

# Bitmap printer driver

Jay Chadderwala



## Bitmap printer driver

The three basic components of the print subsystem are the application, GDI and spooler. On a print command from any application, Win32 GDI functions of GDI graphics engine are called, which in turn either spools the drawing instructions as an enhanced metafile (EMF) file or renders a printable image that can be sent to the spooler in conjunction with a printer driver. Spooler components interpret EMF files, and they can insert page layout information and job control instructions into the data stream. The spooler then sends the data stream to the serial, parallel, or network port driver associated with the target printer's I/O port.

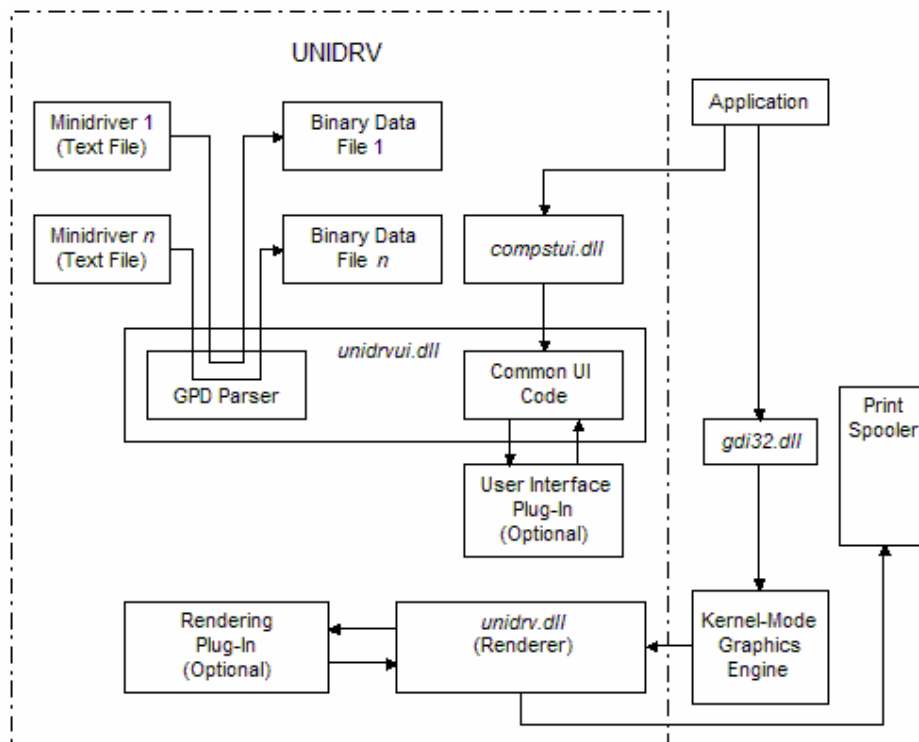


Figure 1: Unidrv Components

Unidrv consists of the following components:

- A **printer interface DLL** provides both a user interface to the driver's configuration parameters and an interface the spooler can call to notify the driver of print-related system events. To customize the printer interface DLL, one must provide one or more user interface plug-ins. This DLL is referred to as a user interface plug-in or just UI plug-in.

A UI plug-in can modify the printer property sheet by adding, removing, or replacing options within the property sheet's Device Settings page. It can also add a new page. Likewise, the plug-in can modify the document property sheet by adding,

removing, or replacing options within the property sheet's Layout, Paper/Quality, and Advanced pages, or it can add a new page.

- A **printer graphics DLL** assists GDI in rendering a print job, and sends the rendered data stream to the print spooler. To customize printer graphics DLL, one must provide one or more rendering plug-ins. This DLL is referred to as a rendering plug-in.

A rendering plug-in can modify the rendered image or scan line data stream, or insert Postscript code at specific injection points, before the data stream is sent to the spooler. It can also provide customized versions of some graphics DDI rendering functions.

The IPrintOemUni COM interface is the means by which the printer graphics DLL for Unidrv communicates with a rendering plug-in. The IPrintOemUni interface is implemented by each rendering plug-in.

## ❖ **How to write Bitmap printer driver**

The entry point for writing a bitmap driver is to write a rendering plugin that implements **IPrintOemUni::ImageProcessing** of COM Interface IPrintOemUni. IPrintOemUni::ImageProcessing is callback method called quite early in the Unidrv rendering code path. It can be used to modify bitmap data, perform color formatting or half toning. The method can return the modified bitmap to Unidrv or send it directly to the print spooler. For Bitmap printer driver return modified bitmap buffer to Unidrv.

ImageProcessing callback gets access to bitmap buffer data one band at a time. Rendering plug-in should buffer the band data every time it is called. The ImageProcessing implementation is also responsible for filling in the BITMAPINFOHEADER and COLORTABLE structures that is necessary for dumping the data out to a bitmap file. In a nutshell, the ImageProcessing method does the processing required to get to a point where we can dump the bitmap data out to file. The important points to remember while implementing it are:

- Set parameters DevBPP and DevNumOfPlanes to 0 in the GPD file.
- Set parameter IPCallbackID to a non-zero number in the GPD file.
- Set value TRUE for \*ppbResult in ImageProcessing method as mentioned:  
(\*ppbResult = (LPBYTE)(INT\_PTR)(TRUE);)
- Return S\_OK from ImageProcessing.

Other functions needed to implement are OEMXXXX functions, which are non-COM based device driver interface hook functions. Basic sequence of calling such DDI hook functions are as follows:

- **OEMStartDoc** – called by GDI engine when it is ready to start sending a document to driver for rendering. In this function we get the name of document being in print operation. It is passed by application and hence application dependant.
- **OEMStartPage** – called by GDI when it is ready to start sending the contents of physical page to the driver for rendering. It will be called for each page in document. If you want each page saved as a separate bitmap file, you should perform the dumping of the bitmap data per page in this function.

- **OEMEndDoc** – called by GDI when it has finished sending document to driver for rendering. It is the best place for dumping the bitmap data into file, which can result in all the pages being part of a single output bitmap file or last page of the document if you want each page as bitmap file.

If you are looking for post-processing (like watermark on the image, copyright or company's logo, etc...) of the bitmap file the best way to implement it will be **IPrintOemUni::FilterGraphics**, which is used to modify scan line data before it is sent to the print spooler.

GPD (generic printer description) files are used for creating Unidrv minidrivers (which are text files that contain descriptions of printers). The GPD file contains entries in GPD language to provide printer commands, printer options, printer font descriptions and printer attributes etc... Example of such GPD language identifiers are DevBPP(no. of bit per pixels), DevNumOfPlanes (number of color planes supported by the printer), and IPCallbackID (positive numeric identifier), and many more.

If bitmap driver requires user to be asked for name and path for the output bitmap file then printer interface DLL needs to be written as well. It must have a prompt dialog which asks user for output file information and such information can be passed to rendering plug-in via Device object (PDEV) data structure.

## ❖ **Conclusion**

One can easily write a bitmap printer driver based Microsoft Universal driver model and implement interface and non-COM based DDI methods provided by that model. One may also provide user configuration by writing printer interface DLL component and implicit configuration of bitmap image (like height, width, DPI, etc...) by configuring GPD file for printer graphic DLL (driver file).