GDI Overview

Ron Gery
Microsoft Developer Network Technology Group

Created: March 20, 1992

Abstract

This article provides a brief overview of the design and theory of the graphical component of the Microsoft® Windows™ graphical environment. No details are covered.

Overview

The graphical component of the Microsoft® Windows™ graphical environment is the graphics device interface (GDI). It communicates between the application and the device driver, which performs the hardware-specific functions that generate output. By acting as a buffer between applications and output devices, GDI presents a device-independent view of the world for the application while interacting in a device-dependent format with the device.

In the GDI environment are two working spaces—the logical and the physical. Logical space is inhabited by applications; it is the "ideal" world in which all colors are available, all fonts scale, and output resolution is phenomenal. Physical space, on the other hand, is the real world of devices, with limited color, strange output formats, and differing drawing capabilities. In Windows, an application does not need to understand the quirkiness of a new device. GDI code works on the new device if the device has a device driver.

GDI concepts mapped between the logical and the physical are objects (pens, brushes, fonts, palettes, and bitmaps), output primitives, and coordinates.

Objects are converted from logical objects to physical objects using the realization process. For example, an application creates a logical pen by calling **CreatePen** with the appropriate parameters. When the logical pen object is selected into a device context (DC) using **SelectObject**, GDI realizes the pen into a physical pen object that is used to communicate with the device. GDI passes the logical object to the device, and the device creates a device-specific object containing device-specific information. During realization, requested (logical) color is mapped to available colors, fonts are matched to the best available fonts, and patterns are prepared for output. Font selection is more complex than other realizations, and GDI, not the driver, performs most of the realization work. Similarly, palette realization (done at **RealizePalette** time as opposed to **SelectPalette** time) is done entirely within GDI. Bitmaps are an exception to the object realization process; although they have the device-independent bitmap (DIB) logical form, bitmap objects are always device specific and are never actually realized.

Output primitives are similarly passed as "logical" requests (to stretch the definition) to the device driver, which draws the primitive to the best of its ability and resolution. If the driver cannot handle a certain primitive—for example, it cannot draw an ellipse—GDI simulates the operation. For an **Ellipse** call, GDI calculates a polygon that represents a digitized ellipse. The resulting polygon can then be simulated as a polyline and a series of scanline fills if the device cannot draw polygons itself. The application, though, does not care what system component does the actual work; the primitive gets drawn.

An application can set up for itself any logical coordinate system, using **SetMapMode**, **SetWindowExt**, **SetWindowOrg**, **SetViewportExt**, and **SetViewportOrg**. In GDI that coordinate system is mapped to the device coordinate system, in which one unit equals one pixel and (0,0) defines the topmost, leftmost pixel on the output surface. The device driver sees only coordinates in its own space, whereas the application operates only in a coordinate space of its own, disregarding the physical pixel layout of the destination.

By maintaining the two separate but linked spaces, logical for the applications and physical for the devices, GDI creates a device-independent interface. Applications that make full use of the logical space and avoid device-specific assumptions can expect to operate successfully on any output device that comes down the turnpike.