Printing

by Ron Gery

Abstract

This article concerns the basics of printing under Windows. It does *not* cover the setting up of a printer and its Device Context (DC), concentrating instead on operations needed to get output to the printer after it is set up. The use of the main printing functions, **StartDoc**, **EndDoc**, **StartPage**, **EndPage**, **AbortDoc**, and **SetAbortProc**, are discussed, as are the principles of banding.

Basic Steps

The sequence of events used for a printing operation are as follows:

1. Create the Device Context (DC) for the printer
2. Set an AbortProc to handle possible abortive conditions (for example, out-of-disk-space or user cancellation)
3. Start the document
4. Output to the printer DC and advance through the pages as needed
5. End the document
6. Clean up (printer DC, AbortProc instance)

This article assumes that the first step has already been accomplished. This step can be greatly simplified by using the **PrintDlg** common dialog.

New Functions for Windows 3.1

While the methodology of printing has not changed from Windows 3.0 to Windows 3.1, some new functions exist in Windows 3.1 to simplify the application interface. Under Windows 3.0, the **Escape** function is the primary means of controlling the flow of a print job. While this interface continues to function under Windows 3.1, a set of functions was added to GDI to separate the printing control operations from the device-specific escapes. The following table lists the 3.1 functions, the escapes that they replace, and their functionality:

3.1 function	***corresponding escape***	***purpose***
StartDoc	STARTDOC	start a document
EndDoc	ENDDOC	end a document
StartPage	implicit part of NEWFRAME and STARTDOC	start a new page

EndPage	NEWFRAME	end current page
SetAbort Proc	SETABORTPROC	set the AbortProc for print job
AbortDoc	ABORTDOC or ABORTPIC	abort a document

The mapping between the 3.1 functions and the escapes is one-to-one with the exception of the NEWFRAME escape, which is replaced by two functions, **StartPage** and **EndPage**. The NEWFRAME escape's function was to end the current page and start the next page; the STARTDOC escape was responsible for indicating the start of the first page. The **StartPage** and **EndPage** functions work to better define the interface and are paired around every page being printed, including the first.

While both approaches work under Windows 3.1 for backward compatibility, applications written for Windows 3.1 and later are strongly encouraged to use the functions interface.

Very Simple Example

The following code demonstrates the bare essentials of printing using the Windows 3.1 functions:

```
// for a print session this small, the AbortProc won't actually come into
// play, but this is an example of a non-aborting version.
NonAbort(HDC hPrintDC, short uCode)
{
   return(1);              // continue printing
}

// this routine prints a single 100x100 rectangle in the upper left corner
// of the page.
BOOL PrintRect(HDC hPrintDC)
{
   FARPROC lpAbortProc;
   DOCINFO diInfo;
   char DocName[5] = "hello";

   lpAbortProc = MakeProcInstance((FARPROC)NonAbort, hInst);
   SetAbortProc(hPrintDC, lpAbortProc);

   diInfo.cbSize = sizeof(DOCINFO);
   diInfo.lpszDocName = (LPSTR)DocName;
   diInfo.lpszOutput = NULL;
   StartDoc(hPrintDC, (LPDOCINFO)&diInfo);
   StartPage(hPrintDC);
   Rectangle(hPrintDC, 0, 0, 100, 100);      // actual output to printer
   EndPage(hPrintDC);
   EndDoc(hPrintDC);
   // clean up AbortProc instance
   FreeProcInstance(lpAbortProc);
}
```

This example performs the bare minimum needed for a print job. The abort procedure is essentially empty and does not allow for user input, no explicit banding is supported, and only a single page is printed.

Start and End of a Document

An application starts a print job using the **StartDoc** function (or the STARTDOC escape). The function accepts two parameters, the printer DC and a **DOCINFO** structure that identifies the document. The **lpszDocName** field specifies the name of the document; the name is used for display in the Print Manager window. The **lpszOutput** allows an application to specify an output file for the printer that could be different from the one set up in Control Panel. For example, if a printer is set up to output to "FILE:" an application can query the user for the output file name and use the actual name to start the print job. This avoids the invocation of GDI's dialog box requesting the name of the output file. After performing some spooler preparation work and bookkeeping, the DC is ready for real printing to begin. The state of the DC is saved at this point; it is restored by GDI for the start of every page and band. This is slightly different in Windows version 3.0 where the state of the DC is saved at **CreateDC** time instead.

With the escape-only method, STARTDOC also defines the start of the document's first page.

The **EndDoc** function (or the ENDDOC escape) marks the end of a print job. All needed clean up is done at this point, and the DC is restored to its state immediately preceding the call to **StartDoc**.

Start and End of a Page

The **StartPage** marks the start of a new page of output. Some bookkeeping is performed by GDI to prepare the DC for the page, but nothing really exciting takes place. The Windows 3.0 interface did not have the pure concept of a page start; it was implicitly defined by an output operation following a STARTDOC escape for the first page, a NEXTFRAME escape for subsequent pages, or by a NEXTBAND request after the previous page was done banding.

Predictably, the **EndPage** function denotes the end of a page. This is same functionality as the NEXTFRAME escape or of the last NEXTBAND escape on a page. At the printer level, this function causes a page eject, while at the GDI level a virtual page eject is performed by restoring the DC to its state at the time of the **StartDoc** call. If GDI is handling banding for the application (more on that below), the end of the page is when the actual banding work is carried out. The DC is now ready for another page of output.

Banding

Banding is the process in which a single page of output is generated using one of more separate rectangles, or bands. When the bands are all placed on the page, a complete image is the result. This approach is often used by raster printers that do not have sufficient memory or ability to image a full page at one time. Banding devices include most dot matrix printers as well as some laser printers.

An application determines if a printer is a banding printer by calling the **GetDeviceCaps** function with the RASTERCAPS index and then checking the RC_BANDING bit. When this bit is set, the device is a banding device. If the bit is not set, the driver outputs a full page at a time. Because GDI can transparently handle any necessary banding for banding printers, applications are not required to explicitly support banding regardless of how the RC_BANDING bit is set.

The advantages of using banding over allowing the GDI simulations are improved printing speed and possible reduction of disk space needed for printing. Speed can be increased by selectively performing output based on the band rectangles, thereby eliminating unneeded calls. Disk space is saved by avoiding GDI's banding support, which uses a disk metafile to represent the page image.

Standard

An application indicates that it intends to explicitly handle banding by calling **Escape** with NEXTBAND as its operation. This tells GDI that the application is doing its own banding as well as requesting the next band from the printer. In response to the NEXTBAND escape, the printer returns the bounding rectangle for the current band; all subsequent output to this band are automatically clipped to this rectangle. The dimensions of the bands depend on the driver and sometimes the amount of memory available in the system. When there are no more bands to process on a page, the printer returns an empty rectangle for the band. Notice that the rectangles do not necessarily band in the y direction; because the banding is usually performed based on the paper as it is situated in portrait mode, documents using landscape orientation may band in the x direction.

In order to explicitly use the device's banding, an application loops through the bands on a page and outputs to each band as needed. Because output is clipped to the current band, the application does not have to limit its output to the band's rectangle, but pre-clipping often speeds up printing. Once the empty band is encountered, the application moves to the next page.

The NEXTBAND escape is supported on non-banding drivers by defining a single band that encompasses the full page in order to remain consistent with the banding model. On these printers, an application gets two bands per page, the first one being the full page band and the second being the empty band that marks the end of the page.

At the start of the document, GDI calls **SaveDC** to save the state of the printer DC. For every NEXTBAND escape, GDI calls **RestoreDC** so that any attribute changes performed during the band are undone, and the DC is restored to its state at the time the document was started.

The following code uses the Windows 3.1 printing functions in conjunction with the NEXTBAND escape to print multiple pages of a document while banding each page:

```
ret = 1;                    // assume nothing has gone wrong
for (i = 1; (ret >= 0) && (i < NumPages + 1); i++)
{
   StartPage(hDC);
   // for real bands, output the document
   // once the empty band is encountered, end the page
   while ((ret = Escape(hDC, NEXTBAND, NULL, NULL, lpRect)) >= 0 &&
         !IsRectEmpty(lpRect))
   {
      PageOutput(i, lpRect);   // output to lpRect on page i
   }
   EndPage(hDC);
}
```

There is one difference in the way the Windows 3.0 interface handles banding that also applies under Windows 3.1 when the printing-specific functions are not used (i.e. only escapes are used). Basically, an application is not allowed to use *both* the NEXTBAND and NEXTFRAME escapes. An application that wishes to do its own banding cannot use the NEXTFRAME escape; in order to proceed to the next page, the application calls the NEXTBAND escape again after the last band (the one with the empty rectangle) of the previous page. Continuous banding causes page breaks when appropriate. Similarly, an application using the NEXTFRAME escape is assumed to be working in full-page mode, and it should not use the NEXTBAND escape. An application written for Windows version 3.1 should call **StartPage** and **EndPage** for all pages, regardless of whether or not it is banding.

Ignoring Banding

An application does not need to be aware of banding. By using only the **StartPage**/**EndPage** (or NEWFRAME escape) interface with no NEXTBAND escapes, an application indicates that GDI should handle all of the banding procedures. To accomplish this, GDI records all of the output calls into a metafile that represents that page. When the page is finished (**EndPage** is called by the application), GDI steps through the band on the page (using the NEXTBAND escape) and plays the metafile into every band. When the page's output is completed, the metafile is deleted and a new one is created for the following page. All of the metafile processes are completely transparent to the application.

Using a metafile to simulate a full-page image may suggest that the output capabilities are restricted by the limits of metafiles, but this is not the case. Because the metafile is associated with a real device, any operation that can be performed on a regular output DC can be performed on this metafiling DC. Query functions return values based on the printer, and functions like **DrawText** can be used because they are simulated using queried values. Also, the scaling limitations of metafiles do not apply since the metafile is never scaled-- it is played at its defined size.

There is one subtle limitation in GDI's banding support. If an application blts a color bitmap directly to a color printer, the metafile simulation does not produce the same results as manual banding. The metafile is built with a record containing a DIB representation of the bitmap; if the printer driver does not support the **GetDIBits** functionality, GDI simulates by building a monochrome DIB, and the color information is lost. An application that finds itself in need of blting a color bitmap to a color printer has to perform its own banding in order to achieve the best results. If color DIBs are used, this limitation does not exist.

The banding metafile is not used when an application performs its own banding.

Text vs. Graphics and the BANDINFO Escape

Certain printers, notably the HP LaserJet family under low memory conditions, separate output into two passes, one for text and one for graphics. After the text composed of built-in and downloaded fonts is placed on the page, a second pass is made to output graphics primitives to the page. While the difference between the two passes is meaningless for an application that performs a complete page drawing to every band, applications that wish to selectively draw to bands should be aware of this behavior. A text operation is any output performed using the **TextOut** or **ExtTextOut** functions.

The BANDINFO escape is used to get information about the band. Returned is a BANDINFOSTRUCT that identifies the band. The **bifName** field identifies the name of the band. The **bifType** field specifies the type of band. It can be one of BIF_ROCK, BIF_REGGAE, BIF_OOMPAH, BIF_JAZZ, BIF_CLASSICAL, BIF_METAL, or any of the other types that can be arbitrarily defined.

The BANDINFO escape is used to get information about the nature of the current band. When called after a new band has begun (after a NEXTBAND), this escape tells an application whether the printer is expecting text or graphics or both for this band. A driver does not have to support the BANDINFO escape. If the escape is not supported, this indicates that every band handles both text and graphics. To determine if an escape is supported by a driver, an application uses the QUERYESCSUPPORT escape. Also, a driver not supporting this escape will simply return 0 if the escape is attempted

The BANDINFO escape accepts two BANDINFOSTRUCT parameters, one for input and one for output. On input (the *lpInData* parameter), the application specifies what type of output it intends to put on this page and, if graphics are to be output, a graphics bounding rectangle. This input is only meaningful when used with the first band on the page. It is used by the driver to optimize banding--no graphics bands will be used if they are not needed. If the application passes a NULL *lpInData*, the driver will expect a full page of text and graphics and band accordingly.

The driver uses the output BANDINFOSTRUCT (the *lpOutData* parameter) to specify the nature of the current band. If only the text flag is set (**fTextFlag** set to TRUE), the band is a text-only band. All graphics output is ignored for this band. If during the text band, text is encountered that requires glyphs to be generated in the graphics band (see below for more details), graphics bands will be used even if the application specified that only text is being output to the page (see above). In bands where the graphics flag is set (**fGraphicsFlag** set to TRUE), the printer is expecting graphics operations. Text operations that involve fonts used in the text band are ignored. If both flags are set, the band is a graphics band, and the printer is also expecting text output that could not be performed in the text band.

The complication in the scheme is that depending on the font being used, it is possible to output text during the graphics band. The example found in Windows 3.0 is vector fonts; for Windows 3.1, TrueType glyphs are output to the graphics band if they are rotated or if the font is larger than the threshold for downloading or if the user selected "Print TrueType as Graphics" during printer setup. As a result of this expanded definition of what text output means, printer drivers often set *both* the text and the graphics bits when the current band is a graphics band.

Applications should not make the assumption that a text band always exists. Given sufficient memory, the Windows version 3.1 driver for the LaserJet printers usually has only one band total that combines text and graphics.

Optimizing

By using the rectangle information that defines a band, an application can optimize its output to eliminate unnecessary work by the system. For example, if the current band covers the first 300 lines of the page, outputting a graphics primitive that appears between lines 600 and 900 results in the system doing much work to get the primitive ready but ultimately clipping it out. An application that pre-clips its output to the bands can speed up its printing on a banding printer, especially if the output requires extensive simulations in GDI. On the system end of things, a graphics or text object is rarely clipped to the destination before final output time, long after any needed simulations have been performed.

It is valid and efficient to avoid graphics operations during the text band. On the

other hand, an area that deserves caution is optimizing text output on printers that have separate text and graphics bands. Because text may appear in the graphics band depending on the font used, it is not valid to assume that text output can be ignored in a graphics band (unless all text was output in the text band and the **fTextFlag** was not set for the graphics band).

Scaling Factors

It is common for printers that use text and graphics bands to allow the two bands to be at different resolutions (the text band remains at the highest possible resolution). For example, a LaserJet configured for 75dpi will output the text band at 300dpi while outputting the graphics band at 75dpi. This discrepancy allows printer fonts to always look their best while at the same time speeding up the printing of graphics by using less resolution. This effect is achieved by using a different scaling factor for the two types of bands.

The GETSCALINGFACTOR escape returns the scaling factor for the current band. The value is an exponent of 2 (so that when a value at text resolution is shifted right by the scaling factor, the result is the value at graphics resolution). If the escape is not supported or the scaling factor is 0, there is no scaling taking place. In the LaserJet at 75dpi example, the first band (the text band) has a factor of 0 and all subsequent bands on the page (the graphics bands) have a factor of 2.

At the application level, all output is performed at full resolution at all times, and applications do not have to worry about the scaling factor. Coordinates are scaled when appropriate as part of the normal coordinate mapping done by GDI to convert logical units to device units. The only time the scaling becomes a concern is while performing pixel-specific processing, where 4 pixels at a logical 300dpi could map to a single pixel at 75dpi. Also, because clipping regions are specified in device units, a new clipping region designed for the scaled resolution needs to be built for the scaled bands.

Aborting a Print Job

A print job can be aborted in a variety of ways (discussed below), all sharing the same result. Any data that has not yet been sent to the printer is flushed out. Data that has already reached the printer cannot be recalled and is printed. The printer is notified of the abort and recovers as best it can. If the Print Manager is being used, the job is removed from the queue. If GDI was handling the banding, the banding metafile is deleted. The DC is returned to its state at the start of the document. The application is still responsible for freeing its AbortProc function instance.

AbortProc

The **SetAbortProc** function (as well as the SETABORTPROC escape) is used to set up what is known as the AbortProc. This AbortProc is a function that resides in the application and is called by GDI during a print job to inform the application of spooler errors and to allow the application to abort the job when desired. The AbortProc is called with information about why it is being called; this value is either an error code from the spooler or zero, which indicates that the function is being called simply to allow an abort.

The AbortProc is called routine during several steps of the printing process:

- after every write to the printer port when printing directly to printer (no spooling)
- after every write to a file when printing directly to a file (no spooling)
- after every write to the spooler file when spooling
- periodically when out of disk space for spooling due to other spool jobs
- before playing every metafile record when GDI is performing banding simulation
- occasionally from some older printer drivers

When the AbortProc is called, the application has the choice of either continuing the print job by returning a nonzero value or aborting the print job by returning zero.

Only the fourth case in the above list actually results from an error condition during printing. In this situation, the spooler is indicating that sufficient disk space will exist for the print job to complete once one or more of other print jobs currently being spooled are completed. By returning a nonzero value, the application indicates that it wishes to continue waiting for that disk space.

A big problem with the AbortProc given in the simple example above is that it does not allow other applications to run during the print job. A more system-friendly AbortProc processes pending messages before returning:

```
NonAbort(HDC hPrintDC, short uCode)
{
   MSG msg;

   while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
   return(TRUE);
}
```

Application Aborts

An application that decides to abort a print job on its own (i.e. without using the AbortProc) can do so by using the **AbortDoc** function or the ABORTPIC escape

(also called ABORTDOC). This has the same effect as returning a zero from the AbortProc. This approach to aborting a print job makes sense if the application decides to abort while performing intensive operations where it would not be convenient to wait for the AbortProc routine to be called.

Using a "Print Cancel" Dialog

Most applications present the user with a chance to abort a print job by providing a dialog box with a cancel button or a variation on that theme. There are several time slices during a print job that an application can use for checking for a user cancellation of the printing. When the AbortProc function is called, the printing process is yielding to the application for exactly such purposes; this is a good place for checking for user input. When the application itself is performing some time intensive operation, it can yield to the abort-checking code when desired.

Once the user has indicated that the print job should be cancelled, the application has the choice of using its AbortProc or directly calling the **AbortDoc** function. The **AbortDoc** function should not be called from within the application's AbortProc.

System Aborts

If GDI is handling the banding for an application, there exists a possibility that the printing will be aborted by GDI due to system errors. In these cases, GDI sends an ABORTPIC escape to the printer driver and cleans up the current banding metafile. The application is notified of the abort with an error return from the **EndPage** function (or NEXTFRAME escape).

End.