How To Use DDRV Preview

Purpose
The intent of this document is to overview certain procedures and functions in ddrv.dll. It provides valuable insight into the use of the DDRV Print Preview system.

Overview
DDRV.DLL and DDRVEXE.EXE combine to allow the programmer to control a complete print/print preview system. The programmer writes a single set of code to control the printer and/or a print-preview screen. Basically, the DLL is called by the programmer to create a temporary file which describes the page. The EXE is run by the last call to the DLL and feeds the EXE the name of the temp file and some command-line options. The EXE opens the temp file and uses it's instructions to draw the page to the print preview screen and/or the printer. If you understand how the dll and exe interact, the process of creating a print-preview becomes easier to follow and more intuitive

What the previewer dll does not do
There are several things that some programmers are used to in the VB printer object, that DDRV Print Preview does not do. The print previewer does not keep track of your current position on the page. This is up to the programmer. For example, in VB you must set Printer.CurrentX and Printer.CurrentY any time you wish to change the current position. With DDRV, you pass the starting X and Y on each call. For example, to print a line of text you make the call:
rvText X!, Y!, align%, "This is some text"
The viewer is passed the x-coordinate, the y-coordinate, the alignment of the text, and the text.

Also, the viewer will may not retain font information correctly when switching from page to page forward and backward in the viewer. This information must be re-initialized at the beginning of each new page. This is important because of the random access nature of the file - no page must be dependent on the page before or after it.

The viewer is not a child of the calling program (at least not yet, I'm working on it); it is a separate executable. I do, however, provide calls to see if the viewer is active or not, to float it on top of other screens, and to close the viewer. The fact that the viewer is a stand-alone exe and not just a dll can be fairly advantageous as your program can have multiple viewers open at once. In version 2.0, I will allow the programmer to create a document, but not view or print it right away, just create the file. The programmer will then be able to open the document into the print previewer at his convenience.

Basic Calls - An example
Let us assume, for example purposes, that we have a list of names, phone numbers, and corresponding pledges (dollar amounts) that we would like to print to the screen or possibly just to the printer. We would also like to put a bitmap logo at the top of the first page and a footer containing a horizontal line and the page number at the bottom. We will assume the following declarations:

```
Type ListOfNamesType
        Name as String * 30
        Phone as String * 10
        Pledge as Double
End Type
```

```
Dim ThePeople(1 to 200) as ListOfNames
Dim NumberOfPeople as Integer
Dim CurPage%
```

For those of you not familiar with Visual Basic, I will quickly explain what may be obvious to some. We have defined a structure called ListOfNamesType. We then declared an array called ThePeople based on this structure. Memory is set aside for up to 200 members in this array. The "Type" keyword in VB is analogous to the "Record" keyword in Pascal or the "Struct" keyword in C. The members of the structure are: Name - an array of 30 characters, Phone - an array of 10 characters, and Pledge - a double precision float (8 bytes). I also define an integer called NumberOfPeople that will tell us how many members the array currently holds (a maximum of 200 as per the dimensioning of the array, ThePeople(1 to 200)). CurPage% is an integer that will tell us what page we're on while making DLL calls.

My job now, will be to explain how we transfer this information to the print preview screen or printer by calling the DLL. I will outline a simple routine below called PrintList.

```
'First, the subroutine declaration:
        in VB the declaration would look like this:
        Sub PrintList (ViewOrPrint%)
```

NOTE: for those users of C or Pascal, the declaration is of a subroutine called "PrintList" with an integer parameter (the % means integer in VB) called "ViewOrPrint". From here-on-out I will not bother to interpret VB code as it is fairly simple to follow. Just remember that any strings (ending in a $ or declared as strings) passed should be pointers to null-terminated strings (or PChars or ZStrings, or ...) in Delphi or C, in VB integers end in a %, singles (4-byte floats) end in a bang (!), doubles (8-byte floats) end in a #, and any comments are preceded by a '. The loop commands and if statements should be easy enough to follow.

```
'We also need some module-level variables to keep track of page position, etc ...
        Dim a%, b%, c%          'dim some integers for counting
        Dim cy!                 'this single will keep track of our current Y position on the page
        Dim lh!                 'this will be the line height
        Dim PageTop!            'the top page margin
        Dim PageBottom!         'the bottom of the page
'Now we should initialize our varables ...
        lh! = .1667             '6 chacraters per inch line height
        PageTop! = 1            '1inch is the top margin
        PageBottom! = 10        '10 inches is the bottom margin
```

When you wish to create a print preview screen the first thing that you must do is initialize the print previewer. This is done with a call to rvInit. This call is mandatory and must be placed before all other calls. As of this time, the integer returned by rvInit is meaningless. Just ignore it; in future releases, I may assign it a meaning. We assume here that the handle to the calling window is Form1.hWnd.

```
        a% = rvInit( Form1.hWnd, "A DDRV report viewer screen", RV_VIEWFIRST,  __
                RV_INCH,   RV_ZOOMFIT)
```

The call above passes the handle to the calling form, the string containing the title of the report, and some options. The first option is RV_VIEWFIRST - this tells the viewer to print preview, not just print the page to the printer. The other option that can be passed is RV_JUSTPRINT - this tells the viewer to print right to the printer. The next option is RV_INCH. This tells the viewer that all calls will be made in the inch system of measurement. The other constant that can be placed here is RV_CM - to make calls with centimeters as their scale. The final call tells the viewer to initialize by fitting the page to the size of the viewer. This constant is meaningless if we are not print previewing. The last parameter is bitwise, that is it can take several constants added together. The possible constants for the last parameter are:

RV_PORTRAIT or RV_LANDSCAPE - determine page orientation
RV_MAGNIFY - enables the magnifying glass zoom feature
RV_PRINTSELECT - enables the prompting of the user for a page range, etc.
RV_ZOOMxxx - tells the viewer the initial zoom state - either 100%, 75%, 50%, or FIT
RV_HIDEDEFPRINTER - hides the button which allows the user to change the default printer.
RV_SAVETEMPFILE - tells the viewer not to delete the temp file.

The above constants should be added together (or "Or'ed") when passing, e.g.

RV_PORTRAIT + RV_MAGNIFY + RV_ZOOM100

After the call to rvInit is called, we will now call an optional routine - rvViewerPos. This routine allows us to specify the initial position of the viewer window and hide or show the window control box. All parameters are passed in pixels.

```
Call rvViewerPos(10, 10, 400, 400, RV_CTLBOX)
```

The above call will initialize the viewer with a window drawn with it's upper left-hand corner at (10, 10) and it's lower right at (400,400) in screen pixels. We could have passed RV_MINIMIZE or RV_MAXIMIZE in the X1% parameter to specify a minimized or maximized initial window state. If we do this, the next three parameters (y1,x2,y2) become meaningless.

The next step will be to initialize the font parameters. It is very important to do this after each call to create a new page (rvNewPage) and after initializing the viewer (rvInit). This must be done before printing any text to the page. If you fail to initialize the fonts after each page, the viewer will print text with weird results, especially when going backwards through the pages.

```
Call rvFontName ("Arial")                ' Set the initial font name
Call rvFontSize (12)                     ' Set the initial font size
Call rvFontStyle (RV_FONTNORMAL)   'Set the font style
```

Note that the font name is passed as a string pointer in C or Pascal, and as a "ByVal" string in VB. The rvFontStyle call passes a single bitwise parameter as an integer. It may be any "Or'ed" combination of RV_FONTBOLD, RV_FONTITALIC, RV_FONTNORMAL, RV_FONTUNDERLINE, and RV_FONTSTRIKEOUT.

Now that we have initialized the viewer and set-up the font, we can start the fun part- drawing to the viewer. First we will place the graphic on the screen. To put a graphic on the screen, we call rvGraphic. The parameters of this call basically outline a rectangle to shrink the graphic to and specify a file name for the bmp. Note that the rvGraphic call will shrink or expand the bitmap to fit the defined rectangle. This means that you're 1" square log will print the same to the 72dpi screen, your 300 dpi laser II or your 600 dpi laser 4 plus - DDRV will force it to fit in the defined rectangle.

```
Call rvGraphic (.5, .5, 2.5, 1.5, "c:\mybitmap.bmp")
```

This call expands or shrinks the file c:\mybitmap.bmp to fit in a 2" wide, 1" high rectangle at the with it's upper left-hand corner at (.5,.5).

Now we will loop through our names and print them to the page. Watch what we do when we hit the end of the page. Remember that the variable cy! will keep track of our current y position on the page and the variable lh! holds the line spacing (6 lines per inch).

```
' -- The loop
        CurPage% = 0
        For a% = 1 to NumberOfPeople
' -- Check for a new page
                if a% = 1 or if cy! >= PageBottom! then
' -- If we aren't on the first page, we need to call rvNewPage to get a new page to draw on.
'   Remember to set the font information at the beginning of each new page
                        if a% > 1 then
                                Call rvNewPage
                                rvFontName "Arial"
                                rvFontSize 12
                                rvFontStyle RV_FONTNORMAL
                                rvLine .5, 10.25, 8, 10.25, 1
                        end if
' -- Now we draw the page line and page number at the bottom of the screen. The call to create
'   a new line just defines the upper left and lower right of the line and passes the line
'   thickness in points (this is 1 point line)
                        rvLine .5, 10.25, 8, 10.25, 1
' -- Print the page number
                        CurPage% = CurPage% + 1
                        rvText 8, 10.35, RV_RIGHT, "Page:" + Str$(CurPage%)
' -- Reset the y position on the page back to the top margin and we're on our way
                        cy! = PageTop!
' -- Print the Column headings in bold text - remembering to set the text back to normal when
'   done.
                        Call rvFontStyle (RV_FONTBOLD)
                        Call rvText (1.5, cy!, RV_LEFT, "NAME")
                        Call rvText (3.5, cy!, RV_LEFT, "PHONE")
                        Call rvText (6, cy!, RV_RIGHT, "Amount Pledged")
                        Call rvFontStyle (RV_FONTNORMAL)
                        cy! = cy! + 2 * lh!  ' increment the Y page position
                End If

' -- We will print the Person's name and phone aligned to the left in columns at 1.5" and 3.5" and
'   the amount of money pledged in a decimal aligned column at 6"
                Call rvText (1.5, cy!, RV_LEFT, ThePeople(a%).Name)
                Call rvText (3.5, cy!, RV_LEFT, ThePeople(a%).Phone)
                Call rvText (6, cy!, RV_DECIMAL, Format$(ThePeople(a%).Pledge, "$#0.00")
' -- Move the Y position place holder down on the page one line
                cy! = cy! + lh!
' -- Now we just loop back and print the next name
        Next a%

' -- Finally we place the call to end the document.
        rvEndDoc
```

Upon the call to rvEndDoc, DDRV.DLL closes the temporary file, and starts up the viewer

program - DDRVEXE.EXE.

Here is the complete sub-routine with-out comments. It may be easier to understand the program flow looking at it like this:

```
Sub PrintList (ViewOrPrint%)

        Dim a%, b%, c%          'dim some integers for counting
        Dim cy!                 'this single will keep track of our current Y position on the page
        Dim lh!                 'this will be the line height
        Dim PageTop!            'the top page margin
        Dim PageBottom!         'the bottom of the page

        lh! = .1667             '6 characters per inch line height
        PageTop! = 1            '1inch is the top margin
        PageBottom! = 10        '10 inches is the bottom margin

        a% = rvInit( Form1.hWnd, "A DDRV report viewer screen", RV_VIEWFIRST,  __
                RV_INCH,   RV_ZOOMFIT)

        Call rvViewerPos(10, 10, 400, 400, RV_CTLBOX)
        Call rvFontName ("Arial")                   ' Set the initial font name
        Call rvFontSize (12)                        ' Set the initial font size
        Call rvFontStyle (RV_FONTNORMAL)   'Set the font style

        Call rvGraphic (.5, .5, 2.5, 1.5, "c:\mybitmap.bmp")

        CurPage% = 0
        For a% = 1 to NumberOfPeople
                if a% = 1 or if cy! >= PageBottom! then
                        if a% > 1 then
                                Call rvNewPage
                                rvFontName "Arial"
                                rvFontSize 12
                                rvFontStyle RV_FONTNORMAL
                        end if
                        rvLine .5, 10.25, 8, 10.25, 1
                        CurPage% = CurPage% + 1
                        rvText 8, 10.35, RV_RIGHT, "Page:" + Str$(CurPage%)
                        cy! = PageTop!
                        Call rvFontStyle (RV_FONTBOLD)
                        Call rvText (1.5, cy!, RV_LEFT, "NAME")
                        Call rvText (3.5, cy!, RV_LEFT, "PHONE")
                        Call rvText (6, cy!, RV_RIGHT, "Amount Pledged")
                        Call rvFontStyle (RV_FONTNORMAL)
                        cy! = cy! + 2 * lh!  ' increment the Y page position
                End If

                Call rvText (1.5, cy!, RV_LEFT, ThePeople(a%).Name)
                Call rvText (3.5, cy!, RV_LEFT, ThePeople(a%).Phone)
                Call rvText (6, cy!, RV_RIGHT, Format$(ThePeople(a%).Pledge, "$#0.00")
                cy! = cy! + lh!
        Next a%
        rvEndDoc
```

That's not so bad, huh? Just remember what the program skeleton should look like:

1. Initialize the viewer
2. Initialize the font information
3. Make calls to draw the page
4. If your report will go more that one page, you need to check for page breaks with some sort of bottom margin check and then call rvNewPage to start a new page. Remember to initialize font info for each new page.
5. When you're done printing call rvEndDoc to start the viewer.

Printing Landscape

This is an important feature, especially in VB where controlling printer is just not a lot of fun. Controlling the orientation of the page is done in the first call - rvInit. Just pass the constant RV_LANDSCAPE in the options% parameter. Remember that this parameter is bitwise, so other options may be mixed in. For example, the following all start the viewer in landscape mode:

```
a% = rvInit(hWnd%, "A Title", RV_JUSTPRINT, RV_CM, RV_LANDSCAPE
a% = rvInit(h%, T$, VP%, RV_INCH, RV_ZOOMFIT +  RV_LANDSCAPE)
a% = rvInit(h%, t$, v%, uom%, RV_ZOOM50 + RV_MAGNIFY + RV_LANDSCAPE)
```

Drawing text at an angle

A new feature of DDRV is to be able to print text at an angle. This call is very simple:

```
Call rvAngleText(x!, y!, TheAngle%, TheText$)
```

x!, y! define the starting point for the text in inches, TheAngle% is the angle given in degrees counter clockwise, e.g. if TheAngle% = 90 the text will be printed straight up from x,y. If TheAngle% = 180, the text will be printed upside down. The font style is taken from the current font settings.

Printing out paragraphs of text

This is a fairly complicated issue. There are two ways to approach the problem.
1. I have a dll routine called rvParagraph which will word-wrap text within a given rectangle.
2. You can use an invisible multi-line text box to hold the text, size the box to the correct width, and font settings, and then read the lines one at a time out of the text box. This sounds complicated, but can be easily abstracted into a VB subroutine (see below).

You may be thinking "Why in the world would I wan to use an edit box to store the text." Well, the rvParagraph control cannot sense a page break and will not tell you how far down on the page the text printed. It should only be used when you have control over the text. If you know you have only 200 characters to print and they are not all uppercase, it is a good bet that they will fit in just 2 or 3 lines so you could use rvParagraph as follows

```
Call rvParagraph (1,1, 7.5,10, TheText$)
```

This call will wordwrap the string given by TheText$ in the bounding rectangle defined by the points (1,1) and (7.5,10). You could then assume that it is safe to start printing your next text about 4 lines down from this.

The second method of printing a paragraph is somewhat more complicated, but a lot more powerful. As you know, a multi-line edit box automatically word wraps the text inside of it. The width and font specs of the multi-line edit box can be adjusted to simulate a paragraph on a page. I provide calls to the dll to tell you how many lines are in the box and retrieve any single line in the box. You can use these calls in conjunction with rvText to print a paragraph one line at a time out of a pre-formatted invisible editbox. For example, the following subroutine can be used to print a paragraph of text.

```
Sub PrintAParagraph(txtEdit as Control, SomeBigStringParagraph$)
        Dim a&, NoPages&, NoChars&
        Dim cy!, lh!
        Dim Buffer as String * 1000

        lh! = .16667      'Set the line height variable to 6 lines per inch

' -- Set up the edit box
        txtEdit.Visible = False
        txtEdit.Text = SomeBigStringParagraph$
        txtEdit.FontBold = False
        txtEdit.FontItalic = False
        txtEdit.FontName = "Arial"
        txtEdit.FontSize = 12

' -- Set the scalemode to inches and size the edit box to six inches wide
        txtEdit.ScaleMode = 5
        txtEdit.Width = 6

' -- Set up the dll font specs
        rvFontName "Arial"
        rvFontStyle RV_FONTNORMAL
        rvFontSize 12

' -- Make a call to find out how many lines are in the edit box
        NoLines& = 0    ' If we pass a zero in the second parameter, the parameter will return
                        ' containing the number of lines in the edit box. Buffer is unused
in this                 ' call, but must be passed anyway.
        Call rvGetEditBoxLine(txtEdit.hWnd, NoLines&, Buffer)

' -- Loop through the lines and print them out, checking for the end of a page
        cy! = 1
        for a& = 1 to NoLines&
                ' This parameter (NoChars&) is passed containing the
                ' line index in the list box whose string want to obtain
                NoChars& = a&
                Call rvGetEditBoxLine(txtEdit.hWnd, NoChars&, Buffer)
                ' When NoChars& returns after the call, it contains the
                ' number of characters copied into Buffer
                rvText .5, lh!, RV_LEFT, Left$(Buffer, Int(NoChars&))
                ' increment the y position
                cy! = cy! + lh!
                ' Check for the end of the page
                if cy! > 10 then rvNewPage: cy! = 1
        Next a&
End Sub
```

The above routine could be easily embellished into your own paragraph printing routine. It should serve as a good base for how to print a paragraph of text using the edit box. Some notes:
1. Make sure that the edit box width is given correctly, as it will be the approximate width of the wrapped text when printed.
2. Be sure that the variable Buffer is pre-dimensioned to hold enough characters or you will get a general protection fault. e.g. either do what I did above - Dim Buffer as String * 1000 or Dim Buffer$ and on the next line do Buffer$ = Space$(1000)
3. The text when printed to the paper tends to be a little wider than when printed on the screen. You can experiment and compensate for any difference. This discrepancy appears to be due to the resolution difference between the screen and printer.

What does rvText use for the decimal place holder

In different countries a different character may be used for a decimal place holder - in the US it's a period (.). In Europe it is a comma (,). DDRV reads the decimal character from the [intl] section of the win.ini file in the sDecimal key word area. This initially set by the Window's setup program when Windows is installed.

Controlling the viewer

You have seen how the initial viewer state can be controlled by the rvInit and rvViewerPos calls. I have also mentioned that the state of the viewer after it has been created may be monitored.

rvGetHandle, rvViewerActive, rvFloatViewer, rvGetCloseHandle, rvKillViewer

rvGetHandle passes no parameters and returns the handle of the last viewer created. There are a few simple rules about using rvGetHandle:

1. It must be called AFTER rvEndDoc as the viewer handle is not assigned until after this call is made.
2. It only holds the value of the last viewer created, and returns zero after the user exits

e.g.

```
...
rvEndDoc
hWnd% = rvGetHandle
```

If it is your intent to let the user create more than one preview screen at one time, you may wish to store these values in an array so you can use them later.

rvViewerActive takes the viewer handle as a parameter and returns 0 as long as the viewer window exists. It can be used to monitor when the viewer is closed.

e.g.

```
hWnd% = rvGetHandle
Do Until rvViewerActive(hWnd%) <> 0
        DoEvents
Loop
```

The above code will halt your program routine until the users exits the viewer. I realize that this function may not be needed because rvGetHandle will tell you if the last viewer is still active. It does however, have a use if you allow more that one viewer on the screen at once and need to be able to control more than just the last viewer.

rvFloatViewer can make the viewer float on top of all other windows on the desktop. (also, it can return the viewer to it's original "non-floating" state). It takes the window handle and a true/false value as parameters. if the true/false value not zero then the viewer will be made to float. Otherwise it will be made not to float.
e.g.

```
hWnd% = rvGetHandle
rvFloatViewer hWnd%, 1          'Float the viewer
...
rvFloatViewer hWnd%, 0          '"Un-Float" the viewer
```

Incidentally, this function can be used to float any form, not just a DDRV viewer form. Feel free to use it as you wish.

rvGetCloseHandle is used to get the handle of the close button on the viewer form. This is passed to the rvKillViewer routine to terminate a view screen. It returns the close button of the last viewer created, or zero if the form no longer exists or if a viewer has not yet been created. This call must be made AFTER the call to rvEndDoc, as it's value is set on a call-back by the viewer to ddrv.dll after the viewer is created. As with the call to rvGetHandle, it is my suggestion that the programmer store these values in an array for future use, then when the user exits the program, you may call rvKillViewer for each member in the array to ensure that all viewers are closed.

rvKillViewer can be used to destroy a viewer screen. It takes the handle returned by rvGetCloseHandle as a parameter. A technical note, the handle returned is actually the handle of the close button on the viewer form. When you call rvKillViewer, all it does is send a message to the button saying that the user clicked the mouse button on it. There are other ways of terminating a window, but this method insures that you get all of your resources back and that the temp file the viewer is reading gets closed and deleted. e.g.

```
hWndClose% = rvGetCloseHandle
Call rvKillViewer (hWndClose%)
```

The order to make calls

For some reason, this seems to confuse the most people - "How do I know when to call what?" Well let's outline it ...

1. **rvInit** is always called first to start creating a document
2. **rvViewerPos** should be called next if you need to use it - it is optional
3. Call **rvButtons**, **rvButtonCaption** next if you need to.
4. **rvFontName**, **rvFontStyle**, **rvFontSize** should be called right after **rvInit** and after each call to **rvNewPage**.
5. Any of the text or graphics routines may now be called. To start a new page call **rvNewPage** - remember to specify font info after calling **rvNewPage**.
6. **rvWhatIsTempFile** needs to be called before **rvEndDoc** if you need to get the name of the temp file.
7. **rvEndDoc** ends the document - no more viewer routines should be called except those listed below.
8. The following must be called after **rvEndDoc** to have any effect:
   rvGetHandle, **rvGetCloseHandle**, **rvFloatViewer**, **rvViewerActive**, **rvKillViewer**

Other Calls

DDRV.DLL contains other page formatting calls - rvButtons to change the button graphics, rvButtonCaption to change the tool-bar hints, rvRect to draw rectangles,etc. Please refer to DDRV_VB.WRI and DDRV_DEL.WRI for more information about these functions. They are fairly

straight forward.

Creating a document without printing or print previewing

There may be situations where you wish to create a document and save it for the user to view or print later. This version of DDRV Print Preview will allow the programmer to do this. With the risk of being repetitive, I will quickly explain once again how ddrv.dll and ddrvexe.exe combine to allow print previewing.

1. Calls are made to ddrv.dll in order to create a temporary page description file.
2. When rvEndDoc is called, the document is considered complete and the file is closed by ddrv.dll.
3. ddrv.dll now shells to ddrvexe.exe passing the name of the page description file and some other parameters on the command line.
4. ddrvexe.exe then opens the temp file and uses it's contents to draw the print preview.
5. when the user exits the preview screen, the temp file is erased.

It is very important to remember that ddrv.dll makes no more page drawing calls to the exe after rvEndDoc is called - DDRVEXE.EXE is an independent program. For completeness, it is necessary to point out that ddrvexe.exe does place calls to the dll after it starts. The main point is that the page description file is no longer added to by ddrv.dll once the viewer is started up.

What I allow the user to do now is to pass the constant RV_CREATETEMP to rvInit in the ViewOrPrint% parameter. This tells the dll NOT to start the viewer and NOT to start printing after rvEndDoc is called and the temp file is NOT erased. Anytime before calling rvEndDoc and after calling rvInit, the programmer may call the function rvWhatIsTempFile(FileName$). This function fills the FileName$ character buffer with the name of the temp file ddrv.dll is writing to, and returns the number of characters written into the buffer. Again, the FileName$ buffer must be pre-dimensioned by the user to avoid a general protection fault - i suggest 129 bytes to be on the safe side.

e.g

```
Dim NoChars%
Dim FileName$

FileName$ = Space$(129)        'remember to get space for the string

NoChars% = rvWhatIsTempFile(FileName$)
MsgBox "The file name is: " & Left$(FileName$, NoChars%)
```

Once you have the page description file, you can re-name it and put it where you want. Now you have a document, comparable to a PageMaker or Word file which you can call up in the print previewer or print on a whim. The only thing left is to explain what command-line parameters you must pass ...

ddrvexe.exe must have the following parameters passed to it, in this order.
1. The name of the temp file
2. P or a V for View or Print
3. The Options% as specified in rvInit. (convert the integer to a string)
4. The handle of the calling window (This isn't used for much now, but you should pass it anyway.)

For example suppose you have a page description file called my_page.tmp

From VB (or another language that can call an api or has a shell command) you could shell to ddrvexe.exe like this ...

```
Options% = RV_PORTRAIT + RV_ZOOMFIT + RV_MAGNIFY + RV_PRINTSELECT
CommandLine$ =  "ddrvexe.exe my_page.tmp V " + Trim$(Str$(AnyOptions%)) + " " + ___
                    Trim$(Str$(Form1.hWnd))
e% = Shell%(CommandLine$, 1)
```

This example uses the VB command "Shell." If you need to shell and aren't using VB, you could use the windows api call ShellExecute to accomplish the same thing. Remember to pass the capital letter V or P as the second parameter to indicate whether you wish to print or view on the screen. Also remember that the Options% parameter is just an integer comprising the same Options% parameter values as rvInit will pass - RV_PORTRAIT, RV_LANDSCAPE, RV_MAGNIFY, RV_PRINTSELECT, RV_ZOOM100, RV_ZOOM75, RV_ZOOM50 , RV_ZOOMFIT, or RV_HIDEDEFPRINTER. These options are bitwise, so add them ("Or" them).

A final tip on displaying saved temp files - use the RV_SAVETEMPFILE constant in the Options% parameter of the rvInit function to tell the user not to delete it. When the viewer closes, you may then rename, move, delete, etc. the temp file with your code.