This will give you a better idea of how ImageKnife/VB works by explaining how some of the demo features were implemented.

What is ImageKnife/VB™?

ImageKnife/VB is a Visual Basic™ custom control (VBX) that you can add to any Visual Basic application. Its functionality at design time is similar to the Picture control that is shipped with VB. The combination of Visual Basic and ImageKnife/VB makes it extremely easy to create Microsoft® Windows™-based applications that include image handling.

The PicBuf

The basic element of ImageKnife/VB is the PicBuf, which is what we call the control that is created on-screen at design time. It is a rectangular control that can be placed on any form. Like most controls, you can create a control array of PicBufs at either design time or run time.

Properties, Functions, Methods, and Events

The behavior of a PicBuf and its contents are controlled through the use of properties, methods, and function calls as triggered either in code or by Windows events.

Here is a synopsis of the properties, functions, methods, and events:

Standard Visual Basic Properties (identical to other VB controls except as noted)

AutoSize

BackColor

BorderStyle
  None or fixed single

DragIcon

DragMode

Enabled

Height

Index

Left

MousePointer

LinkItem

LinkMode

LinkTimeout

LinkTopic

Name

Parent

Picture
  Does not support VB graphics methods

TabIndex

TabStop

Tag

Top

Visible

Width

Special Properties

AssignMode
  Controls behavior of PicBuf assignment statements, such as PicBuf1.FullPicture = PicBuf2.FullPicture. This affects whether or not two images will be combined (e.g. a montage), as well as how the images will be resized if they are combined into a single image. The four modes are total replacement, union composite, intersection composite, and resize composite. See also

ResizeMode, Xorigin, and Yorigin.

AutoScale
If set to TRUE, the screen image is automatically scaled to fit the control. It is important to note that resizing and scaling are quite different. See section on Scaling and Resizing below.

Bitmap
A handle to the pixel data of the entire image. Not available at design time.

ColorDepth
The quantity of color data per pixel, in bits (i.e. 1,4,8, or 24). Not available at design time. Read only.

FullPicture
A handle to a device independent bitmap that contains the full data of the image, regardless of screen resolution or color depth. Includes the bitmap, palette (if any), color depth, Xresolution and Yresolution. Not available at design time.

HeightPrint
Height, in pixels, of the portion of the image to be printed. See also LeftPrint, TopPrint, and WidthPrint.

HeightSelect
Height , in pixels, of the portion of the image that has been selected. See also LeftSelect, Selection, TopSelect, and WidthSelect.

IMKVersion
The version of ImageKnife in use. Contains two numbers, the first for the DLL, the second for the VBX file. Read only.

LeftPrint
Offset, in pixels, of the left edge of the portion of the image to be printed from the left edge of the full image. See also HeightPrint, TopPrint, and WidthPrint.

LeftSelect
Offset, in pixels, of the left edge of the portion of the image that is selected from the left edge of the image. See also HeightSelect, Selection, TopSelect, and WidthSelect.

Palette
Handle to the color palette, if one exists. Not available at design time.

PrinterhDC
The device context of the printer. Not available at design time.

ResizeMode
Controls behavior of PicBuf assignment statements, such as PicBuf1.FullPicture = PicBuf2.FullPicture. This mode allows you determine the method used for resizing the image if resizing takes place in the course of a PicBuf assignment statement. The four modes are delete/replicate pixels but keep aspect ratio constant, delete/replicate pixels but allow aspect ratio to float (i.e. allow image to be stretched), resample the image but keep aspect ratio constant, and resample the image but allow the aspect ratio to float. Resampling achieves a superior quality image at the expense of speed. It is important to note that resizing and scaling are quite different. See section on Scaling and Resizing below. See also AssignMode, Xorigin, and Yorigin.

ScreenPalette
Forces the palette of this control to be realized (i.e. become the palette in use for the entire screen) when this control receives the focus.

Selection
A handle to a device independent bitmap that contains the portion of the image defined by WidthSelect, LeftSelect, TopSelect, and WidthSelect. The Selection property can be used in lieu of the FullPicture property for most ImageKnife functions and assignment statements.

TopPrint
Offset, in pixels, of the top edge of the portion of the image to be printed from the top edge of the image. See also HeightPrint, LeftPrint, and WidthPrint.

TopSelect
Offset, in pixels, of the top edge of the portion of the image to be selected from the top edge of the image. See also HeightSelect, LeftSelect, Selection, and WidthSelect.

WidthPrint
Width, in pixels, of the portion of the image to be printed. See also LeftPrint, HeightPrint, and

TopPrint.

WidthSelect
Width, in pixels, of the portion of the image to be selected. See also LeftSelect, HeightSelect, Selection, and TopSelect.

Xorigin
Offset, in pixels, to be applied to X-axis during assignment statements. See also AssignMode, ResizeMode, and Yorigin.

Xpan
Offset, in pixels, of the left edge of the image from the left edge of the control. See also Ypan.

Xresolution
Width, in pixels, of the actual image. Note that this is not the same as the Width property, which defines the width of the control. Not available at design time. Read only. See also Xresolution.

Yorigin
Offset, in pixels, to be applied to Y-axis during assignment statements. See also AssignMode, ResizeMode, and Xorigin.

Ypan
Offset, in pixels, of the top edge of the image from the top edge of the control. See also Xpan.

Yresolution
Height, in pixels, of the actual image. Note that this is not the same as the Height property, which defines the height of the control. Not available at design time. Read only. See also Xresolution.

ZoomFactor
Magnification level of the screen image. 25% of actual size is expressed as a ZoomFactor of .25 and 200% of actual size is expressed as a ZoomFactor of 2.
The aspect ratio of the screen image is always fixed. Note that this only scales the screen image; it does not resize the actual image. See section on Scaling and Resizing below. See also AutoScale, ResizeMode, Xorigin, and Yorigin.

Functions

imkAppendTiff *(FileSpec, PicBuf.FullPicture, compression method)*
Stores a PicBuf image as the last image in a multiple image TIFF file.

imkBlur *(PicBuf.FullPicture)*
Returns a blurred image of a source PicBuf image.

imkBrightCont *(PicBuf.FullPicture, BrightnessPercent, ContrastPercent)*
Returns an image with adjusted brightness and contrast from a source PicBuf image.

imkColorReplace *PicBuf.FullPicture, RGB1, RGB2, flag, NewRGB)*
Returns an image that has all pixels that fall between *RBG1* and *RGB2,* inclusive, changed to *NewRGB.* If *flag* is 1, then all pixels **not** between *RGB1* and *RGB2,* inclusive are changed to *NewRGB.* In palettized images, the palette is not altered. If a value specified does not exist, the closest match in the palette is used instead.

imkCountImages *(FileSpec, format)*
Returns the number of images in a multi-image file (e.g. GIF or TIFF).

imkExtractEdges *(PicBuf.FullPicture, magnitude)*
Returns an image with the edges extracted from the source image, where *magnitude* is the amount of image processing to apply (1 to 10).

imkGamma *(PicBuf.FullPicture, magnitude)*
Returns an image with adjusted gamma from the source image, where *magnitude* is the amount of image processing to apply (1 to 10).

imkGetColor *(PicBuf.FullPicture, X, Y)*
Returns the color value of a pixel at *X, Y,* in the source image.

imkGetColorCount *(PicBuf.FullPicture)*
Returns the total number of colors actually used in the source image.

imkGetPalColor *(PicBuf.Palette, position)*
Returns the color value of the palette entry at *position* within the source palette.

imkGetPalette *(PicBuf.Palette, PaletteRecord)*

Puts the palette in the source palette into.....

imkGetScanLine *(PicBuf.Bitmap, LineNum, ScanLineArray)*
Gets an entire row of pixels at *RowNum* from the source image's bitmap and places them into a dynamic array.

imkGrayScale *(PicBuf.FullPicture)*
Returns a grayscale image of a source PicBuf image.

imkIncreaseColors *(PicBuf.FullPicture, depth)*
Returns an image with the color depth increased to *depth* from a source PicBuf image.

imkInit *(depth, width, height, background)*
Returns an image that is *width* by *height* pixels in area, *depth* bits deep (i.e. 1,4,8,24), and has all pixels set to the color *background.*

imkLoad *(FileSpec, ImageNumber, format)*
Returns an image from a file, where *ImageNumber* is the rank of the image within a multi-image file and *format* is the file type (e.g. TIFF).

imkLoadPal *(FileSpec, ImageNumber, format)*
Returns a palette from a file, where *ImageNumber* is the rank of the image within a multi-image file to obtain the palette from and *format* is the file type (e.g. TIFF).

imkMatrixFilter *(PicBuf.FullPicture, MatrixSize, Matrix, Scale, Norm, Offset)*
Returns a filtered image of the source using the matrix supplied in *Matrix,* which is a user-defined two-dimensional array that is *MatrixSize* by *MatrixSize,* where *MatrixSize* is either three or five. *Scale, Norm,* and *Offset* provide more detailed control of the filtering operation.

imkMirror *(PicBuf.FullPicture, Horizontal, Vertical)*
Returns a mirrored image of the source. *Hori* and *Vert* are booleans that, when set to true, will cause the image to be mirrored horizontally and/or vertically respectively.

imkNegate *(PicBuf.FullPicture)*
Returns a negated image (a negative) of the source image.

imkOptimizePal *(FullPicture(0), Number, Colors)*
Returns a palette that is a "best fit" between multiple PicBuf images, where *FullPicture(0)* is an array of the PicBuf.FullPictures to be used to create the optimized palette, *Number* is the number of elements in the array, and *Colors* is the number of colors to limit the new palette to (must be less than 256).

imkPutScanLine *(PicBuf.FullPicture, LineNum, ScanLineArray)*
Gets one scanline of raw RGB data from the source image at *LineNum* and puts it into *ScanLineArray,* which is an array of long where each element of the array will contain one pixel's worth of RGB data.

imkReduceColors *(PicBuf.FullPicture, Colors, Optimize, Dither, NoColorBleed)*
Returns a image that is the source image with its colors reduced, where *Colors* is the maximum number of colors that the new image will contain, *Optimize* is a boolean that when set to true will also create an optimized palette, *Dither* is a boolean that when set to true will dither the image, and *NoColorBleed* is a boolean that when set to true will reduce color bleeding. The bit depth of the new image will be set to the minimum that will accommodate the number of colors in *Colors* (i.e. 1, 4, 8, or 24 bits deep).

imkRemapPal *(PicBuf1.FullPicture, PicBuf2.Palette)*
Returns an image with the colors of the source image remapped to closest available color in *PicBuf2.Palette.*

imkRotate *(PicBuf.FullPicture, Angle, BackColor)*
Returns a rotated image of the source, where *Angle* is the amount of rotation to apply, in whole degrees and *BackColor* is the color to use to fill in areas that do not contain any image data after rotation (i.e. after a rotation that is not a multiple of 90 degrees). The returned image is resized to contain the rotated image in its entirety.

imkSetPalColor *(PicBuf.Palette, Position, Color)*
Sets the color value of a PicBuf palette at the index value specified by *Position* to be *Color.*

imkSetPalette *(NumColors, ColorValues)*
Creates a PicBuf palette where *NumColors* is the number of colors contained in the palette and *ColorValues* is an array of long containing *NumColors* elements, each with an RGB color value.

imkSetScreenPal *(PicBuf.Palette)*
Forces the screen palette to be the same as a given PicBuf palette.

imkSharpen *(PicBuf.FullPicture, Sharpness)*
Returns a sharpened image of the source where *Sharpness* is the amount of sharpening (from 1 to 10) to apply.

imkSoften *(PicBuf.FullPicture, Softness)*
Returns a softened image of the source where *Softness* is the amount of softening (from 1 to 10) to apply.

imkStore *(FileSpec, PicBuf.FullPicture, Format, Compression)*
Writes a PicBuf image to a file, where *FileSpec* is the path, *Format* is the file format to write it as (i.e. TIFF, TARGA, Windows Bitmap, GIF, DIB, or PCX), and *Compression* is the type of compression to use (i.e. none, LZW (GIF and TIFF), or RLE (PCX only)).

imkStorePal *(FileSpec, PicBuf.Palette, Format)*
Writes a file with a palette only (i.e. no image data), where *FileSpec* is the pathname, and *Format* is the format to write the file in (i.e. Windows Bitmap, GIF, or DIB).

imkVerify *(hDIB)*
Returns true if the integrity of a DIB is verified successfully.

Methods

Drag
LinkExecute
LinkPoke
LinkRequest
LinkSend
Move
Refresh
SetFocus
Zorder

Events

Change
Click
DblClick
DragDrop
DragOver
GotFocus
KeyDown
KeyPress
KeyUp
LostFocus
MouseDown
MouseMove
MouseUp
Paint

Albums

The album files that were created for this demo are a non-standard file format based on multi-image TIFF files. There is some ASCII text appended to the end of the album file that contains information needed by the demo, such as where the actual images live. The images contained in the album file are thumbnails of the original images.

When you create an album, the demo asks you to specify the color depth you want to use for the album.

All images added to the album will be color reduced to that color depth, both to help keep the file size of the album down and also to help ease palettizing problems.

When you add an image to an album, the demo color reduces it and then resizes it to thumbnail size in just a few lines of code. When you save an album, the demo only saves the thumbnail and the pointer back to the original image file.

Clicking on an image in an album with the Image Information window open will update the Image Information window with information about that particular image. Double-clicking on an image in the album will cause the demo to open an image processing window and load the original image file into it for image processing operations.

Loading an album

To load an album, we use imkCountImages to find out how many images we have to load, imkLoad to sequentially load the images from a multi-image TIFF file and Visual Basic's file handling routines to read the text at the end of the file.

```
' first, find out how many images are in the file
For i = 1 to imkCountImages (FileName, imk_TIFF)
  If i > 1 Then Load PicBuf(i)  ' create a new instance of the control if
needed
  ' now read the ith image into the ith element of the control array
  ' note that because album files use a .imk extension, we cannot use
  ' ImageKnife's automatic file type detection by extension
  PicBuf(i).FullPicture = imkLoad (FileName, i, imk_TIFF)
Next
' now we use VB to get the rest of the data at the end of the file.
```

Use of resizing with resample

The resizing of the thumbnails for the albums is done using ImageKnife's resampling capabilities. This is how you would go about doing this:

```
' This code fragment assumes that PicBuf1 contains the image we want to
resize
' and that we want to resize the image to fit in a 100 x 100 pixel area
PicBufInvis.AssignMode = 0  ' Force all xfers to be Total Replacements
' this next line creates a 100 x 100 pixel image
' that is all white in the invisible buffer
PicBufInvis.FullPicture = imkInit (PicBuf1.ColorDepth, 100, 100, RGB
(255, 255, 255))
PicBufInvis.AssignMode = 3  ' now change to Resize Composite mode
PicBufInvis.ResizeMode = 2  ' any resizing involving this PicBuf will use
                            ' resampling with a fixed aspect ratio
' This is where the magic happens. This will transfer the image in
PicBuf1 to the
' invisible PicBuf, resize it, and resample the image all in one step.
PicBufInvis.FullPicture = PicBuf1.FullPicture  ' now our image is the
right size
PicBuf1.AssignMode = 0  ' force all image xfers to be Total Replacements
' now just copy the resized image over the old one
PicBuf1.FullPicture = PicBufInvis.FullPicture
```

Use of color reduction

To handle the color reduction of the thumbnails (when the user requests it at album creation time, that is), we check to see what the color depth of the album is and then reduce colors if necessary.

```
' Figure out what the color depth is supposed to be
Select Case AlbumState.ColorDepth
  Case 16  ' if we're using 4 bits...
    If PicBuf.ColorDepth > 4 Then ' and the image is more than 4 bits deep
      ' reduce to 16 colors using optimization, dithering, and bleed
reduction
      PicBuf.FullPicture = imkReduceColors (PicBuf.FullPicture, 16, True,
True, True)
```

```
    End If
  Case 256 ' if we're using 8 bits...
    If PicBuf.ColorDepth > 8 Then ' and the image is more than 8 bits deep
      ' reduce to 236 colors (allow 20 for the system)
      ' use optimization, no dithering, and bleed reduction
      PicBuf.FullPicture = imkReduceColors (PicBuf.FullPicture, 236,
True, False,          True)
    End If
End Select
```

Use of imkAppendTiff and VB file functions

To save the actual album files, we use two imkFunctions: imkAppendTiff and imkStore.

```
' write the first image to the file as an uncompressed TIFF
' we do not compress the album thumbnails because they are generally
fairly small
' and because we are primarily interested in speed, not file size
imkStore Filename, (PicBuf(1).FullPicture), imk_TIFF, imk_uncomp
' now append the rest of the images onto the end of the file
For i = 2 To AlbumState.ImageCount
  imkAppendTiff Filename, (PicBuf(i).FullPicture), imk_uncomp
Next
' now we use VB to write our ASCII text info
```

Panning and zooming an image

Panning and zooming with ImageKnife is controlled with four PicBuf properties: Xpan, Ypan, AutoScale, and ZoomFactor. For the demo, we just hooked up scroll bars in the Image Processing window and fed the Value properties to the Pan properties.

```
Sub hsbXpan_Change ()
  PicBuf.Xpan = hsbXpan.Value
End Sub
```

We took the ZoomFactor into account by changing the scrollbars' Max properties whenever we changed the ZoomFactor.

```
hsbXpan.Max = ((PicBuf.Xresolution * PicBuf.ZoomFactor) - PicBuf.Width) /
PicBuf.ZoomFactor
vsbYpan.Max = (PicBuf.Yresolution * PicBuf.ZoomFactor) - (PicBuf.Height)'
* PicBuf.ZoomFactor)
```

Zooming was accomplished by simply setting the ZoomFactor to whatever the user requested from the View menu. The Fit in Window command simply sets AutoScale to true and resizes the PicBuf to fill the form.

Copying and cropping images

Copying and cropping images is all done using assignment statements and the AssignMode and ResizeMode properties. Assignment statements are the key to unlocking ImageKnife/VB's full capabilities. Assignment statements are used to do resizing, compositing, and transfers between PicBufs. You can freely interchange the Selection and FullPicture properties both in assignment statements and also in imkFunction calls.

To copy an image without resizing from one PicBuf to another:

```
PicBuf2.AssignMode = 0 ' total replacement
PicBuf2.FullPicture = PicBuf1.FullPicture
```

To copy a portion of an image to another PicBuf:

```
PicBuf2.AssignMode = 0 ' total replacement
PicBuf2.FullPicture = PicBuf1.Selection
```

To copy a portion of one image to a specific area of another PicBuf

```
PicBuf2.AssignMode = 0 ' total replacement
PicBuf2.Selection = PicBuf1.Selection
```

To composite two images together:

```
' set the coordinates for where you want the images to overlap
PicBuf2.Xorigin = 100
PicBuf2.Yorigin = 50
PicBuf2.AssignMode = 1 ' Union composite
PicBuf2.FullPicture = PicBuf1.FullPicture
```

To crop an image:

```
PicBuf2.AssignMode = 0 ' total replacement
' create a new image that is, presumably, smaller than the original
PicBuf2.FullPicture = imkInit (PicBuf1.ColorDepth, 100, 100, RGB (255,
255, 255))
' now xfer the image and the excess will be trimmed away
PicBuf2.FullPicture = PicBuf1.FullPicture
```

There are many other combinations that you can use to make the image you want.

Image Processing

All of ImageKnife's image processing power is handled through function calls to optimized C and assembler code routines. A simple example is mirroring an image. In the demo, we used code like this:

```
' to mirror it vertically...
PicBuf.FullPicture = imkMirror (PicBuf.FullPicture, False, True)
' to mirror it horizontally...
PicBuf.FullPicture = imkMirror (PicBuf.FullPicture, True, False)
```

This makes it possible to perform either non-destructive or destructive image processing.

Color manipulation

ImageKnife/VB provides a wealth of color manipulation capabilities. The demo only touches on a few of them with the Image Convert menu, which has already been discussed, and the Image Adjust Brightness/Contrast menu, which we will discuss here.

```
' this gets the values from two different horizontal scroll bars and
makes the changes
' We use the source image in our undo buffer so that our changes are all
made from
' the same beginning instead of making changes to changes
' the division by 100 is to convert our scroll bar values to a percentage
PicBuf.FullPicture = imkBrightCont (PicBufUndo.FullPicture,
hsbBrightness.Value / 100, hsbContrast.Value / 100)
```

All of ImageKnife's color manipulation routines, whether they be for palette manipulation or for gamma adjustment, operate in a similar fashion. They take very little code and are extremely powerful.

Summary

We hope you find this demo application a useful tool in determining ImageKnife/VB's capabilities. It was built largely as a learning tool by our technical support representative during his first month on the job for Media Architects.

If you would like a copy, it will be included as a fully-commented, fully-implemented Visual Basic sample application with ImageKnife/VB version 1.2. We hope that this demo will spur you to come up with your own creative uses for ImageKnife/VB, possibly by using portions of this demo within your own application. Regardless, we believe you will find ImageKnife/VB to be an easy-to-use, yet sophisticated image handling extension to Visual Basic that will enhance any VB application that uses images.

Michael Heggen
Technical Support
Media Architects, Inc.
CompuServe: 71662,371
Internet: mike@shower.rain.com
Telephone: (503) 297-5010
Fax: (503) 297-6744