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:

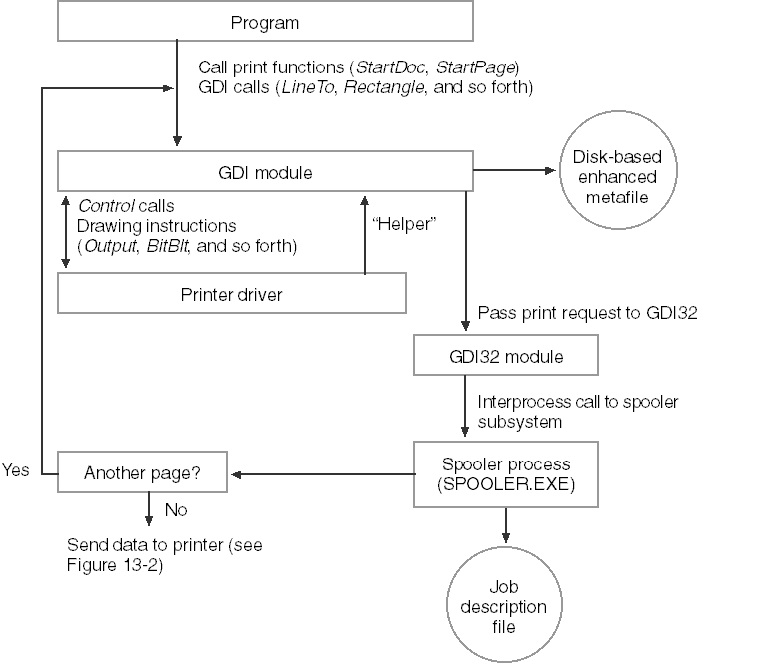
* Once the entire page is processed, the GDI module informs the print spooler about the new print job.
* This allows the spooler to manage and prioritize printing tasks for multiple applications.

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:

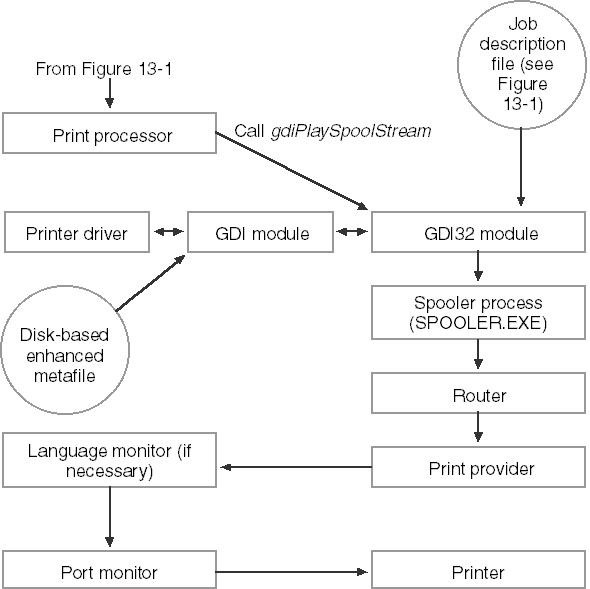
Applies to printers capable of two-way communication.

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.

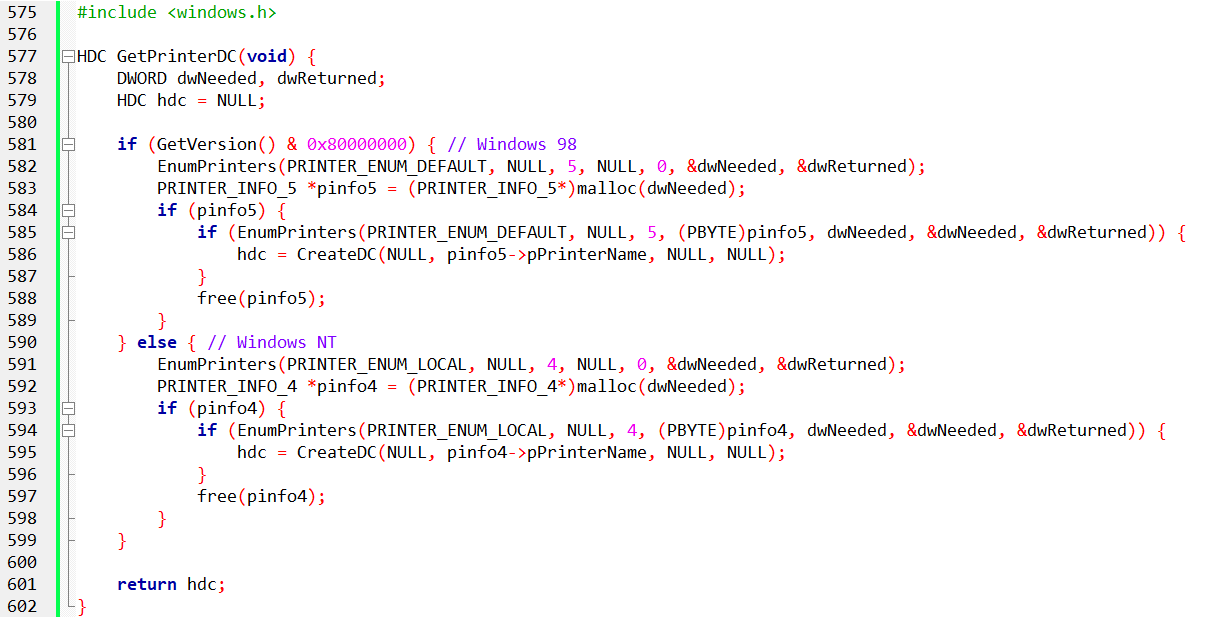
Alternatives to PRINTER\_INFO\_4:

* 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.

GETPRINTERDC FUNCTION: EXPLAINED



The GetPrinterDC function serves a crucial purpose in printing applications: it retrieves the device context handle for the default printer. Understanding its operation is crucial for effective printing.

1. Version Check:

The function first utilizes GetVersion to determine the operating system: Windows 98 or Windows NT.

2. Structure Selection and Memory Allocation:

*Windows 98:*

EnumPrinters is called twice:

* First call: Determines the required size for the PRINTER\_INFO\_5 structure.
* Second call: Allocates memory for PRINTER\_INFO\_5 and retrieves information about the default printer.

CreateDC creates the device context handle using the obtained printer name from pinfo5.

Allocated memory is then freed.

*Windows NT:*

Similar approach as Windows 98, but uses the PRINTER\_INFO\_4 structure instead.

3. CreateDC and Device Context Handle:

The chosen structure (pinfo5 or pinfo4) provides the printer name, which CreateDC utilizes to create the device context handle (hdc). This handle is ultimately returned by the function.

4. Efficiency and Documentation:

The function employs a two-pass approach with EnumPrinters to optimize memory usage. The specific structures (PRINTER\_INFO\_5 for Windows 98 and PRINTER\_INFO\_4 for Windows NT) are chosen based on their efficiency and ease of use, as mentioned in the Microsoft documentation.

5. Conclusion:

GetPrinterDC demonstrates the dynamic selection of appropriate structures based on the operating system and prioritizes efficiency while retrieving the necessary information for obtaining the default printer's device context handle. This functionality is essential for printing applications to interact with the chosen printer and issue drawing commands.

DEVCAPS2 PROGRAM

* The DEVCAPS2 program is a Windows application written in C that provides a graphical user interface to display various device capabilities information for both the video display and all printers attached to the system. It is an extended version of the original DEVCAPS1 program, displaying more detailed information obtained through the GetDeviceCaps function.
* Upon execution, the program creates a window with menus allowing the user to select either the video display or one of the available printers for which they want to view device capabilities. The user can also choose from different categories of information, such as basic information, other information, curve capabilities, line capabilities, polygonal capabilities, and text capabilities.
* The program utilizes the GetDeviceCaps function to retrieve information about the specified device context (either the video display or a printer). It covers a wide range of device capabilities, including dimensions (size in millimeters and pixels), color information (bits per pixel, color planes), and various other capabilities like the number of brushes, pens, markers, fonts, and available colors.
* The program dynamically populates the menu with the names of all local and remote printers attached to the system. It also provides an option to view printer properties, which opens a dialog displaying additional information about the selected printer.
* The graphical user interface is implemented using the Windows API, and the window displays the chosen device and information category. The information is presented in a readable format, and the program uses a fixed system font for consistent text rendering.
* In addition to displaying basic information, the program delves into more detailed capabilities, such as clipping, rasterization, curve capabilities, line capabilities, polygonal capabilities, and text capabilities. For each category, the corresponding capabilities are presented in a clear and organized manner, allowing users to understand the capabilities of the selected device.
* Overall, DEVCAPS2 serves as a practical tool for users and developers to explore and understand the capabilities of their display devices and printers and to gather valuable information about the features and limitations of the connected hardware. The program's structure is modular, with distinct functions handling different aspects of device capabilities, making it easy to extend or modify for future enhancements.
* The initialization process involves setting up the window class and creating the main window. The program responds to user actions such as selecting a different device or information category through menu options. It dynamically updates the menu with available printer names and provides a straightforward interface for users to interact with.

Window Procedure (WndProc):

The program defines a window procedure (WndProc) that handles various messages sent to the application window. It includes cases for window creation, command handling, initialization of menu items, painting the window, and handling the destruction of the window.

Initialization (WM\_CREATE and WM\_SETTINGCHANGE):

In the WM\_CREATE case, the program retrieves device context information, such as the average character width and height, to determine the size of characters in the system font.

The WM\_SETTINGCHANGE case is used to update the menu when the system settings change, such as when a new printer is added or removed.

Menu Handling (WM\_COMMAND and WM\_INITMENUPOPUP):

The program responds to menu commands in the WM\_COMMAND case. It distinguishes between selections related to the display (IDM\_SCREEN), printers, and properties. It also allows the user to choose different information categories, such as basic information, other information, curve capabilities, etc.

The WM\_INITMENUPOPUP case enables or disables the "Properties" menu item based on whether the current selection is the screen or a printer.

Painting the Window (WM\_PAINT):

In the WM\_PAINT case, the program prepares and displays information about the selected device and information category. It creates a device context for information retrieval (hdcInfo), selects a system-fixed font, and uses TextOut to display the gathered information.

Basic Information (DoBasicInfo):

The DoBasicInfo function displays fundamental information about the selected device, such as size in millimeters, resolution in pixels, color depth, number of brushes, pens, markers, fonts, and available colors.

Other Information (DoOtherInfo):

The DoOtherInfo function provides additional details about the selected device, including driver version, technology type, and capabilities related to clipping and rasterization.

Bit-Coded Capabilities (DoBitCodedCaps):

The DoBitCodedCaps function handles bit-coded capabilities for curves, lines, polygons, and text. It interprets bit flags to determine whether specific capabilities are supported by the device.

Structures (BITS and bitinfo):

The program defines structures (BITS and bitinfo) to organize and store information about various capabilities, making the code more readable and maintainable.

Memory Management (Dynamic Allocation):

The program dynamically allocates memory to store information about printers using malloc and frees the memory using free when it is no longer needed.

Printing Properties (IDM\_DEVMODE):

The program allows the user to view printer properties by selecting the "Properties" menu item. It calls PrinterProperties with the handle to the selected printer.

Cleanup (WM\_DESTROY):

In the WM\_DESTROY case, the program posts a quit message to terminate the application. Before exiting, it releases resources and cleans up by calling PostQuitMessage and returning 0.

Overall, the DEVCAPS2 program combines a well-organized structure with Windows API calls to create a graphical interface that efficiently presents detailed information about the capabilities of the selected display or printer.

The modular design allows for easy expansion or modification of functionality related to device capabilities on the Windows platform.

PRINTER PROPERTIES DEEP DIVE

The "Properties" option in DEVCAPS2's Device menu invokes a dialog box generated by the chosen printer driver. This dialog allows users to configure settings like paper size and orientation.

Dialog Origin:

* The dialog box is not part of DEVCAPS2 itself.
* It is triggered by the printer driver's ExtDeviceMode function.
* This function allows users to configure various settings, with at least paper size being a standard option.
* Most drivers also offer portrait and landscape orientation options.
* Selecting landscape mode swaps the horizontal and vertical size/resolution values reported by GetDeviceCaps.
* Color plotters might have more extensive property sheets, allowing configuration of pen colors and paper type.

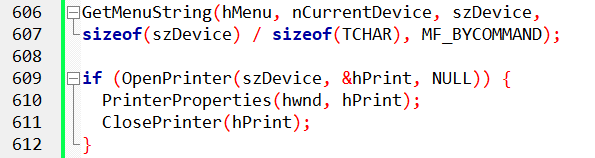
Configuration Persistence:

* Some printer drivers store user-configured settings in their own Registry section.
* This allows access to saved settings during subsequent Windows sessions.
* Programs like DEVCAPS2 can only access settings saved by the driver.

Program Interaction:

* Most programs use the PrintDlg function for printer selection and property configuration.
* This function handles user interaction and settings changes seamlessly.
* While directly calling the driver's ExtDeviceMode or ExtDeveModePropSheet functions is possible, it's not recommended.
* DEVCAPS2 demonstrates the preferred approach using PrinterProperties:
* This function requires a printer object handle obtained through OpenPrinter.
* Upon displaying the properties dialog, DEVCAPS2 uses ClosePrinter to release the handle.

Code Breakdown:



* Retrieves the selected printer name from the menu and stores it in szDevice.
* Opens the printer using OpenPrinter and assigns the handle to hPrint.
* If opening succeeds, displays the property sheet with PrinterProperties.
* Finally, closes the printer handle using ClosePrinter.

CHECKING FOR BITBLT CAPABILITY: A DEEP DIVE

While GetDeviceCaps provides valuable information about the printer's printable area, resolution, and other capabilities, GDI often handles limitations by simulating functionalities. However, one crucial capability applications should explicitly check is bit-block transfer.

BitBlt Capability and Its Impact:

Bit-block transfer capability is determined by the RC\_BITBLT bit in the RASTERCAPS value returned by GetDeviceCaps.

Most printers (dot-matrix, laser, ink-jet) support bit-block transfers, but plotters do not.

Devices lacking this capability cannot utilize these GDI functions:

Bitmap Creation and Manipulation:

* CreateCompatibleDC
* CreateCompatibleBitmap

Pattern and Bit-Block Operations:

* PatBlt
* BitBlt
* StretchBlt

Text and Icon Drawing:

* GrayString
* DrawIcon

Pixel Access:

* SetPixel
* GetPixel

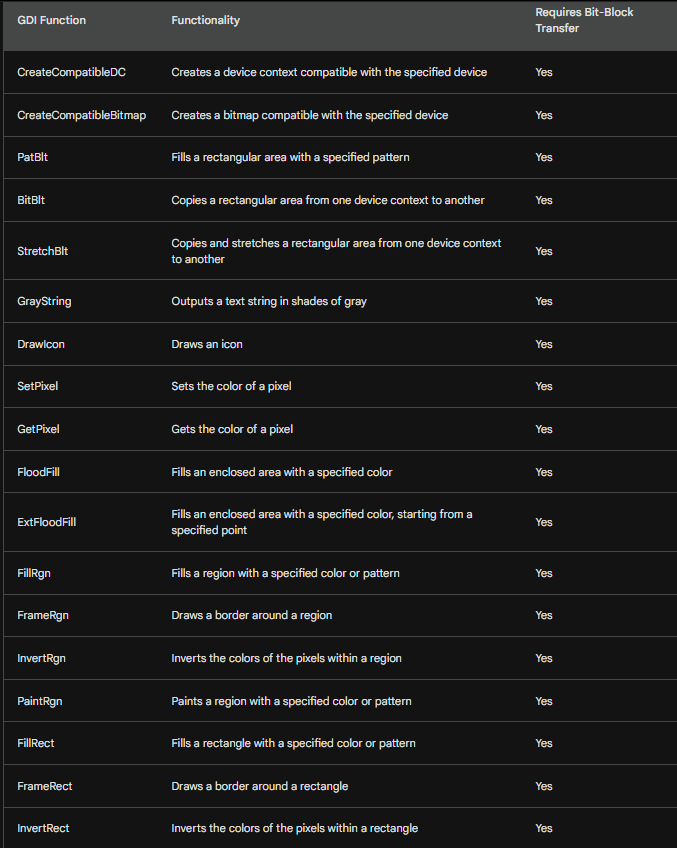
Region Filling and Operations:

* FloodFill
* ExtFloodFill
* FillRgn
* FrameRgn
* InvertRgn
* PaintRgn

Rectangle Operations:

* FillRect
* FrameRect
* InvertRect

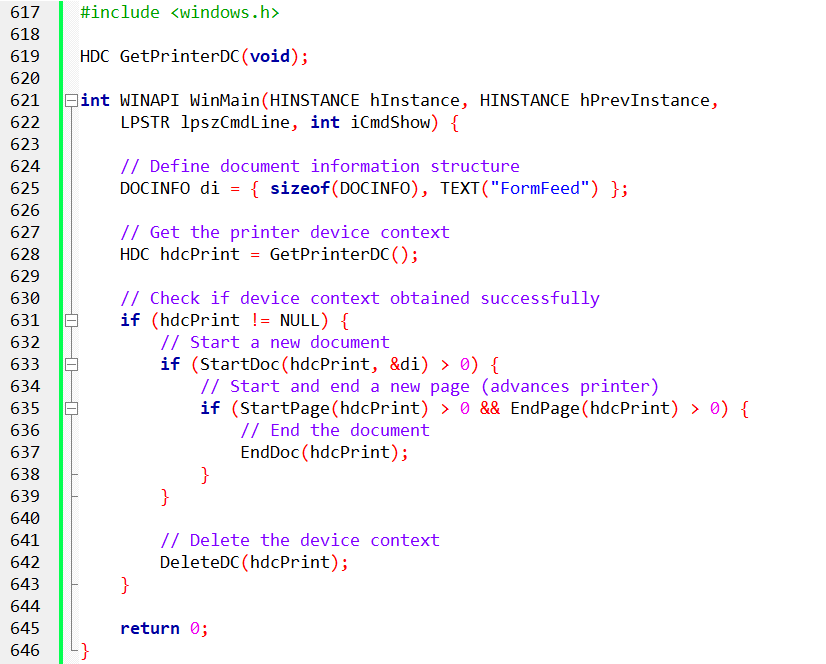
Table Summary:



The Simplest Printing Program: FORMFEED.C

This program demonstrates the minimum requirements for printing by simply causing a printer form feed.

Code Breakdown:



Key Points:

* GetPrinterDC: This function retrieves the printer's device context, allowing the program to interact with the printer.
* StartDoc: This function starts a new printing document and takes the DOCINFO structure as an argument.
* The DOCINFO structure includes the size of the structure and a descriptive text string ("FormFeed" in this case).
* This string appears in the printer's job queue during printing.
* StartPage/EndPage: These functions respectively start and end a printing page. Calling them consecutively advances the printer to a new page.
* Error Checking: The program checks the return values of each function to ensure successful execution.
* A positive return value indicates success, while an error code signifies an issue.
* GDI automatically aborts the document if an error occurs.
* To report specific errors to the user, call GetLastError.

Alternative Approaches:

* While tempting, directly accessing the printer port and writing ASCII code 12 for form feed is discouraged.
* Determining the appropriate port and ensuring exclusivity is complex.
* Different printers interpret form feed commands differently.
* Windows provides dedicated printing functions for reliable and robust operation.

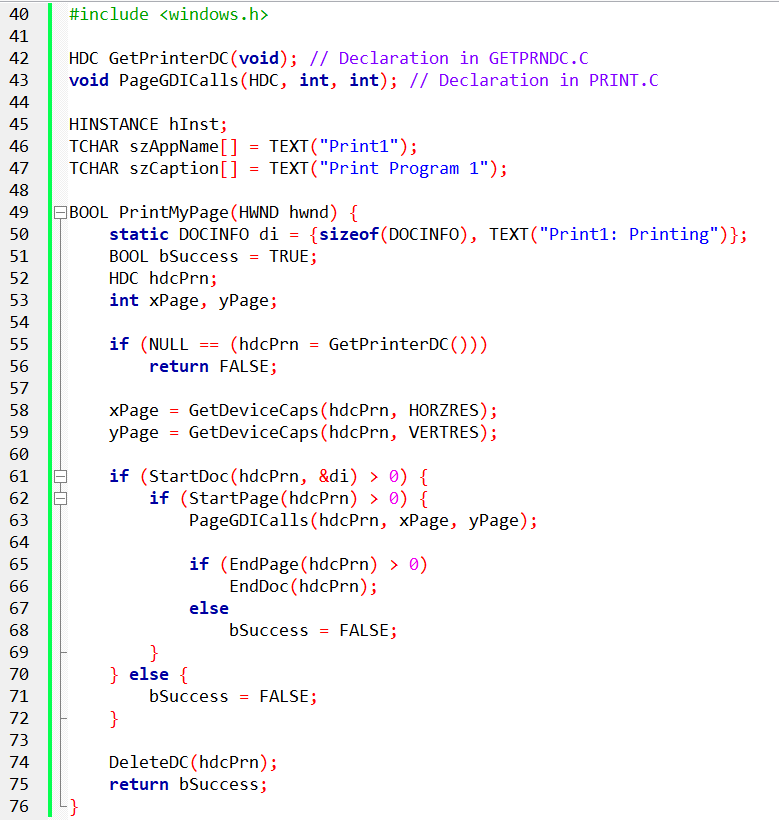
Overall:

FORMFEED.C demonstrates the fundamental principles of printing with Windows API functions. It provides a basic framework for understanding how to start documents, pages, and handle errors. This foundation can be used to build more complex printing applications.

PRINT 1-BareBones printing

The *Print.c program* is a Windows application that demonstrates basic printing functionality. It consists of three versions: PRINT1, PRINT2, and PRINT3. Each version is designed to print one page of text and graphics. The code is structured in a way that allows for reuse of common routines in the PRINT.C file, and it utilizes the GETPRNDC.C file to obtain the printer device context.

Let's break down the key components and functionalities of the program:



WndProc Function:

* The WndProc function is the window procedure that handles messages for the main window.
* During the WM\_CREATE message, the program modifies the system menu to include a "Print" option.
* The WM\_SIZE message updates the client area dimensions when the window is resized.
* The WM\_SYSCOMMAND message triggers printing when the "Print" option is selected from the system menu.
* The WM\_PAINT message initiates the drawing of graphics on the window using the PageGDICalls function.
* The WM\_DESTROY message handles the termination of the program.

PageGDICalls Function:

* The PageGDICalls function is responsible for drawing graphics on the printer device context.
* It draws a rectangle around the entire page, two lines between opposite corners, an ellipse in the middle, and the text "Hello, Printer!" centered on the ellipse.
* Graphics operations like Rectangle, MoveToEx, LineTo, SaveDC, SetMapMode, SetWindowExtEx, SetViewportExtEx, SetViewportOrgEx, Ellipse, SetTextAlign, TextOut, and RestoreDC are used.

PrintMyPage Function:

* The PrintMyPage function is called when the user selects the "Print" option from the system menu.
* It initiates the printing process and returns TRUE if the page is printed successfully, or FALSE if an error occurs.
* In the provided code, PrintMyPage simply calls PageGDICalls to draw graphics on the printer device context. In more advanced versions, this function could handle more complex printing tasks.

WinMain Function:

* The WinMain function is the entry point of the program.
* It registers the window class, creates the main window, and enters the message loop.

The Print1 program unveils its printing power through a well-orchestrated dance of functions and variables. Let's delve into its steps:

1. Setting the Stage:

Header Lineup: Essential Windows header files like windows.h are enlisted to provide the tools for printer interaction.

Function Friends: GetPrinterDC and PageGDICalls are declared, hinting at their crucial roles in the printing process. These functions reside in other modules, waiting for their turn to shine.

Global Introductions: The program introduces itself through hInst, a handle to its unique identity. szAppName proudly announces it as "Print1", while szCaption declares its purpose with "Print Program 1".

2. PrintMyPage: The Master of Ceremonies:

This function takes center stage, orchestrating the entire printing performance.

* Document Identity: A DOCINFO structure is declared, holding information like the document's name ("Print1: Printing"), ready to be shared with the printer.
* Success by Default: Optimistically, bSuccess is set to TRUE, hoping for a flawless printing experience.
* Meeting the Printer: A crucial handshake happens with GetPrinterDC, acquiring a handle to the printer's device context (DC). If this handshake fails, the printing dream fades.
* Sizing Up the Stage: The printer's page dimensions are measured using GetDeviceCaps, revealing the canvas for the upcoming artwork.
* Opening the Curtain: StartDoc initiates the printing process, introducing the DOCINFO to the printer. A negative response here signals an error, dimming the lights on the performance.
* A Fresh Page: With StartPage, a blank canvas is prepared for the magic to unfold. Again, a negative response brings the curtain down.
* The Act Begins: PageGDICalls steps into the spotlight, wielding the printer DC, page width, and page height like paintbrushes. This mysterious function, defined elsewhere, is where the actual content is drawn on the page.
* Wrapping Up the Act: EndPage signals the completion of the on-page masterpiece. If it fails, bSuccess is sadly lowered.
* The Grand Finale: If StartDoc succeeded, EndDoc gracefully closes the document, ensuring a complete performance.
* Releasing the Stage: Finally, the printer DC is released with DeleteDC, allowing others to use it.

3. PageGDICalls: The Hidden Artist:

While shrouded in mystery, this function receives the baton from PrintMyPage, armed with the printer DC, page dimensions, and its own artistic vision. It uses these tools to paint the desired content onto the page, bringing the document to life.

The Final Curtain:

With these steps, Print1 paints a picture on the chosen paper, proving its mastery of the printing art. Remember, this breakdown focuses on the orchestration of the process, leaving the specific artistic details of PageGDICalls to its own private performance.

Overall, the program showcases basic Windows graphics programming and introduces printing capabilities. The structure allows for incremental enhancements in the subsequent versions (PRINT2 and PRINT3) to introduce more advanced printing features.

DEEP DIVE INTO PRINT1 PRINTING AND ABORT PROCEDURE

Obtaining Device Context and Page Dimensions:

If PrintMyPage fails to get a printer device context (DC) handle, WndProc displays an error message.

If successful, it retrieves the page size (horizontal and vertical pixels) using GetDeviceCaps on HORZRES and VERTRES.

This value represents the printable area, not the full paper size.

PRINT1 Printing Logic:

Structurally similar to FORMFEED, but PageGDICalls is called between StartPage and EndPage.

EndDoc is only called if all previous calls (StartDoc, StartPage, EndPage) are successful.

Canceling Printing with Abort Procedure:

Large documents need a way to cancel printing during the process.

An "abort procedure" allows the program to control printing progress.

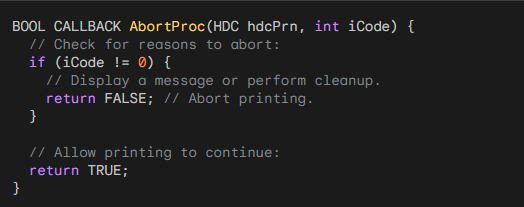
This function, named AbortProc, takes two arguments:

* hdcPrn: The printer device context handle.
* iCode: Indicates the reason for the call (0 for success, SP\_OUTOFDISK for disk space issues).

Implementing Abort Procedure:

Register the procedure: Call SetAbortProc(hdcPrn, AbortProc) before StartDoc.

Define AbortProc:



GDI Calls AbortProc:

During processing of EndPage, GDI calls AbortProc repeatedly.

This allows the program to check and potentially abort the print job.

Keeping the Message Loop Going:

The provided example uses PeekMessage instead of GetMessage.

* PeekMessage retrieves messages without blocking, allowing user input to interrupt printing.
* The loop continues until no more messages are available, then returns control to Windows.

Code Examples:

