)
b) The Default and Cancel properties of the VB6UserControl object don't behave like in VB6.
c) The VB6UserControl class contains WidthTwips and HeightTwips properties (in addition to Width and Height ones). These properties always return a value in twips when they are accessed from inside the UserControl class regardless of the parent form's ScaleMode or the current UserControl's Scalemode. VB Migration Partner correctly converts Width/Height in WidthTwips/HeightTwips when used inside a UserControl.
d) The VB6 Container property is an object property, and assigning it an object that can't work as a container causes an error 425-Invalid object use. This behavior isn't replicated in migrated apps.
e) The Ambient property is partially supported, and returns an instance of the VB6Ambient class that exposes all the properties of the VB6 Ambient object. 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.
f) AsyncRead and CancelAsyncRead methods and AsyncReadComplete and AsyncReadProgress events are supported, 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.
g) The EventsFrozen method is partially supported. 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.
h) ClipControls, ClipBehavior, DrawMode, FontTransparent, MaskPicture, MaskColor, PropertyPages, Hyperlink, Palette, PaletteMode, AccessKeys and HitBehavior properties are not supported.
i) AccessKeyPressed and HitTest events are not supported and are never raised.
j) CanPropertyChange and PropertyChanged methods are not supported and do nothing when executed.

Related KB articles:

http://www.vbmigration.com/detknowledgebase.aspx?Id=103
http://www.vbmigration.com/detknowledgebase.aspx?Id=381

Variant

The **VB6Variant** type mimics the behavior of the VB6 Variant type as closely as possible, for example by providing support for the special Null and Empty values. In VB6 is possible to assign a user-defined type (UDT) to a Variant variable and then access the UDT fields in late-bound mode. VB.NET supports this technique if the Variant is converted to an Object variable, but not if it is converted to the special VB6Variant type.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=598

VB6 global objects

VB6 global objects are supported, with the following limitations:

- **App:** most members are supported, included PrevInstance and methods related to event logging; the OleRequest and OleServer properties aren't supported and are marked as obsolete.
- **Clipboard:** all members are supported, except for GetData/SetData that have problems working with the DIB format
- **Screen:** all members are supported, except Fonts, FontCount, MousePointer, and MouseIcon are flagged as obsolete; assignments to MousePointer and MouseIcon throw an exception.
- **Printer** and **Printers:** all members are supported and behave exactly in VB6, except for **Orientation** property or offset of the output strings

Using these objects in serializable properties may crash Visual Studio.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=59
http://www.vbmigration.com/detknowledgebase.aspx?Id=62
http://www.vbmigration.com/detknowledgebase.aspx?Id=63
http://www.vbmigration.com/detknowledgebase.aspx?Id=634

Window-less controls

All window-less controls work as in VB6, except for the Group property of WLOption control.

Related KB article:
http://www.vbmigration.com/detknowledgebase.aspx?Id=366

ZOrder

VB6 Label and Image controls are *lightweight controls*, which means that they don't correspond to actual Windows controls. Instead, the corresponding .NET label or image are just "drawn" over the parent form's surface (or more in general, the surface of the container control). For this reason, a VB6 Label or Image control can never appear in front of other non-lightweight controls.
In order to send lightweight controls "behind" regular controls in all forms of the migrated application you have to set the VB6Form.ArrangeLightweightControls property to true.
Alternately, you can use the VB6Utils.ArrangeLightweightControls method to fix the lightweight controls' ZOrder for a specific form.

Related KB articles:
http://www.vbmigration.com/detknowledgebase.aspx?Id=658
http://www.vbmigration.com/detknowledgebase.aspx?Id=44