9

Version 1.1

QUICKDRAW PRELIMINARIES

Includes Demonstration Program GDevicePascal

QuickDraw and Imaging

QuickDraw is a collection of system software routines that your application uses to perform most **imaging** operations on Macintosh computers. Imaging entails the construction and display of graphical information, including shapes, pictures, and text, which can be displayed on such output devices as screens and printers.

Versions of QuickDraw

As the system software has developed, QuickDraw has progressed through the following three main evolutionary stages:

- Basic QuickDraw, which was designed for the early black-and-white Macintoshes. System 7 added new capabilities to basic QuickDraw, including support for offscreen graphics worlds and the extended version 2 picture format. Despite having been designed for the early black-and-white Macintoshes, basic QuickDraw has always supported eight pre-defined colours.[1]

- The **original version of Color QuickDraw**, which was introduced with the first Macintosh II systems, and which could support up to 256 colours.

- The **current version of Color QuickDraw**, which was originally introduced as **32-bit Color QuickDraw,** and which is now part of System 7. This version has been expanded to support millions of colours.

Applications which use only basic QuickDraw routines are compatible with all Macintosh systems. However, applications which use routines specific to Color QuickDraw cannot run on computers supporting only basic QuickDraw.

Graphics Ports

Your application performs all graphics operations in a **graphics port**. A graphics port is a drawing environment which contains the information QuickDraw needs to transmit drawing operations from bits in memory to onscreen pixels. There are two types of graphics port:

- A **basic graphics port,** which is the drawing environment provided by basic QuickDraw. A basic graphics port contains the information basic QuickDraw uses to create and manipulate

[1]Basic QuickDraw is still used on black-and-white Macintoshes such as the Classic.

onscreen black-and-white images, or colour images that employ basic QuickDraw's eight-colour system. A basic graphics port is defined in a `GrafPort` record.

- A **colour graphics port**, which is the sophisticated colour drawing environment provided by Color QuickDraw. A colour graphics port is defined in a `CGrafPort` record.

A graphics port:

- Specifies the bitmap or pixel map (see below) that points to the area of memory in which your drawing operations take place.
- Contains a metaphorical graphics **pen** with which to perform drawing operations. (You can set this pen to different sizes, patterns and colours.)
- Holds information about text, which is styled and sized according to information in the graphics port.

The fields of a graphics port are maintained by QuickDraw. QuickDraw provides routines for changing and reading those fields. For example, routines are available to reshape and resize the pen, change the pen's pattern and colour, switch fonts, point to an image in another part of memory, etc.

Bitmaps and PixelMaps

The visible image in a graphics port is contained in one or other of the following:

- A **bitmap**, which is defined in a data structure of type `BitMap` and which represents the positions and states of a corresponding set of pixels. The pixels can be either black and white or one of the eight basic colours provided by basic QuickDraw. A bitmap is contained within a basic graphics port.
- A **pixel map**, which is defined by a data structure of type `PixMap` and which represents the positions and states of a corresponding set of colour pixels. A handle to a pixel map is contained within a colour graphics port.

Printing Graphics Port

Your application can print the images it prepares in graphics ports by drawing into a **printing graphics port** using QuickDraw drawing routines. A printing graphics port is the printing environment defined by a `TPrPort` record[2].

Offscreen Graphics Worlds

While your application can draw directly into basic and colour graphics ports, you can often improve aspects of your application's appearance and performance by constructing images in **offscreen graphics worlds** and then copying them to onscreen graphics ports. An offscreen graphics world is defined in a private data structure referred to by a pointer of type `GWorldPtr` and contains a graphics port of its own[3].

Colour

The earliest Macintosh models all used basic QuickDraw to draw to built-in screens with known characteristics. The Macintosh II introduced Color QuickDraw, which supports a variety of screens of differing sizes and colour capabilities.

[2]See Chapter 13 — Printing.
[3]See Chapter 12 — Offscreen Graphics Worlds, Pictures, Cursors, and Icons.

Pixels and Colour

A **pixel** (picture element) is the smallest dot that QuickDraw can draw. On a black-and-white monitor, one pixel is a single-colour phosphor dot that displays in one of two states — black or white. On a colour screen, three phosphor dots (red, green and blue) comprise each colour pixel.

Foreground and Background Colour

A pair of fields in the graphics port specify a foreground colour and background colour. The **foreground colour** (which, by default, is black) is the colour used for bit patterns and for the graphics pen when drawing. The **background colour** (which, by default is white) is the colour of the pixels in the bitmap or pixel map where no drawing has taken place.

Basic QuickDraw's Eight-Colour System

Because basic QuickDraw supports an eight-colour system, you can draw in colour on a colour screen even when you are using a basic graphics port. Because Color QuickDraw also supports this eight-colour system, it is compatible across all Macintosh platforms. Basic QuickDraw defines the following constants for those standard colours:

Constant	Value
`blackColor`	33
`whiteColor`	30
`redColor`	205
`greenColor`	341
`blueColor`	409
`cyanColor`	273
`magentaColor`	137
`yellowColor`	69

These are the only colours available in basic QuickDraw (or with Color QuickDraw drawing into a basic graphics port). When you specify these colours on a Macintosh computer with Color QuickDraw, Color QuickDraw draws the colours if the user has set the screen (using the Monitors control panel) to colour mode.

It is important to note that, while these colours are recorded when drawing into a **QuickDraw picture**[4], they cannot be stored in a bitmap.

Advantages and Disadvantages

The advantages of using basic QuickDraw's colour system are that it is available across all platforms and is much simpler to use than Color QuickDraw. The main disadvantage is the eight-colours limit. Another problem is that, if the graphics port you are working in happens to be a colour graphics port, the two colour systems may clash.

Color QuickDraw Routines Available to Basic QuickDraw

In System 7, the Color QuickDraw routines `RGBForeColor` (set the foreground colour), `RGBBackColor`, (set the background colour) `GetForeColor`, (get the foreground colour) and `GetBackColor` (get the background colour) are available to basic QuickDraw. These routines can assist you in manipulating the eight-colour system of basic QuickDraw. Without using a colour graphics port, you can use both `ForeColor` and Color QuickDraw's `RGBForeColor` to set the drawing colour, and either `BackColor` or Color QuickDraw's `RGBBackColor` to set the background colour, on a colour screen.

Colours in Color QuickDraw

In Color QuickDraw, a colour pixel represents up to 48 bits in memory.

[4] See Chapter 12 — Offscreen Graphics Worlds, Pictures, Cursors, and Icons.

Device-Independent Colour

Color QuickDraw is device-independent. Accordingly, you do not have to concern yourself with the capabilities of individual screens. For example, if your application uses an `RGBColor` record to specify a colour by its red, green and blue components, with each component defined in a 16-bit integer, Color QuickDraw compares the resulting 48-bit value with the colours actually available on a video device (such as a plug-in video card or a built-in video interface) at execution time and then chooses the closest match. What the user finally sees depends on the characteristics of the actual video device and screen.

Influence of the Video Device

Screens may display colour or black-and-white; the video device that controls them may have either:

- **Indexed colours**, which support pixels of 1-bit, 2-bit, 4-bit, or 8-bit depths.[5]
- **Direct colours**, which support pixels of 16-bit or 32-bit depths.

Color QuickDraw automatically determines which method is used by the video device and matches your requested colour with the closest available colour.

About Indexed Colour and Direct Colour

Indexed Colours

Video devices using indexed colours support a maximum of 256 colours at any one time, that is, with indexed colour, the maximum value of a pixel in a `PixMap` record is limited to a single byte, with each pixel's byte specifying one of 256 (2^8) different values.

Video devices implementing indexed colour contain a data structure called a **colour lookup table** (or, more commonly, a **CLUT**). The CLUT, in turn, contains entries for all possible colour values.

256 colours is, for many images, sufficient for near-photographic quality. The problem is that the colours needed for one photographic image may not be appropriate for another. Because most indexed video devices use a **variable CLUT**, however, you can display one image using one set of 256 colours and then use system software to reload the CLUT with a second set of 256 colours that are appropriate for the next image.[6] If your application needs this sort of control on indexed video devices, you can use the Palette Manager to arrange palettes (that is, sets of colours) for particular images and for video devices with differing colour capabilities.

If your application uses a 48-bit `RGBColor` record to specify a colour, the Color Manager examines the colours available in the CLUT on the video device. Comparing the CLUT entries to the `RGBColor` record you specify, the Color Manager determines which colour in the CLUT is closest, and gives Color QuickDraw the index to this colour. Color QuickDraw then draws with this colour.

Fig 1 illustrates this process. In Fig 1, the user selects a colour for some object in an application (1). Using a 48-bit `RGBColor` record to specify the colour, the application calls a Color QuickDraw routine to draw the object in that colour (2).

Color QuickDraw uses the Color Manager to determine what colour in the video devices's CLUT comes closest to the requested colour (3). At startup, the video device's declaration ROM supplies information for the creation of the `GDevice` record that describes the characteristics of the device. The resulting record contains a `ColorTable` record that is kept synchronised with the card's CLUT. The Color Manager examines the `GDevice` record to find what colours are currently available (4) and to decide which colour comes closest to the one requested by the application. The Color Manager gets the index

[5]The indexed colour system was introduced with the Macintosh II, i.e., at a time when memory was scarce and moving megabyte images around was impractical.
[6]Some Macintosh computers, such as grayscale PowerBook computers, have a fixed CLUT, which your application cannot change.

value for the best match and returns the value to Color QuickDraw (5), which puts the index value into those places in video RAM which store the object.

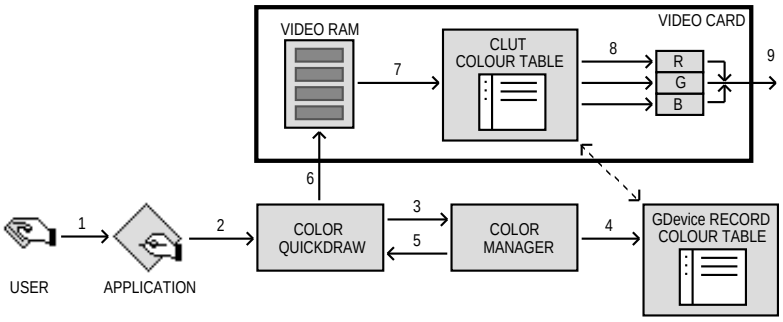

FIG 1 - INDEXED COLOUR SYSTEM

The video device continually displays video RAM by taking the index values, converting them to colours according to CLUT entries at those indexes (7), and sending them to the digital-to-analog converters (8) which produce a signal for the screen (9).

Direct Colours

Video devices which implement direct colour eliminate the competition for limited table spaces and remove the need for colour table matching. By using direct colour, video devices can support thousands or millions of colours.

When you specify a colour using a 48-bit `RGBColor` record on a direct colour system, Color QuickDraw truncates the least significant bits of its red, green and blue components to either 16 bits (five bits each for red, green and blue, with one bit unused) or 32 bits (eight bits for red, green and blue, with eight bits unused). Using 16 bits, direct video devices can display 32,768 colours. Using 32 bits, the device can display 16,777,215 different colours.

Fig 2 illustrates the direct colour system. A user chooses a colour for some object (1) and, using a 48-bit `RGBColor` record to specify the colour, the application uses a Color QuickDraw routine to draw the object in that colour (2). Color QuickDraw knows from the `GDevice` record (3) that the screen is controlled by a direct device in which pixels are 32 bits deep, which means that eight bits are used for each of the red, green and blue components of the requested colour. Colour QuickDraw passes the high eight bits from each 16-bit component of the 48-bit `RGBColor` record to the video device (4), which stores the resulting 24-bit value in video RAM for the object. The video device continually displays video RAM by sending the 8-bit red, green and blue values for the colour to digital-to-analog converters (5) which produce a signal for the screen (6).

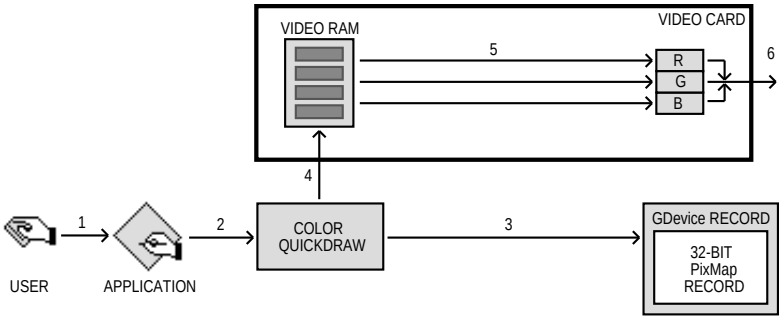

FIG 2 - DIRECT COLOUR SYSTEM

Direct colour not only removes much of the complexity of the CLUT mechanism for video device developers, but it also allows the display of thousands or millions of colours simultaneously, resulting in near-photographic resolution.

Graphics Devices and `GDevice` Records

Graphics Devices

A **graphics device** is anything into which QuickDraw can draw. There are three types of graphics device:

- **Video devices**, such as video cards and built-in video interfaces, that control screens.
- **Offscreen graphics worlds**, which allow your application to build complex images offscreen before displaying them.[7]
- **Printing graphics ports** for printers.[8]

`GDevice` Record

For a video device or an offscreen graphics world, Color QuickDraw automatically creates, and stores state information in, a `GDevice` **record**. Note that printers do not have `GDevice` records. Note also that basic QuickDraw, unlike Color QuickDraw, does not create `GDevice` records.

When the system starts up, QuickDraw uses information supplied by the Slot Manager to create and initialise a `GDevice` record for each video device found during startup. When you use the `NewGWorld` function to create an offscreen graphics world, Color QuickDraw automatically creates a `GDevice` record.

All existing `GDevice` records are linked together in a list called a **device list**. The global variable `DeviceList` holds a handle to the first record in the list. At any given time, exactly one graphics device is the current device (called the **active device**), that is, the one in which drawing is actually taking place. A handle to its `GDevice` record is stored in the global variable `TheGDevice`. By default, the `GDevice` record corresponding to the first video device found is marked as the current device.

The `GDevice` record is as follows:

```
type
GDevice = record
  gdRefNum:      integer;      {Reference Number of Driver.}
  gdID:          integer;      {Client ID for search procedures.}
  gdType:        integer;      {Type of device (indexed or direct).}
  gdITable:      ITabHandle;   {Handle to inverse lookup table for Color Manager.}
  gdResPref:     integer;      {Preferred resolution of GDITable.}
  gdSearchProc:  SProcHndl;    {Handle to list of search procedures.}
  gdCompProc:    CProcHndl;    {Handle to list of complement functions.}
  gdFlags:       integer;      {Graphics device flags.}
  gdPMap:        PixMapHandle; {Handle to pixel map for displayed image.}
  gdRefCon:      longint;      {Reference value.}
  gdNextGD:      Handle;       {Handle to next GDevice record.}
  gdRect:        Rect;         {Device's bounds in global coordinates.}
  gdMode:        longint;      {Device's current mode.}
  gdCCBytes:     integer;      {Depth of expanded cursor data.}
  gdCCDepth:     integer;      {Depth of expanded cursor data.}
  gdCCXData:     Handle;       {Handle to cursor's expanded data.}
  gdCCXMask:     Handle;       {Handle to cursor's expanded mask.}
  gdReserved:    longint;      {Reserved for future use. Must be 0.}
  end;

GDPtr = ^GDevice;
GDHandle = ^GDPtr;
```

[7] See Chapter 12 — Offscreen Graphics Worlds, Pictures, Cursors, and Icons.
[8] See Chapter 13 — Printing.

Field Descriptions

Descriptions of some key fields in the `GDevice` record are as follows:

`gdType` The general type of device, that is, a fixed CLUT device, a variable CLUT device, or a direct device.

`gdITable` Points to an **inverse table**. An inverse table is a special Color Manager data structure arranged in such a manner that, given an arbitrary RGB colour, its pixel value (that is, its index number in the CLUT) can be found quickly. The process is very fast once the table has been built; however, if a colour is changed in the video device's CLUT, the Color Manager must rebuild the inverse table the next time it has to find a colour.

`gdFlags` Device attributes (that is, whether the device is a screen, whether it is the main screen, whether it is set to black-and-white or colour, whether it is the active device, etc.)

`gdPMap` Contains a handle to the pixel map (`PixMap`) record, which contains the dimensions of the image buffer, along with the characteristics of the graphics device (resolution, storage format, **pixel depth** (see below) and colour table.

Note that Color QuickDraw automatically synchronises the pixel map's colour table (`ColorTable`) record with the CLUT on the video device.

`gdRect` Describes the graphics device's boundary rectangle in global coordinates. Color QuickDraw maps the (0, 0) origin point of the global coordinate plane to the main screen's upper left corner.

Setting a Device's Pixel Depth

As stated in the description of the `gdPMap` field of the `GDevice` record, the pixel map's `PixelSize` field contains the pixel depth of the device.

The Monitors control panel is the user interface for changing the pixel depth and colour capabilities of video devices. Note that, when a user uses the Monitors control panel to set a 16-bit or 32-bit device to use 2, 4, 16 or 256 colours as a grayscale or colour device, the direct device creates a CLUT and operates like an indexed device.

Since the user can control the capabilities of the video device, your application should be flexible, that is, although it may have a preferred pixel depth, it should do its best to accommodate less than ideal conditions. Your application can use the `SetDepth` function to change the pixel depth, but it should not do so without the consent of the user. Before calling `SetDepth`, you should use the `HasDepth` function to determine whether the available hardware can support the pixel depth you require.

Other Graphics Managers

In addition to the QuickDraw routines, several other collections of system software routines are available to assist you in drawing images.

Palette Manager

To provide more sophisticated colour support on indexed graphics devices, your application can use the Palette Manager. The Palette Manager allows your application to specify sets of colours that it needs on a window-by-window basis. On a video device that uses a variable CLUT, your application can use the Palette Manager to display any number of palettes (that is, sets of colours) consisting of 256 colours each. Remember, though, that only one set of colours (palette) can be displayed at any one time.

Color Picker Utilities

To solicit colour choices from users, your application can use the Color Picker Utilities. The Color Picker Utilities also provide routines that allow your application to convert between colours specified in `RGBColor` records and colours specified for other colour models, such as the CMYK (cyan, magenta, yellow, key - usually black) model used for many colour printers. (See Chapter 22— Miscellany.)

Main Constants, Data Types and Routines Relating to Graphics Devices

Constants

Flag Bits of `gdType` Field of `GDevice` Record

```
clutType      = 0   CLUT device, that is, one with colours mapped with colour lookup table.
fixedType     = 1   Fixed colours, that is, colour lookup table cannot be changed.
directType    = 2   Direct RGB colours.
```

Flag Bits of `gdFlags` Field of `GDevice` Record

```
gdDevType     = 0   0 = black-and-white. 1 = color.
burstDevice   = 7   If bit is set to 1, graphics device supports block transfer.
ext32Device   = 8   If bit is set to 1, graphics device must be used in 32-bit mode
ramInit       = 10  If bit is set to 1, graphics device was initialised from RAM.
mainScreen    = 11  If bit is set to 1, graphics device is the main screen.
allInit       = 12  If bit is set to 1, all devices were initialised from 'scrn' resource.
screendevice  = 13  If bit is set to 1, graphics device is a screen device.
noDriver      = 14  If bit is set to 1, GDevice record has no driver.
screenActive  = 15  If bit is set to 1, graphics device is current device.
```

Data Types

```
QDErr = integer;
```

GDevice Record

```
GDevice = record
  gdRefNum:      integer;      {Reference Number of Driver.}
  gdID:          integer;      {Client ID for search procedures.}
  gdType:        integer;      {Type of device (indexed or direct).}
  gdITable:      ITabHandle;   {Handle to inverse lookup table for Color Manager.}
  gdResPref:     integer;      {Preferred resolution of GDITable.}
  gdSearchProc:  SProcHndl;    {Handle to list of search procedures.}
  gdCompProc:    CProcHndl;    {Handle to list of complement functions.}
  gdFlags:       integer;      {Graphics device flags.}
  gdPMap:        PixMapHandle; {Handle to pixel map for displayed image.}
  gdRefCon:      longint;      {Reference value.}
  gdNextGD:      Handle;       {Handle to next GDevice record.}
  gdRect:        Rect;         {Device's bounds in global coordinates.}
  gdMode:        longint;      {Device's current mode.}
  gdCCBytes:     integer;      {Depth of expanded cursor data.}
  gdCCDepth:     integer;      {Depth of expanded cursor data.}
  gdCCXData:     Handle;       {Handle to cursor's expanded data.}
  gdCCXMask:     Handle;       {Handle to cursor's expanded mask.}
  gdReserved:    longint;      {Reserved for future use. Must be 0.}
  end;

GDPtr = ^GDevice;
GDHandle = ^GDPtr;
```

Routines

Creating, Setting and Disposing of Graphics Device Records

```
function   NewGDevice(refNum: integer; mode: longint): GDHandle;
procedure  InitGDevice(qdRefNum: integer; mode: longint; gdh: GDHandle);
procedure  SetDeviceAttribute(gdh: GDHandle; attribute: integer; value: boolean);
procedure  SetGDevice(gd: GDHandle);
```

```
procedure  DisposeGDevice(gdh: GDHandle);
```

Getting the Available Graphics Devices

```
function   GetGDevice: GDHandle;
function   GetDeviceList: GDHandle;
function   LMGetMainDevice: GDHandle;
function   GetNextDevice(curDevice: GDHandle): GDHandle;
function   LMGetDeviceList : GDHandle;
```

Determining the Characteristics of a Video Device

```
function   testdeviceattribute(gdh: GDhandle; attribute: integer): boolean;
procedure  ScreenRes(var scrnHRes: integer; var scrnVRes: integer);
```

Changing the Pixel Depth of a Video Device

```
function   SetDepth(gd: GDHandle; depth: integer; whichFlags: integer; flags: integer): OSErr;
function   HasDepth(gd: GDHandle; depth: integer; whichFlags: integer; flags: integer):
           integer;
```

Demonstration Program

```
{ ############################################################################################
// GDevicePascal.p
// ############################################################################################
//
// This program opens a small window, gets a handle to the GDevice record for the main
// device and displays some information obtained from that record.
//
// The program will run only on Macintoshes with Color QuickDraw.
//
// The program utilises an 'ALRT' resource, a 'DITL' resource, and a 'WIND' resource.
//
// ############################################################################################ }

program GDevicePascal(input,  output);

{ ……… include the following Universal Interfaces }

uses

  Windows, Fonts, Menus, TextEdit, Quickdraw, Dialogs, QuickdrawText, Processes, Types,
  Events, TextUtils, ToolUtils, Devices, GestaltEqu, LowMem, Segload;

{ ……… define the following constants }

const

rAlert = 128;
rWindow = 128;

{ ……… global variables }

var

  theErr, ignored : OSErr;
  response : longint;
  myWindowPtr : WindowPtr;
  deviceHdl : GDHandle;
  deviceType,  bytesPerRow : integer;
  theRect : Rect;
  theString : string;
  pixMapHdl : PixMapHandle;

{ ############################################################ start of main program }

begin

  { ……… initialise managers }

  InitGraf(@qd.thePort);
  InitFonts;
```

```
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs(nil);
    InitCursor;

    { ……… check for Color QuickDraw }

    theErr := Gestalt(gestaltQuickdrawVersion, response);
    if (response < gestalt8BitQD) then
      begin
      ParamText('This program will run only on Macintoshes with Color QuickDraw',
              '', '', '');
      ignored := StopAlert(rAlert, nil);
      ExitToShell;
      end;

    { ……… open a window }

    myWindowPtr := GetNewWindow(128, nil, WindowPtr(-1));
    if (myWindowPtr = nil) then
      ExitToShell;

    SetPort(myWindowPtr);
    TextSize(10);

    { ……… get handle to GDevice record for main device }

    deviceHdl := LMGetMainDevice;

    { ……… print some information from the GDevice record }

    MoveTo(10, 20);
    deviceType := deviceHdl^^.gdType;
    case (deviceType) of

      0:  begin
        DrawString('Indexed device with variable CLUT.');
        end;

      1:  begin
        DrawString('Indexed device with fixed CLUT.');
        end;

      2:  begin
        DrawString('Direct device.');
        end;

      end;
        {of case statement}

    MoveTo(10, 40);
    theRect := deviceHdl^^.gdRect;
    DrawString('Boundary rectangle top = ');;
    NumToString(longint(theRect.top), theString);
    DrawString(theString);

    MoveTo(10, 55);
    DrawString('Boundary rectangle left = ');;
    NumToString(longint(theRect.left), theString);
    DrawString(theString);

    MoveTo(10, 70);
    DrawString('Boundary rectangle bottom = ');;
    NumToString(longint(theRect.bottom), theString);
    DrawString(theString);

    MoveTo(10, 85);
    DrawString('Boundary rectangle right = ');;
    NumToString(longint(theRect.right), theString);
    DrawString(theString);

    MoveTo(10, 105);
    pixMapHdl := deviceHdl^^.gdPMap;
    DrawString('Pixel depth := ');
    NumToString(longint(pixMapHdl^^.pixelSize), theString);
    DrawString(theString);
```

```
128
129     MoveTo(10, 120);
130     bytesPerRow := BAnd(pixMapHdl^^.rowBytes, $7FFF);
131     DrawString('Bytes per row := ');
132     NumToString(longint(bytesPerRow), theString);
133     DrawString(theString);
134
135     MoveTo(10, 135);
136     DrawString('Total pixel image bytes = ');
137     NumToString(longint(bytesPerRow) * theRect.bottom, theString);
138     DrawString(theString);
139
140     MoveTo(10, 155);
141     if (pixMapHdl^^.hRes = $00480000) then
142       DrawString('Resolution = 72 dpi');
143
144     MoveTo(10, 175);
145     if (BitTst(@deviceHdl^^.gdFlags, screenActive))
146       then  DrawString('Device is the current device')
147       else  DrawString('Device is not the current device');
148
149     while not (Button) do ;
150
151   end.
152
153   { ############################################################################################ }
```

Demonstration Program Comments

This program is not a demonstration per se. It simply opens a window and prints some information retrieved from the GDevice record for the main device.

The first item indicates whether the device is a direct device, an indexed device with a fixed CLUT or an indexed device with a variable CLUT. Other items include the boundary rectangle, the pixel depth, the number of bytes per row, the total bytes in the pixel image, the resolution, and whether the device is the current device.

Users with colour monitors should open the Monitors control panel, change the settings in the monitor characteristics section, run the program again, and note the changes to the pixel depth, bytes per row, and total pixel image bytes.

Users with direct devices capable of supporting 32-bit colour (16, 777, 215 colours) and/or 16-bit colour (32, 768 colours) should note particularly that, when the 256 colours setting is selected in the Monitors control panel, the gdType field of the GDevice record reports that the device is an indexed device with a variable CLUT. This happens because, when a 16-bit or 32-bit device is set to use 2, 4, 16 or 256 colours as a grayscale or colour device, the direct device creates a CLUT and operates like an indexed device.

Also note that the subject of coping with a multiple monitors environment is addressed at Chapter 22 — Miscellany.