@h = How to... Write applications using Visual Basic
@sf = Nick continues to add more functionality to the Doodler application in this month's installment

Last month, we added the ability to load existing images into our Doodler application. Now, we'll add the code to let you save any modifications you make.

@xh = Saving your picture

First of all, we need to add a "Save As..." menu item to the File menu. Open last month's project, click on <I>frmDoodler<D> in the form designer, and then click the <I>Menu Editor<D> button on VB's toolbar. Select the blank line below "&Open..." and change its caption to "Save &As...". Give our new menu item the name <I>mnuFileSaveAs<D> and click the right-facing arrow to the left of the <I>Next<D> button to let VB know that the new item belongs to the File menu. Finally, click OK and you will be returned to the form designer.

To add the code for the new menu item, choose <I>Save As...<D> from the <I>File<D> menu on the form itself, <B>not<D> from VB's File menu. Enter the following code:

```
@l = Call mDoSaveAs
```

As you can see, we'll be calling one of our own custom functions called <I>mDoSaveAs<D> to handle this. Press Ctrl-End and enter the following code:

```
@l = Private Sub mDoSaveAs()
@l =
@l =     Dim strFileName As String
@l =
@l = TryAgain:
@l =     strFileName = InputBox("Please enter a name for this image:", "Save As")
@l =     If strFileName <060><062> "" Then
@l =         If UCase(Right(strFileName, 4)) <060><062> ".BMP" Then
@l =             strFileName = strFileName & ".bmp"
@l =         End If
@l =         On Error GoTo TellUser
@l =         SavePicture picDrawingArea.Image, strFileName
@l =     End If
@l =
@l =     Exit Sub
@l =
@l = TellUser:
@l =     Dim lngResult As Long
@l =     lngResult = MsgBox("Unable to write the file specified." & vbCrLf & _
@l =                     "Reason - " & Err.Description & vbCrLf & _
@l =                     "Do you want to try another filename?", vbExclamation + _
@l =                     vbYesNo, "Save Failed")
@l =
@l =     If lngResult = vbYes Then
@l =         Resume TryAgain
@l =     End If
@l =
@l = End Sub
```

This code is pretty straightforward - it asks the user for a filename and then writes the current image to filename specified, handling any errors that might occur.

I've forced the filename entered by the user to end with a <I>bmp<D> extension. This has been done because the <I>SavePicture<D> function can only save bitmap (.bmp) files when writing image data from the <I>Image<D> property of a control. Notice that I've converted the filename to uppercase before checking the extension (if any). This is because the user might enter <I>BMP<D>, <I>bmp<D>, or <I>bMp<D> and so forth - converting the filename to uppercase means that the one test will cover all the possible cases. Start the application and try opening and saving a few images.

@xh = Superimposed Graphics

The observant amongst you might have noticed an inconsistency here - we load images into the <I>Picture<D> property of the picture box, but save them from its <I>Image<D> property. This is because the <I>Picture<D> property refers to the original image that was loaded, not the modified version that we have drawn onto. Any graphics methods such as <I>Line<D> or <I>Circle<D> are superimposed onto the loaded image rather than changing the image itself. Therefore, we use the <I>Image<D> property whilst saving, which refers to the original image plus the superimposed changes that we've made.

Speaking of superimposed graphics, have you noticed that the <I>Clear<D> button doesn't work quite as expected when an image has been loaded? This is because it only clears the graphics that we've superimposed (drawn) but leaves the underlying original image in place. To make the Clear button remove the loaded image as well, change its event handler to the following:

```
@l = Private Sub cmdClear_Click()
@l =     picDrawingArea.Cls
@l =     Set picDrawingArea.Picture = Nothing
@l = End Sub
```

The extra line we've added re-targets the <I>Picture<D> property at <I>Nothing<D>, i.e. no picture. This ensures that both the graphics that we've drawn, and any image the user might have loaded are both removed from the drawing area.

@xh = Adding the polish

Now that we've got the basic nuts and bolts in place, it's time to make Doodler behave more like a standard Windows application. Here is a list of improvements that immediately spring to mind:

***BULLET The <I>File<D> menu needs an <I>Exit<D> item
***BULLET The user should be able to choose filenames from the standard Windows dialog box rather than entering the name directly
***BULLET A toolbar should be provided to enable quick access to the standard Open and Save features
***BULLET There is no concept of a "current" file hence the lack of a <I>Save<D> item on the <I>File<D> menu.
***BULLET If the close box is clicked, the program quits immediately without offering the user a chance to save his/her work first
***BULLET You can't scroll around any large images that have been loaded – instead, they are clipped to fit inside the drawing area

@xh = Making an exit

Adding an <I>Exit<D> menu item is straightforward enough. Open VB's menu editor and add a new menu item to the end of the File menu. Set its name to <I>mnuFileExit<D> and its <I>Caption<D> to <I>E&xit<D>. Close the menu editor and choose the new menu item you've just added to form. Enter the following line:

@l = Unload Me

This tells your form to unload (remove) itself from memory, causing the program to exit. The variable <I>Me<D> is a special variable provided by VB which refers to the current module, which is <I>frmDoodler<D> in our case. It would have been just as valid to enter <I>Unload frmDoodler<D> instead. However, recall that VB doesn't automatically change your code if you rename any controls, forms, or other bits and pieces. Using <I>Me<D> rather than the form name (<I>frmDoodler<D>) means that if you rename the form in the future, your code will continue to work without requiring any changes to be made.

@xh = Don't use END

Users of older versions of BASIC might wonder why we aren't using the <I>End<D> statement even though VB supports this. The reason is that <I>End<D> doesn't perform an orderly shutdown of your program. As you are aware, when a form is loaded, it raises a <I>Form_Load <D>event into which you normally place any startup code such as setting defaults and so forth.

Correspondingly, there is a <I>Form_Unload<D> event which runs whenever a form is unloaded. Clicking the close box or programmatically unloading a form via the <I>Unload<D> statement causes VB to execute any code in the <I>Form_Unload<D> event. Therefore, using the command <I>Unload Me<D> is effectively the same as clicking the close box. However, the <I>End<D> statement doesn't go through this orderly shutdown sequence, it effectively pulls the rug out from underneath your feet. Using the <I>End<D> statement bypasses any code in <I>Form_Unload<D> and your program is immediately terminated. Since this behaviour is highly undesirable, I recommend that you use the <I>Unload<D> statement instead.

@xh = Déjà vu

Next on the agenda - a standard Open/Save dialog box. Most Windows applications share the same Open/Save dialog box to handle getting filenames to and from the user. These Open/Save dialog boxes are part of a collection of dialog boxes known as the <I>Common Dialogs<D>. Windows itself provides these services, saving us the need to write our own equivalents using VB. As you would expect, VB provides a fairly simple way to invoke these common dialogs. However, this functionality is not enabled by default – to enable it, do the following:

Choose <I>Components<D> from the <I>Project<D> menu
Scroll down the list of components and place a check mark next to the line that says <I>Microsoft Common Dialog Control 5.0<D>
Click OK

You should see that a new control has appeared in the toolbox - this is the common dialog control, which is VB's gateway to these standard dialog boxes.

Select the common dialog control from the toolbox and drag out a rectangle of any size above the drawing area. You'll notice that the rectangle you dragged immediately snaps back to a small picture of a common dialog control regardless of how large you made it. This is because the common dialog control is not visible when your program runs; it just sits on your form and makes its services available. To use it, you need to invoke one of its methods depending on which common dialog you want to use. Change the control's name to <I>dlgFileOps<D> and then choose <I>Open<D> from the <I>File<D> menu on the form. You'll see the subroutine call to load the file - right-click on <I>mDoOpen<D> and choose <I>Definition<D>. VB takes you to the function definition of <I>mDoOpen<D>. Change this subroutine so that it looks like the following:

```
@l = Private Sub mDoOpen()
@l =
@l =     Dim strFileName As String
@l =
@l = TryAgain:
@l =     With dlgFileOps
@l =         .Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly
@l =         .Filter = "All Files (*.*)|*.*|Bitmaps (*.bmp)|*.bmp|CompuServe GIF (*.gif)|*.gif|JPEG
(*.jpg)|*.jpg|Icons (*.ico)|*.ico|Windows Metafiles (*.wmf)|*.wmf|Run Length Encoded (*.rle)|*.rle|
Enhanced Metafiles (*.emf)|*.emf"
@l =         .ShowOpen
@l =         strFileName = .filename
@l =     End With
@l =
@l =     If strFileName <060><062> "" Then
@l =         On Error GoTo TellUser
@l =         picDrawingArea.Picture = LoadPicture(strFileName)
@l =     End If
@l =
@l =     Exit Sub
@l =
@l = TellUser:
@l =     Dim lngResult As Long
@l =     lngResult = MsgBox("Unable to open the file specified." & vbCrLf & _
@l =                     "Reason - " & Err.Description & vbCrLf & _
@l =                     "Do you want to try another file?", vbExclamation + _
@l =                     vbYesNo, "Open Failed")
@l =
@l =     If lngResult = vbYes Then
@l =         Resume TryAgain
@l =     End If
@l =
@l = End Sub
```

The core of the code is almost unchanged except the call to <I>InputBox<D>, which has been replaced with a call to invoke the Open Common Dialog instead.

@xh = In Closing

That's all for this month - you can find the accompanying project files on the cover disc as usual. Next month, I'll explain how this code works and we'll continue to enhance the Doodler application.

Cheers,
Nick.

Nicholas Scott is a freelance columnist who currently works for MIS Computer Services in Northwich. Nick can be contacted via email at nicks@miscs.com.

***BOX OUT
@sh = Where does it end?
A VB program ends when all of its forms have been unloaded. Therefore, it is important that a program unload the forms that it opens in order for it to shut down correctly. This hasn't affected us so far since all our projects have consisted of only one form, but future tutorials will deal with opening multiple forms.
***END BOX

***BOX OUT
@sh = An improved file open dialog box

***IMPROVED_OPEN
The new open dialog box, provided courtesy of the Common Dialog control. This dialog box is intelligent and will only accept valid filenames, making life as a programmer much easier.
***END BOX

***BOX OUT
@sh = A simple file open dialog box
***ORIGINAL_OPEN
This dialog box has provided the mechanism for getting filenames to and from the user so far. Although simple, it is hardly up to the standard of the usual Windows dialog box, which contains facilities for creating new directories and so forth. Compare this to the dialog box presented in figure 2, which is much more conventional.
***END BOX