CHAPTER 13: PRINTING WITH GDI: MOVING BEYOND THE SCREEN

This chapter explores the application of GDI for printing, delving into the similarities and differences compared to using it for the video display. While the concept of device independence remains largely applicable, some key distinctions emerge.

Similarities:

* GDI Functions: Many of the same GDI functions used for drawing on the screen can be utilized for printing text, lines, shapes, and other graphics elements on paper.
* Device Independence: The device-independent nature of GDI allows programs to write code without worrying about specific printer hardware details. The underlying system handles the translation and formatting for different printers.
* Drawing Concepts: Fundamental concepts like line styles, colors, brush styles, and text rendering are consistent between screen and printer output.

Differences:

* Printer Capabilities: Printers are not simply a paper-based video display. They have limitations in terms of speed, graphics support (some cannot handle bitmaps), and paper handling (e.g., page ejection).
* Output Speed: Printing is significantly slower than drawing on the screen. Programs need to be mindful of potential performance bottlenecks and optimize accordingly.
* Page Management: Unlike the reusable video display surface, printed pages require separate handling. Each page needs to be initiated with StartPage, completed with EndPage, and ejected upon completion.
* Document Organization: Unlike the windowed environment of the screen, printing output from different applications requires separation into distinct documents or print jobs.
* Printer-Specific Functions: GDI offers additional functions like StartDoc and EndDoc specifically for managing printer output and organizing pages.

Additional Resources:

* Chapter 15: Printing Bitmaps.
* Chapter 17: Printing Formatted Text.
* Chapter 18: Printing Metafiles.

These chapters provide further information on printing specific data formats using GDI and its related functions.

PRINTING PROCESS: A DETAILED BREAKDOWN

This section dives deep into the intricate process of printing in Windows, highlighting the interactions between the application program, the GDI module, the printer driver, and the print spooler.

Initiating the Process:

CreateDC or PrintDlg: The program acquires a handle to the printer device context, triggering the loading and initialization of the printer driver module (if necessary).

StartDoc: This function marks the beginning of a new document, handled by the GDI module. It subsequently calls the printer driver's Control function, preparing the device for printing.

Page Delimiters:

StartDoc/EndDoc: These calls bookend the normal GDI functions used for drawing page content.

StartPage/EndPage: These calls further delimit individual page boundaries within the document.

Metafile Creation:

For each page, the GDI module initially stores the drawing commands in a disk-based metafile (.EMF) for most printers.

This metafile acts as an intermediary representation of the page content.

Banding:

* High-resolution printers often use "banding" to divide the page into smaller, manageable sections.
* GDI obtains the band dimensions from the driver and sets a clipping region accordingly.
* The Output function within the driver then translates the metafile drawing commands for each band.
* This process, called "playing the metafile," ensures efficient handling of large print jobs.

Driver Output Generation:

* Each band requires translation into printer-specific output format.
* For dot-matrix printers, this involves control sequences and graphics commands.
* Laser printers with high-level languages like PostScript generate output in that specific language.

Temporary File Storage:

* The driver-translated output for each band is stored in another temporary file (.SPL) by the GDI module.
* This file acts as a buffer before handing over the entire print job to the spooler.

Print Spooler Interaction:

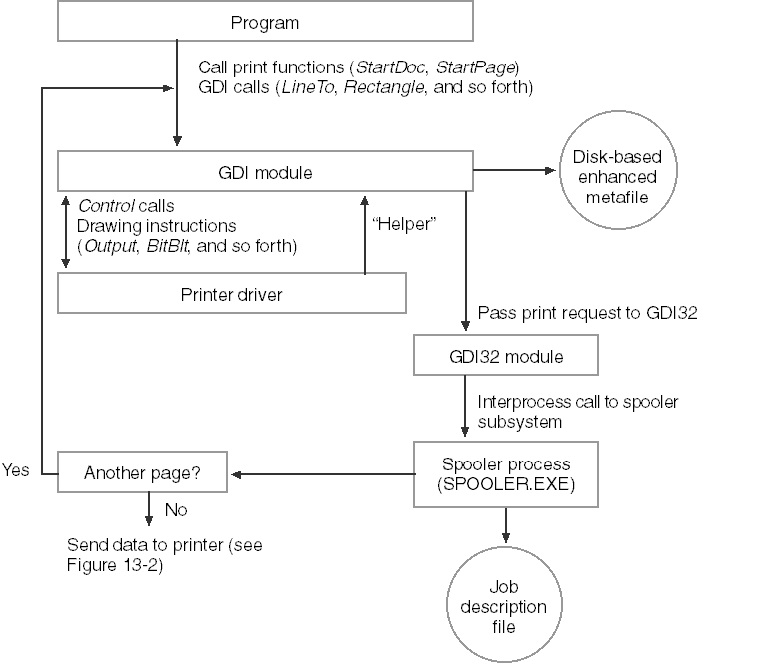
* Once the entire page is processed, the GDI module informs the print spooler about the new print job.
* This allows the spooler to manage and prioritize printing tasks for multiple applications.

EndDoc and Completion:

* After all page processing is complete, the program calls EndDoc to signal the end of the print job.
* This allows for cleanup and finalization of the printing process.

Key Points:

* Printing involves a complex interaction between the application, GDI, printer driver, and print spooler.
* Metafiles act as an intermediate representation of page content for efficient processing.
* Banding helps handle large print jobs on high-resolution printers.
* Driver translation converts drawing commands into printer-specific output formats.
* Temporary files store intermediary data for efficient spooling and management.



THE WINDOWS PRINT SPOOLER: A BREAKDOWN OF ITS COMPONENTS

The Windows print spooler is a complex system consisting of various components working together to ensure efficient and smooth printing. Here's an in-depth breakdown of each component and its respective function:

1. Spooler:

Acts as the central hub, receiving print requests from applications and managing the entire printing process.

Routes the data stream containing the print job to the appropriate print provider.

2. Local Print Provider:

Handles print jobs destined for locally connected printers.

Creates and manages spool files containing the print job data in a format suitable for the specific printer.

3. Network Print Provider:

Responsible for handling print jobs directed towards network printers.

Similar to the Local Print Provider, it creates and manages spool files but for network destinations.

4. Print Processor:

Performs the crucial step of "despooling," which involves converting the device-independent spool file data into a format specifically understandable by the target printer.

This ensures compatibility and proper interpretation of the print job by the printer hardware.

5. Port Monitor:

Manages the communication port to which the printer is physically connected.

Oversees data transfer between the spooler and the printer, ensuring accurate and reliable transmission of the print job.

6. Language Monitor:

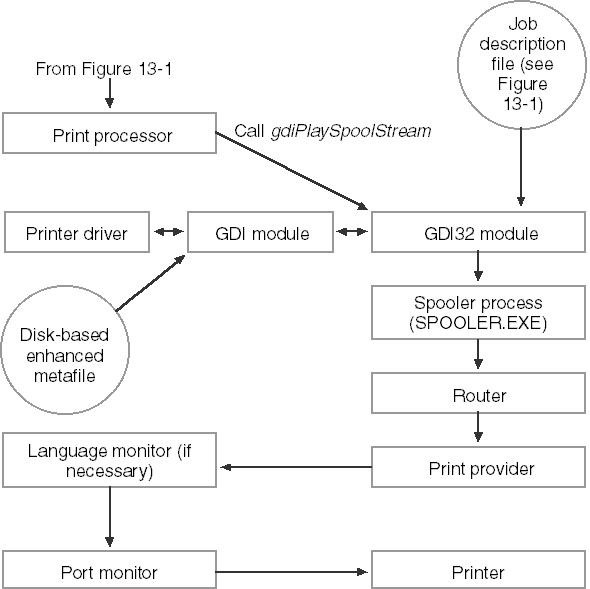
Applies to printers capable of two-way communication.

Enables configuration settings and status monitoring of the printer directly from the spooler.

This provides additional control and feedback about the printing process.

Benefits of the Print Spooler:

* Reduced workload on applications: By handling the intricacies of print job processing and communication with the printer, the spooler offloads this burden from applications, allowing them to focus on their primary functionality.
* Improved performance: Spooling allows applications to continue working while the print job is being processed and sent to the printer, resulting in faster response times and smoother workflow.
* Queued printing: The spooler manages a queue of print jobs, allowing multiple jobs to be submitted while ensuring they are printed in the correct order.
* Background printing: Users can continue working on their computers while the print spooler quietly transfers the print job to the printer in the background, maximizing productivity.



PRINTING PROCESS TRANSPARENCY AND VARIATIONS

While the printing process seems transparent to the application, understanding its nuances is crucial for developers. Here's a deeper look into the variations and potential implications:

1. Transparency for Applications:

Applications experience "printing" only during the time GDI saves the printer output to disk files.

This allows them to continue working while the spooler handles the actual printing, improving responsiveness.

2. Spooler Bypass:

Printing without the spooler is technically possible by disabling it in the printer properties.

Reasons for bypassing the spooler might include:

* Using a faster hardware or software spooler.
* Printing on a network with its own spooler.
* Avoiding the performance overhead of two spoolers.

This approach removes disk storage for print jobs but can potentially slow down the application until printing is complete.

3. GDI Direct Output:

When the spooler is inactive, GDI directly transmits printer output to the port, bypassing file storage.

While faster, this method can hold up the application program until printing finishes.

4. Metafile Variations:

GDI typically stores drawing functions in a metafile used for each band defined by the printer driver.

For drivers not requiring banding, the metafile is skipped, and GDI directly sends functions to the driver.

Alternatively, the application can manage band division, adding complexity but relieving GDI of metafile creation.

5. Potential Problems:

Printing can involve more overhead than video display usage.

Issues like GDI running out of disk space during file creation require handling and user feedback.

6. First Step: Obtaining a Printer Device Context:

The first step in printing from an application is acquiring a printer device context.

This involves functions like CreateDC or PrintDlg and triggers driver loading and initialization.

OBTAINING A PRINTER DEVICE CONTEXT

This section delves into the intricacies of acquiring a printer device context, crucial for printing from your application.

1. Device Context Handle:

Similar to interacting with the video display, printing requires a printer device context handle. This handle serves as the communication channel between your application and the printer driver, allowing you to issue drawing commands.

2. StartDoc and StartPage:

Before issuing drawing commands, you need to signal the start of a new document with StartDoc and the start of a new page with StartPage. These functions inform the system about your printing intentions and prepare the printer driver for receiving commands.

3. Creating the Device Context:

Two main approaches exist for obtaining the printer device context:

Standard Print Dialog (PrintDlg): This function displays a dialog box allowing the user to choose a printer and customize printing options. Upon selection, PrintDlg provides the application with a printer device context handle.

Direct Creation (CreateDC): This function offers more control and avoids displaying a dialog box. It requires the application to provide the printer's device name, which can be obtained through functions like EnumPrinters.

4. CreateDC Syntax:

The syntax for CreateDC when dealing with printers differs slightly from the video display:



* szDeviceName: Pointer to a character string containing the specific printer's device name.
* pInitializationData: Generally set to NULL.

5. Finding Available Printers:

Since multiple printers can be attached to a system, determining the available options becomes crucial. This is achieved using the EnumPrinters function, which fills an array of structures containing information about each attached printer.

6. Getting the Default Printer Device Context:

The GetPrinterDC function shown in Chapter 13 provides a platform-independent approach to retrieving the default printer's device context. It works under both Windows 98 and Microsoft Windows NT.

7. Choosing the Right PRINTER\_INFO\_x Structure:

The specific PRINTER\_INFO\_x structure to use with EnumPrinters depends on the desired level of detail and the operating system version:

* Windows 98: PRINTER\_INFO\_1
* Microsoft Windows NT: PRINTER\_INFO\_2 or PRINTER\_INFO\_4
* Windows 10 and 11: For Windows 10 and 11, the recommended PRINTER\_INFO\_x structure for using with EnumPrinters is PRINTER\_INFO\_4. This structure provides a good balance of information about each printer without being overly detailed.

Here's a breakdown of the available structures and their compatibility:



Reasons for using PRINTER\_INFO\_4:

* Provides important information like printer name, driver name, and location.
* Relatively compact structure size compared to other options.
* Efficient for enumerating large numbers of printers.

Alternatives to PRINTER\_INFO\_4:

* You can still use PRINTER\_INFO\_1 if you only require basic information like printer name.
* For more detailed information like printer properties and capabilities, consider using GetPrinterDriver and GetPrinter functions with the appropriate structures like DRIVER\_INFO\_x and PRINTER\_INFO\_2.

Conclusion:

* Choosing the appropriate PRINTER\_INFO\_x structure depends on your specific needs and the operating system you are targeting. For Windows 10 and 11, PRINTER\_INFO\_4 offers a good balance of information and efficiency for enumerating printers.
* Understanding the different methods for obtaining a printer device context and the intricacies involved in choosing the appropriate approach empowers you to effectively initiate printing from your application and interact with the chosen printer for subsequent drawing commands.