Pictures

Contents

About Pictures 7-4
Picture Formats 7-5
Opcodes: Drawing Commands and Picture Comments 7-6
Color Pictures in Basic Graphics Ports 7-6
'PICT' Files, 'PICT' Resources, and the 'PICT' Scrap Format 7-7
The Picture Utilities 7-8
Using Pictures 7-8
Creating and Drawing Pictures 7-10
Opening and Drawing Pictures 7-13
Drawing a Picture Stored in a 'PICT' File 7-13
Drawing a Picture Stored in the Scrap 7-17
Defining a Destination Rectangle 7-18
Drawing a Picture Stored in a 'PICT' Resource 7-20
Saving Pictures 7-21
Gathering Picture Information 7-24
Pictures Reference 7-26
Data Structures 7-27
QuickDraw and Picture Utilities Routines 7-36
Creating and Disposing of Pictures 7-36
Drawing Pictures 7-43
Collecting Picture Information 7-46
Application-Defined Routines 7-61
Resources 7-67
The Picture Resource 7-67
The Color-Picking Method Resource 7-68
Summary of Pictures and the Picture Utilities 7-69
Pascal Summary 7-69
Constants 7-69
Data Types 7-69

Routines 7-72
Application-Defined Routines 7-73
C Summary 7-73
Constants 7-73
Data Types 7-74
Functions 7-76
Application-Defined Functions 7-77
Assembly-Language Summary 7-78
Data Structures 7-78
Trap Macros 7-80
Result Codes 7-80

This chapter describes QuickDraw **pictures,** which are sequences of saved drawing commands. Pictures provide a common medium for the sharing of image data. Pictures make it easier for your application to draw complex images defined in other applications; pictures also make it easier for other applications to display images created with your application. Virtually all applications should support the creation and drawing of pictures. All applications that support cut and paste, for example, should be able to draw pictures copied by the user from the Clipboard.

Read this chapter to learn how to record QuickDraw drawing commands into a picture and how to draw the picture later by playing back these commands. You should also read this chapter to learn about the Picture Utilities, which allow your application to gather information about pictures—such as their colors, fonts, picture comments, and resolution. You can also use the Picture Utilities to gather information about the colors in pixel maps. Your application can use this information in conjunction with the Palette Manager, for example, to provide the best selection of colors for displaying a picture or other pixel image on an indexed device.

The `OpenCPicture` function, available on all Macintosh computers running System 7, allows your application to create pictures in the **extended version 2 picture format.** This format allows your application to specify resolutions when creating pictures.

Pictures can be created in color or black and white. Computers supporting only basic QuickDraw use black and white to display pictures created in color.

As described in this chapter, your application can use File Manager or Resource Manager routines to save or open pictures stored in files. See the chapter “File Manager” in *Inside Macintosh: Files* for more information about the File Manager; see the chapter “Resource Manager” in *Inside Macintosh: More Macintosh Toolbox* for more information about the Resource Manager. To store or retrieve pictures in the scrap—for example, when the user copies from or pastes to the Clipboard—you must use Scrap Manager routines. See the chapter “Scrap Manager” in *Inside Macintosh: More Macintosh Toolbox* for more information about the Scrap Manager.

You typically use the information gathered with the Picture Utilities in conjunction with other system software managers. You might use the Picture Utilities to determine what fonts are used in a picture, for example, and then use Font Manager routines to help you determine whether those fonts are available on the user’s system. Or, you might use the Picture Utilities to determine the most-used colors in a picture, and then use the Palette Manager or ColorSync Utilities to provide sophisticated support for these colors. For more information about fonts, see the chapter “Font Manager” in *Inside Macintosh: Text*. The Palette Manager and the ColorSync Utilities are described in *Inside Macintosh: Advanced Color Imaging.*

You can also save and collect picture comments within your picture, as described in this chapter. Typically, however, your application uses picture comments to include special drawing commands for printers. Therefore, picture comments are described in greater detail in Appendix B, “Using Picture Comments for Printing,” in this book.

About Pictures

QuickDraw provides a simple set of routines for recording a collection of its drawing commands and then playing the collection back later. Such a collection of drawing commands (as well as its resulting image) is called a *picture*. A replayed collection of drawing commands results in the picture shown in Figure 7-1.

Figure 7-1 A picture of a party hat

When you use the `OpenCPicture` function (or the `OpenPicture` function) to begin defining a picture, QuickDraw collects your subsequent drawing commands in a data structure of type `Picture`. You can define a picture by using any of the drawing routines described in this book—with the exception of the `CopyMask`, `CopyDeepMask`, `SeedFill`, `SeedCFill`, `CalcMask`, and `CalcCMask` routines.

By using the `DrawPicture` procedure, you can draw onscreen the picture defined by the instructions stored in the `Picture` record. Your application typically does not directly manipulate the information in this record. Instead, using a handle to a `Picture` record, your application passes this information to QuickDraw routines and Picture Utilities routines.

Note

The `OpenPicture` function, which is similar to the `OpenCPicture` function, was created for earlier versions of system software. Because of the support for higher resolutions provided by the `OpenCPicture` function, you should use `OpenCPicture` instead of `OpenPicture` to create a picture. ◆

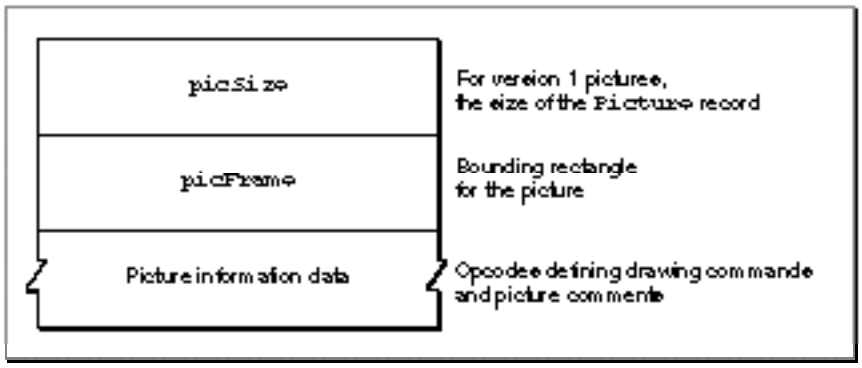
Picture Formats

Through QuickDraw's development, three different formats have evolved for the data contained in a `Picture` record. These formats are

- The extended version 2 format, which is created by the `OpenCPicture` function on all Macintosh computers running System 7, including those supporting only basic QuickDraw. This format permits your application to specify resolutions for pictures in color or black and white. Generally, your application should use the `OpenCPicture` function and create pictures in the extended version 2 format.
- The version 2 picture format, which is created by the `OpenPicture` function on machines with Color QuickDraw when the current graphics port is a color graphics port. Pictures created in this format support color drawing operations at 72 dpi.
- The original format, the version 1 picture format, which is created by the `OpenPicture` function on machines without Color QuickDraw or whenever the current graphics port is a basic graphics port. Pictures created in this format support only black-and-white drawing operations at 72 dpi.

The Pascal data structure for all picture formats is exactly the same. As shown in Figure 7-2, the `Picture` record begins with a `picSize` field and a `picFrame` field, followed by a variable amount of picture definition data.

Figure 7-2 The `Picture` record



To maintain compatibility with the version 1 picture format, the `picSize` field was not changed for the version 2 or extended version 2 picture formats. The information in this field is useful only for version 1 pictures, which cannot exceed 32 KB in size. Version 2 and extended version 2 pictures can be much larger than the 32 KB limit imposed by the 2-byte `picSize` field. You should use the Memory Manager function `GetHandleSize` to determine the size of a picture in memory, the File Manager function `PBGetFInfo` to determine the size of a picture in a file of type `'PICT'`, and the Resource Manager function `MaxSizeResource` to determine the size of a picture in a resource of type `'PICT'`. (See *Inside Macintosh: Memory*, *Inside Macintosh: Files*, and *Inside Macintosh: More Macintosh Toolbox* for more information about these functions.)

The `picFrame` field contains the bounding rectangle for the picture. The `DrawPicture` procedure uses this rectangle to scale the picture when you draw it into a differently sized rectangle.

Compact drawing instructions and picture comments constitute the rest of this record.

Opcodes: Drawing Commands and Picture Comments

Following the `picSize` and `picFrame` fields, a `Picture` record contains data in the form of **opcodes**, which are values that the `DrawPicture` procedure uses to determine what object to draw or what mode to change for subsequent drawing. Your application generally should not read or write these opcodes directly but should instead use the QuickDraw routines described in this chapter for generating and processing the opcodes. (For your application's debugging purposes, these opcodes are listed in Appendix A at the back of this book.)

In addition to compact QuickDraw drawing commands, opcodes can also specify picture comments. Created with the `PicComment` procedure, a **picture comment** contains data or commands for special processing by output devices, such as PostScript printers. If your application requires capability beyond that provided by QuickDraw drawing routines, the `PicComment` procedure allows your application to pass data or commands directly to the output device. For example, picture comments enable highly sophisticated drawing applications that process opcodes directly to reconstruct drawing instructions—such as rotating text—not found in QuickDraw. Picture comments are usually stored in the definition of a picture or are included in the code an application sends to a printer driver.

Unless your application creates highly sophisticated graphics, you typically use QuickDraw commands when drawing to the screen and use picture comments to include special drawing commands for printers only. For example, your application can use picture comments to specify commands—for rotating text and graphics and for drawing dashed lines, fractional line widths, and smoothed polygons—that are supported by some printers but are not accessible through standard QuickDraw calls. These picture comments are described in detail in Appendix B, "Using Picture Comments for Printing," in this book.

Color Pictures in Basic Graphics Ports

You can use Color QuickDraw drawing commands to create a color picture on a computer supporting Color QuickDraw. If the user were to cut the picture and paste it into an application that draws into a basic graphics port, the picture would lose some detail, but should be sufficient for most purposes. This is how basic QuickDraw in System 7 draws an extended version 2 or version 2 picture into a basic graphics port:

- QuickDraw maps foreground and background colors to those most closely approximated in basic QuickDraw's eight-color system.
- QuickDraw draws pixel patterns created with the `MakeRGBPat` procedure as bit patterns having approximately the same luminance as the pixel patterns.

- QuickDraw replaces other color patterns with the bit pattern contained in the `pat1Data` field of the `PixPat` record. (The `pat1Data` field is initialized to 50 percent gray if the pattern is created with the `NewPixPat` function; this field is initialized from a `'ppat'` resource if the pattern is retrieved with the `GetPixPat` function.)
- QuickDraw converts the pixel image to a bit image.
- QuickDraw ignores the values set by the `HiliteColor` and `OpColor` procedures, as well as any changes made to the `chExtra` and `pnLocHFrac` fields of the original `CGrafPort` record.

'PICT' Files, 'PICT' Resources, and the 'PICT' Scrap Format

QuickDraw provides routines for creating and drawing pictures; to read pictures from and to write pictures to disk, you use File Manager and Resource Manager routines. To read pictures from and write pictures to the scrap, you use Scrap Manager routines.

Files consist of two forks: a data fork and a resource fork. A **data fork** is the part of a file that contains data accessed using the File Manager. This data usually corresponds to data entered by the user. A **resource fork** is the part of a file that contains the file's resources, which contain data accessed using the Resource Manager. This data usually corresponds to data—such as menu, icon, and control definitions—created by the application developer, but it may also include data created by the user while the application is running.

A picture can be stored in the data fork of a file of type `'PICT'`. A picture can also be stored as a resource of type `'PICT'` in the resource fork of any file type.

Normally, an application sets the file type in the file's `FInfo` record when the application creates a new file; for example, the File Manager function `FSpCreate` takes a four-character file type—such as `'PICT'`—as a parameter. The data fork of a `'PICT'` file begins with a 512-byte header that applications can use for their own purposes. A `Picture` record follows this header. To read and write `'PICT'` files, your application should use File Manager routines.

You may find it useful to store pictures in the resource fork of your application or document file. For example, in response to the user choosing the About command in the Apple menu for your application, you might wish to display a window containing your company's logo. Or, if yours is a page-layout application, you might want to store all the images created by the user for a document as resources in the document file.

You can use high-level tools like the ResEdit resource editor, available from APDA, to create and store pictures as `'PICT'` resources for distribution with your files. To create `'PICT'` resources while your application is running, you should use Resource Manager routines. To retrieve a picture stored in a `'PICT'` resource, you can use the `GetPicture` function.

For each application, the Scrap Manager maintains a storage area to hold the last data cut or copied by the user. The area that is available to your application for this purpose is called the **scrap.** The scrap can reside either in memory or on disk. All applications that support copy-and-paste operations read data from and write data to the scrap. The

'PICT' scrap format is one of two standard scrap formats. (The other is 'TEXT'.) To support copy-and-paste operations, your application should use Scrap Manager routines to read and write data in 'PICT' scrap format.

The Picture Utilities

In addition to the QuickDraw routines for creating and drawing pictures, system software provides a group of routines called the **Picture Utilities** for examining the contents of pictures. You typically use the Picture Utilities before displaying a picture.

The Picture Utilities allow you to gather color, comment, font, resolution, and additional information about pictures. You might use the Picture Utilities, for example, to determine the 256 most-used colors in a picture, and then use the Palette Manager to make these colors available for the window in which your application needs to draw the picture.

You can also use the Picture Utilities to collect colors from pixel maps. You typically use this information in conjunction with the Palette Manager and the ColorSync Utilities to provide advanced color imaging features for your users. These features are described in *Inside Macintosh: Advanced Color Imaging*.

The Picture Utilities also collect information from black-and-white pictures and bitmaps. The Picture Utilities are supported in System 7 even by computers running only basic QuickDraw. However, when collecting color information on a computer running only basic QuickDraw, the Picture Utilities return NIL instead of handles to Palette and ColorTable records.

Using Pictures

To create a picture, you should

- use the OpenCPicture function to create a Picture record and begin defining the picture
- issue QuickDraw drawing commands, which are collected in the Picture record
- use the PicComment procedure to include picture comments in the picture definition (optional)
- use the ClosePicture procedure to conclude the picture definition

To open an existing picture, you should

- use File Manager routines to get a picture stored in a 'PICT' file
- use the GetPicture function to get a picture stored in a 'PICT' resource
- use the Scrap Manager function GetScrap to get a picture stored in the scrap

To draw a picture, you should use the DrawPicture procedure.

To save a picture, you should

- use File Manager routines to save the picture in a `'PICT'` file
- use Resource Manager routines to save the picture in a `'PICT'` resource
- use the Scrap Manager function `PutScrap` to place the picture in the scrap

To conserve memory, you can spool large pictures to and from disk storage; you should

- write your own low-level procedures—using File Manager routines—that read and write temporary `'PICT'` files to disk
- use the `SetStdCProcs` procedure for a color graphics port (or the `SetStdProcs` procedure for a basic graphics port) and replace QuickDraw's standard low-level procedures `StdGetPic` and `StdPutPic` with your own procedures for reading and writing temporary `'PICT'` files to disk

To gather information about a single picture, pixel map, or bitmap, you should

- use the `GetPictInfo` function to get information about a picture, or use the `GetPixMapInfo` function to get information about a pixel map or bitmap
- use the `Palette` record or the `ColorTable` record, the handles of which are returned by these functions in a `PictInfo` record, to examine the colors collected from the picture, pixel map, or bitmap
- use the `FontSpec` record, the handle of which is returned by `GetPictInfo` in a `PictInfo` record, to examine the fonts contained in the picture
- use the `CommentSpec` record, the handle of which is returned by `GetPictInfo` in a `PictInfo` record, to examine the picture comments contained in the picture
- examine the rest of the fields of the `PictInfo` record for additional information—such as pixel depth or optimal resolution—about the picture, pixel map, or bitmap
- use the Memory Manager procedure `DisposeHandle` to release the memory occupied by the `PictInfo`, `FontSpec`, and `CommentSpec` records; use the Palette Manager procedure `DisposePalette` to release the memory occupied by a `Palette` record; and use the Color QuickDraw procedure `DisposeCTable` to release the memory occupied by a `ColorTable` record when you are finished with the information collected by the `GetPictInfo` function

To gather information about multiple pictures, pixel maps, and bitmaps, you should

- use the `NewPictInfo` function to begin collecting pictures, pixel maps, and bitmaps for your survey
- use the `RecordPictInfo` function to add the information for a picture to your survey
- use the `RecordPixMapInfo` function to add the information for a pixel map or bitmap to your survey
- use the `RetrievePictInfo` function to return the collected information in a `PictInfo` record
- use the `Palette` record or the `ColorTable` record, the handles of which are returned in the `PictInfo` record, to examine the colors collected from the pictures, pixel maps, and bitmaps

- use the `FontSpec` record, the handle of which is returned in the `PictInfo` record, to examine the fonts contained in the collected pictures
- use the `CommentSpec` record, the handle of which is returned in the `PictInfo` record, to examine the picture comments contained in the collected pictures
- examine the rest of the fields of the `PictInfo` record for additional information about the pictures, pixel maps, and bitmaps in your survey
- use the `DisposePictInfo` function to dispose of the private data structures allocated by the `NewPictInfo` function; use the Memory Manager procedure `DisposeHandle` to release the memory occupied by `PictInfo`, `FontSpec`, and `CommentSpec` records; use the Palette Manager procedure `DisposePalette` to release the memory occupied by a `Palette` record; and use the Color QuickDraw procedure `DisposeCTable` to release the memory occupied by a `ColorTable` record when you are finished with the information collected by `NewPictInfo`

When you are finished using a picture (such as when you close the window containing it), you should

- release the memory it occupies by calling the `KillPicture` procedure if the picture is not stored in a `'PICT'` resource
- release the memory it occupies by calling the Resource Manager procedure `ReleaseResource` if the picture is stored in a `'PICT'` resource

Before using the routines described in this chapter, you must use the `InitGraf` procedure, described in the chapter "Basic QuickDraw" in this book, to initialize QuickDraw. The routines in this chapter are available on all computers running System 7—including those supporting only basic QuickDraw. To test for the existence of System 7, use the `Gestalt` function with the `gestaltSystemVersion` selector. Test the low-order word in the `response` parameter; if the value is $0700 or greater, all of the routines in this chapter are supported.

Note

On computers running only basic QuickDraw, the Picture Utilities return `NIL` in place of handles to `Palette` and `ColorTable` records. ◆

Creating and Drawing Pictures

Use the `OpenCPicture` function to begin defining a picture. `OpenCPicture` collects your subsequent QuickDraw drawing commands in a new `Picture` record. To complete the collection of drawing and picture comment commands that define your picture, use the `ClosePicture` procedure.

Note

Operations with the following routines are not recorded in pictures: `CopyMask`, `CopyDeepMask`, `SeedFill`, `SeedCFill`, `CalcMask`, `CalcCMask`, and `PlotCIcon`. ◆

You pass information to `OpenCPicture` in the form of an `OpenCPicParams` record. This record provides a simple mechanism for specifying resolutions when creating images. For example, applications that create pictures from scanned images can specify resolutions higher than 72 dpi for these pictures in `OpenCPicParams` records.

Listing 7-1 shows an application-defined routine, `MyCreateAndDrawPict`, that begins creating a picture by assigning values to the fields of an `OpenCPicParams` record. In this example, the normal screen resolution of 72 dpi is specified as the picture's resolution. You also specify a rectangle for best displaying the picture at this resolution.

Listing 7-1 Creating and drawing a picture

```
FUNCTION MyCreateAndDrawPict(pFrame: Rect): PicHandle;
CONST
   cHRes = $00480000;   {for 72 dpi}
   cVRes = $00480000;   {for 72 dpi}
VAR
   myOpenCPicParams: OpenCPicParams;
   myPic:            PicHandle;
   trianglePoly:     PolyHandle;
BEGIN
   WITH myOpenCPicParams DO BEGIN
      srcRect := pFrame;   {best rectangle for displaying this picture}
      hRes := cHRes;       {horizontal resolution}
      vRes := cVRes;       {vertical resolution}
      version := - 2;      {always set this field to -2}
      reserved1 := 0;      {this field is unused}
      reserved2 := 0;      {this field is unused}
   END;
   myPic := OpenCPicture(myOpenCPicParams);  {start creating the picture}
   ClipRect(pFrame);                      {always set a valid clip region}
   FillRect(pFrame,dkGray);   {create a dark gray rectangle for background}
   FillOval(pFrame,ltGray);   {overlay the rectangle with a light gray oval}
   trianglePoly := OpenPoly;  {start creating a triangle}
   WITH pFrame DO BEGIN
      MoveTo(left,bottom);
      LineTo((right - left) DIV 2,top);
      LineTo(right,bottom);
      LineTo(left,bottom);
   END;
   ClosePoly;                 {finish the triangle}
   PaintPoly(trianglePoly);   {paint the triangle}
   KillPoly(trianglePoly);    {dispose of the memory for the triangle}
   ClosePicture;              {finish the picture}
```

```
   DrawPicture(myPic,pFrame);       {draw the picture}
   IF QDError <> noErr THEN
      ; {likely error is that there is insufficient memory}
   MyCreateAndDrawPict := myPic;
END;
```

After assigning values to the fields of an `OpenCPicParams` record, the `MyCreateAndDrawPict` routine passes this record to the `OpenCPicture` function.

IMPORTANT

Always use the `ClipRect` procedure to specify a clipping region appropriate for your picture before you call `OpenCPicture`. If you do not use `ClipRect` to specify a clipping region, `OpenCPicture` uses the clipping region specified in the current graphics port. If the clipping region is very large (as it is when a graphics port is initialized) and you scale the picture when drawing it, the clipping region can become invalid when `DrawPicture` scales the clipping region—in which case, your picture will not be drawn. On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when you draw it. Setting a clipping region equal to the port rectangle of the current graphics port, as shown in Listing 7-1, always sets a valid clipping region. ▲

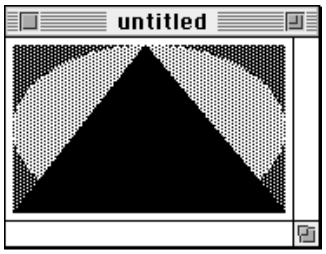
The `MyCreateAndDrawPict` routine uses QuickDraw commands to draw a filled rectangle, a filled oval, and a black triangle. These commands are stored in the `Picture` record.

Note

If there is insufficient memory to draw a picture in Color QuickDraw, the `QDError` function (described in the chapter "Color QuickDraw" in this book) returns the result code `noMemForPictPlaybackErr`. ◆

The `MyCreateAndDrawPict` routine concludes the picture definition by using the `ClosePicture` procedure. By passing to the `DrawPicture` procedure the handle to the newly defined picture, `MyCreateAndDrawPict` replays in the current graphics port the drawing commands stored in the `Picture` record. Figure 7-3 shows the resulting figure.

Figure 7-3 A simple picture



Note

After using `DrawPicture` to draw a picture, your application can use the Window Manager procedure `SetWindowPic` to save a handle to the picture in the window record. When the window's content region must be updated, the Window Manager draws this picture, or only a part of it as necessary, instead of generating an update event. Another Window Manager routine, the `GetWindowPic` function, allows your application to retrieve the picture handle that you store using `SetWindowPic`. When you use the Window Manager procedure `DisposeWindow` to close a window, `DisposeWindow` automatically calls the `KillPicture` procedure to release the memory allocated to a picture referenced in the window record. These routines and the window record are described in the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*. ◆

Opening and Drawing Pictures

Using File Manager routines, your application can retrieve pictures saved in `'PICT'` files; using the `GetPicture` function, your application can retrieve pictures saved in the resource forks of other file types; and using the Scrap Manager function `GetScrap`, your application can retrieve pictures stored in the scrap.

Drawing a Picture Stored in a 'PICT' File

Listing 7-2 illustrates an application-defined routine, called `MyDrawFilePicture`, that uses File Manager routines to retrieve a picture saved in a `'PICT'` file. The `MyDrawFilePicture` routine takes a file reference number as a parameter.

Listing 7-2 Opening and drawing a picture from disk

```
FUNCTION MyDrawFilePicture(pictFileRefNum: Integer; destRect: Rect): OSErr;
CONST
   cPicFileHeaderSize = 512;
VAR
   myPic:      PicHandle;
   dataLength: LongInt;
   err:        OSErr;
BEGIN    {This listing assumes the current graphics port is set.}
   err := GetEOF(pictFileRefNum,dataLength);    {get file length}
   IF err = noErr THEN BEGIN
      err := SetFPos(pictFileRefNum,fsFromStart,
                     cPicFileHeaderSize); {move past the 512-byte 'PICT' }
                                          { file header}
      dataLength := dataLength - cPicFileHeaderSize;  {remove 512-byte }
                                    { 'PICT' file header from file length}
      myPic := PicHandle(NewHandle(dataLength)); {allocate picture handle}
```

```
      IF (err = noErr) & (myPic <> NIL) THEN BEGIN
         HLock(Handle(myPic));   {lock picture handle before using FSRead}
         err := FSRead(pictFileRefNum,dataLength,Ptr(myPic^)); {read file}
         HUnlock(Handle(myPic)); {unlock picture handle after using FSRead}
         MyAdjustDestRect(myPic,destRect);   {see Listing 7-7 on page 7-18}
         DrawPicture(myPic,destRect);
         IF QDError <> noErr THEN
            ; {likely error is that there is insufficient memory}
         KillPicture(myPic);
      END;
   END;
   MyDrawFilePicture := err;
END;
```

In code not shown in Listing 7-2, this application uses the File Manager procedure `StandardGetFile` to display a dialog box that asks the user for the name of a `'PICT'` file; using the file system specification record returned by `StandardGetFile`, the application calls the File Manager function `FSpOpenDF` to return a file reference number for the file. The application then passes this file reference number to `MyDrawFilePicture`.

Because every `'PICT'` file contains a 512-byte header for application-specific use, `MyDrawFilePicture` uses the File Manager function `SetFPos` to skip past this header information. The `MyDrawFilePicture` function then uses the File Manager function `FSRead` to read the file's remaining bytes—those of the `Picture` record—into memory.

The `MyDrawFilePicture` function creates a handle for the buffer into which the `Picture` record is read. Passing this handle to the `DrawPicture` procedure, `MyDrawFilePicture` is able to replay onscreen the commands stored in the `Picture` record.

For large `'PICT'` files, it is useful to spool the picture data from disk as necessary instead of reading all of it directly into memory. In low-memory conditions, for example, your application might find it useful to create a temporary file on disk for storing drawing instructions; your application can read this information as necessary. The application-defined routine `MyReplaceGetPic` shown in Listing 7-3 replaces the `getPicProc` field of the current graphics port's `CQDProcs` record with an application-defined low-level routine, called `MyFileGetPic`. While QuickDraw's standard `StdGetPic` procedure reads picture data from memory, `MyFileGetPic` reads the picture data from disk. (Listing 7-10 on page 7-22 shows how to replace QuickDraw's standard `StdPutPic` procedure with one that writes data to a file so that your application can spool a large picture to disk.)

Listing 7-3 Replacing QuickDraw's standard low-level picture-reading routine

```
FUNCTION MyReplaceGetPic: QDProcsPtr;
VAR
   currPort:      GrafPtr;
   customProcs:   QDProcs;
   customCProcs:  CQDProcs;
   savedProcs:    QDProcsPtr;
BEGIN
   GetPort(currPort);
   savedProcs := currPort^.grafProcs;  {save current CQDProcs }
                                       { or QDProcs record}
   IF MyIsColorPort(currPort) THEN     {this is a color graphics port}
   BEGIN
      SetStdCProcs(customCProcs);   {create new CQDProcs record containing }
                                    { standard Color QuickDraw low-level }
                                    { routines}
      customCProcs.getPicProc := @MyFileGetPic; {replace StdGetPic with }
                                                { address of custom }
                                                { low-level routine }
                                                { shown in Listing 7-5}
      currPort^.grafProcs := @customCProcs;  {replace current CQDProcs }
                                             { record}
   END
   ELSE
   BEGIN                            {this is a basic graphics port}
      SetStdProcs(customProcs);     {create new QDProcs record containing }
                                    { standard basic QuickDraw low-level }
                                    { routines}
      customProcs.getPicProc := @MyFileGetPic;  {replace StdGetPic with }
                                                { address of custom }
                                                { low-level routine }
                                                { shown in Listing 7-5}
      currPort^.grafProcs := @customProcs;   {replace current QDProcs record}
   END;
   MyReplaceGetPic := savedProcs;
END;
```

Listing 7-4 shows the application-defined procedure `MyIsColorPort`, which `MyReplaceGetPic` calls to determine whether to replace the low-level picture-reading routine for a color graphics port or a basic graphics port.

Listing 7-4 Determining whether a graphics port is color or basic

```
FUNCTION MyIsColorPort(aPort: GrafPtr): Boolean;
BEGIN
   MyIsColorPort := (aPort^.portBits.rowBytes < 0)
END;
```

Listing 7-5 shows the application-defined procedure `MyFileGetPic`, which uses the File Manager function `FSRead` to read the file with the file reference number assigned to the application-defined global variable `gPictFileRefNum`.

Listing 7-5 A custom low-level procedure for spooling a picture from disk

```
PROCEDURE MyFileGetPic (dataPtr: Ptr; byteCount: Integer);
VAR
   longCount:  LongInt;
   myErr:      OSErr;
BEGIN
   longCount := byteCount;
   myErr := FSRead(gPictFileRefNum, longCount, dataPtr);
END;
```

Your application does not keep track of where `FSRead` stops or resumes reading a file. After reading a portion of a file, `FSRead` automatically handles where to begin reading next. See *Inside Macintosh: Files* for more information about using `FSRead` and other File Manager routines to retrieve data stored in files.

Drawing a Picture Stored in the Scrap

As described in the chapter "Scrap Manager" in *Inside Macintosh: More Macintosh Toolbox*, your application can use the Scrap Manager to copy and paste data within a document created by your application, among different documents created by your application, and among documents created by your application and documents created by other applications. The two standard scrap formats that all Macintosh applications should support are `'PICT'` and `'TEXT'`.

Listing 7-6 illustrates the application-defined routine `MyPastePict`, which retrieves a picture stored on the scrap. For example, a user may have copied to the Clipboard a picture created in another application and then pasted the picture into the application that defines `MyPastePict`. The `MyPastePict` procedure uses the Scrap Manager procedure `GetScrap` to get a handle to the data stored on the scrap; `MyPastePict` then coerces this handle to one of type `PicHandle`, which it can pass to the `DrawPicture` procedure in order to replay the drawing commands stored in the scrap.

Listing 7-6 Pasting in a picture from the scrap

```
PROCEDURE MyPastePict(destRect: Rect);
VAR
   myPic:      PicHandle;
   dataLength: LongInt;
   dontCare:   LongInt;
BEGIN
   myPic := PicHandle(NewHandle(0));   {allocate a handle for the picture}
   dataLength :=
            GetScrap(Handle(myPic),'PICT',dontCare);  {get picture in scrap}
   IF dataLength > 0 THEN {ensure there is PICT data}
   BEGIN
      MyAdjustDestRect(myPic,destRect);   {shown in Listing 7-7}
      DrawPicture(myPic,destRect);
      IF QDError <> noErr THEN
         ; {likely error is that there is insufficient memory}
   END
   ELSE
   ; {handle error for len < or = 0 here}
END;
```

Defining a Destination Rectangle

In addition to taking a handle to a picture as one parameter, `DrawPicture` also expects a destination rectangle as another parameter. You should specify this destination rectangle in coordinates local to the current graphics port. The `DrawPicture` procedure shrinks or stretches the picture as necessary to make it fit into this rectangle.

Listing 7-7 shows an application-defined routine called `MyAdjustDestRect` that centers the picture inside a destination rectangle, which is passed to `DrawPicture` when it's time to draw the picture. (`MyAdjustDestRect` first ensures that the picture fits inside the destination rectangle by scaling the picture if necessary.)

Listing 7-7 Adjusting the destination rectangle for a picture

```
PROCEDURE MyAdjustDestRect(aPict: PicHandle; VAR destRect: Rect);
VAR
   r:                      Rect;
   width, height:          Integer;
   scale, scaleH, scaleV:  Fixed;
BEGIN
   WITH destRect DO BEGIN  {determine width and height of destination rect}
      width := right - left;
      height := bottom - top;
   END;
   r := aPict^^.picFrame;        {get the bounding rectangle of the picture}
   OffsetRect(r, - r.left, - r.top);   {ensure upper-left corner is (0,0)}
   scale := Long2Fix(1);
   scaleH := FixRatio(width,r.right);     {get horizontal and vertical }
   scaleV := FixRatio(height,r.bottom);   { ratios of destination rectangle }
                                          { to bounding rectangle of picture}
   IF scaleH < scale THEN scale := scaleH;   {if bounding rect of picture }
   IF scaleV < scale THEN scale := scaleV;   { is greater than destination }
   IF scale <> Long2Fix(1) THEN              { rect, get scaling factors}
   BEGIN       {scale picture to fit inside destination rectangle}
      r.right := Fix2Long(FixMul(scale,Long2Fix(r.right)));
      r.bottom := Fix2Long(FixMul(scale,Long2Fix(r.bottom)));
    END;
   {next line centers the picture within the destination rectangle}
   OffsetRect(r,(width - r.right) DIV 2,(height - r.bottom) DIV 2);
   destRect := r;
END;
```

The application calling `MyAdjustDestRect` begins defining a destination rectangle by determining a target area within a window—perhaps the entire content area of a window, or perhaps an area selected by the user within a window. The application passes this rectangle to `MyAdjustDestRect`.

A bounding rectangle is stored in the `picFrame` field of the `Picture` record for every picture. The `MyAdjustDestRect` routine uses the boundaries for the picture to determine whether the picture fits within the destination rectangle. If the picture is larger than the destination rectangle, `MyAdjustDestRect` scales the picture to make it fit the destination rectangle.

The `MyAdjustDestRect` routine then centers the picture within the destination rectangle. Finally, `MyAdjustDestRect` assigns the boundary rectangle of the centered picture to be the new destination rectangle. By returning a destination rectangle whose dimensions are identical to those of the bounding rectangle for the picture, `MyAdjustDestRect` assures that the picture is not stretched when drawn into its window.

To display a picture at a resolution other than the one at which it was created, your application should compute an appropriate destination rectangle by scaling its width and height by the following factor:

scale factor = destination resolution / source resolution

For example, if a picture was created at 300 dpi and you want to display it at 75 dpi, then your application should compute the destination rectangle width and height as 1/4 of those of the picture's bounding rectangle. Your application can use the `GetPictInfo` function (described on page 7-47) to gather information about a picture. The `PictInfo` record (described on page 7-32) returned by `GetPictInfo` returns the picture's resolution in its `hRes` and `vRes` fields. The `sourceRect` field contains the bounding rectangle for displaying the image at its optimal resolution.

Drawing a Picture Stored in a 'PICT' Resource

To retrieve a picture stored in a `'PICT'` resource, specify its resource ID to the `GetPicture` function, which returns a handle to the picture. Listing 7-8 illustrates an application-defined routine, called `MyDrawResPICT`, that retrieves and draws a picture stored as a resource.

Listing 7-8 Drawing a picture stored in a resource file

```
PROCEDURE MyDrawResPICT(destRect: Rect; resID: Integer);
VAR
   myPic:   PicHandle;
BEGIN
   myPic := GetPicture(resID);   {get the picture from the resource fork}
   IF myPic <> NIL THEN BEGIN
      MyAdjustDestRect(myPic,destRect);{see Listing 7-7 on page 7-18}
      DrawPicture(myPic,destRect);
      IF QDError <> noErr THEN
         ; {likely error is that there is insufficient memory}
   END
   ELSE
   ; {handle the error here}
END;
```

When you are finished using a picture stored as a `'PICT'` resource, you should use the Resource Manager procedure `ReleaseResource` instead of the QuickDraw procedure `KillResource` to release its memory.

IMPORTANT

If you retrieve a picture stored in a `'PICT'` resource and pass its handle to the Window Manager procedure `SetWindowPic`, the Window Manager procedures `DisposeWindow` and `CloseWindow` do not delete it; instead, you must call `ReleaseResource` before calling `DisposeWindow` or `CloseWindow`. ▲

Saving Pictures

After creating or changing pictures, your application should allow the user to save them. To save a picture in a `'PICT'` file, you should use File Manager routines, such as `FSpCreate`, `FSpOpenDF`, `FSWrite`, and `FSClose`. The use of these routines is illustrated in Listing 7-9, and they are described in detail in the chapter "File Manager" in *Inside Macintosh: Files*. Remember that the first 512 bytes of a `'PICT'` file are reserved for your application's own purposes. As shown in Listing 7-9, your application should store the data (that is, the `Picture` record) after this 512-byte header.

Listing 7-9 Saving a picture as a `'PICT'` file

```
FUNCTION DoSavePICTAsCmd(picH: PicHandle): OSErr;
LABEL 8,9;
VAR
   myReply:                      StandardFileReply;
   err, ignore:                  OSErr;
   pictFileRefNum:               Integer;
   dataLength, zeroData, count:  LongInt;
BEGIN               {display the default Save dialog box}
   StandardPutFile('Save picture as:','untitled',myReply);
   err := noErr;  {return noErr if the user cancels}
   IF myReply.sfGood THEN
   BEGIN
      IF NOT myReply.sfReplacing THEN  {create the file if it doesn't exist}
         err := FSpCreate(myReply.sfFile,'WAVE','PICT',smSystemScript);
      IF err <> noErr THEN GOTO 9;
      err := FSpOpenDF(myReply.sfFile,fsRdWrPerm,pictFileRefNum); {open file}
      IF err <> noErr THEN GOTO 8;
      zeroData := 0;
      dataLength := 4;
      FOR count := 1 TO 512 DIV dataLength DO   {write the PICT file header}
         err := FSWrite(pictFileRefNum,dataLength,
                        @zeroData); {for this app, put 0's in header}
      IF err <> noErr THEN GOTO 8;
      dataLength := GetHandleSize(Handle(picH));
      HLock(Handle(picH)); {lock picture handle before writing data}
```

```
      err := FSWrite(pictFileRefNum,dataLength,Ptr(picH^)); {write picture }
                                                            { data to file}
       HUnlock(Handle(picH)); {unlock picture handle after writing data}
   END;
   8:
      ignore := FSClose(pictFileRefNum);  {close the file}
   9:
      DoSavePICTAsCmd := err;
END;
```

To save a picture in a `'PICT'` resource, you should use Resource Manager routines, such as `FSpOpenResFile` (to open your application's resource fork), `ChangedResource` (to change an existing `'PICT'` resource), `AddResource` (to add a new `'PICT'` resource), `WriteResource` (to write the data to the resource), and `CloseResFile` and `ReleaseResource` (to conclude saving the resource). These routines are described in the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.

To place a picture in the scrap—for example, in response to the user choosing the Copy command to copy a picture to the Clipboard—use the Scrap Manager function `PutScrap`, which is described in the chapter "Scrap Manager" in *Inside Macintosh: More Macintosh Toolbox*.

For large `'PICT'` files, it is useful to spool the picture data to disk instead of writing it all directly into memory. In low-memory conditions, for example, your application might find it useful to create a temporary file on disk for storing drawing instructions; your application can read this information as necessary. The application-defined routine `MyReplacePutPic` shown in Listing 7-10 replaces the `putPicProc` field of the current graphics port's `CQDProcs` record with an application-defined low-level routine, called `MyFilePutPic`. While QuickDraw's standard `StdPutPic` procedure writes picture data to memory, `MyFilePutPic` writes the picture data to disk. (Listing 7-3 on page 7-15 shows how to replace QuickDraw's standard `StdGetPic` procedure with one that reads data from a spool file.)

Listing 7-10 Replacing QuickDraw's standard low-level picture-writing routine

```
FUNCTION MyReplacePutPic: QDProcsPtr;
VAR
   currPort:      GrafPtr;
   customProcs:   QDProcs;
   customCProcs:  CQDProcs;
   savedProcs:    QDProcsPtr;
BEGIN
   GetPort(currPort);
   savedProcs := currPort^.grafProcs;  {save QDProcs or CQDProcs record }
                                       { for current graphics port}
   IF MyIsColorPort(currPort) THEN  {see Listing 7-4 on page 7-16}
```

```
   BEGIN
      SetStdCProcs(customCProcs);   {create new CQDProcs record containing }
                                    { standard Color QuickDraw low-level }
                                    { routines}

      customCProcs.putPicProc := @MyFilePutPic; {replace StdPutPic with }
                                                { address of custom }
                                                { low-level routine }
                                                { shown in Listing 7-11}
      currPort^.grafProcs := @customCProcs;  {replace current CQDProcs}
   END
   ELSE
   BEGIN       {perform similar work for a basic graphics port}
      SetStdProcs(customProcs);
      customProcs.putPicProc := @MyFilePutPic;
      currPort^.grafProcs := @customProcs;
   END;
   gPictureSize := 0;   {track the picture size}
   gSpoolPicture := PicHandle(NewHandle(0));
   MyReplacePutPic := savedProcs;         {return saved CQDProcs or QDProcs }
                                  { record for restoring at a later time}
END;
```

Listing 7-11 shows `MyFilePutPic`, which uses the File Manager function `FSWrite` to write picture data to the file with the file reference number assigned to the application-defined global variable `gPictFileRefNum`. Your application does not keep track of where `FSWrite` stops or resumes writing a file. After writing a portion of a file, `FSWrite` automatically handles where to begin writing next.

Listing 7-11 A custom low-level routine for spooling a picture to disk

```
PROCEDURE MyFilePutPic (dataPtr: Ptr; byteCount: Integer);
VAR
   dataLength: LongInt;
   myErr: OSErr;
BEGIN
   dataLength := byteCount;
   gPictureSize := gPictureSize + byteCount;
   myErr := FSWrite(gPictFileRefNum, dataLength, dataPtr);
   IF gSpoolPicture <> NIL THEN
   gSpoolPicture^^.picSize := gPictureSize;
END;
```

Gathering Picture Information

You can use the Picture Utilities routines to gather extensive information about pictures and to gather color information about pixel maps. You use the `GetPictInfo` function to gather information about a single picture, and you use the `GetPixMapInfo` function to gather color information about a single pixel map or bitmap. Each of these functions returns color and resolution information in a `PictInfo` record (described on page 7-32). A `PictInfo` record can also contain information about the drawing objects, fonts, and comments in a picture.

You can also survey multiple pictures, pixel maps, and bitmaps for this information. Use the `NewPictInfo` function to begin collecting pictures, pixel maps, and bitmaps for your survey. You also use `NewPictInfo` to specify how you would like the color, comment, and font information for the survey returned to you.

To add the information for a picture to your survey, use the `RecordPictInfo` function. To add the information for a pixel map or a bitmap to your survey, use the `RecordPixMapInfo` function. The `RetrievePictInfo` function collects the information about the pictures, pixel maps, and bitmaps that you have added to your survey. The `RetrievePictInfo` function returns this information in a `PictInfo` record.

For example, to use the ColorSync Utilities to match the colors in a single picture to an output device such as a color printer, an application might find it useful to find the `CMBeginProfile` picture comment, which marks the beginning of a color profile in a `Picture` record. (Color profiles and the ColorSync Utilities are described in *Inside Macintosh: Advanced Color Imaging*.) Listing 7-12 shows an application-defined routine, called `MyGetPICTProfileCount`, that uses `GetPictInfo` to record comments in a `CommentSpec` record (which is described on page 7-30). The `MyGetPICTProfileCount` routine uses the `CommentSpec` record to determine whether any color profiles are included in the picture as picture comments.

Listing 7-12 Looking for color profile comments in a picture


```
FUNCTION MyGetPICTProfileCount (hPICT: PicHandle; VAR count: Integer): OSErr;
VAR
   err:              OSErr;
   thePICTInfo:      PictInfo;
   verb:             Integer;
   colorsRequested:  Integer;
   colorPickMethod:  Integer;
   version:          Integer;
   pCommentSpec:     CommentSpecPtr;
   i:                Integer;
BEGIN
   count := 0;
   verb := recordComments;
   colorsRequested := 0;
   colorPickMethod := systemMethod;
   version := 0;
   err := GetPictInfo(hPICT, thePICTInfo, verb, colorsRequested,
                      colorPickMethod, version);
   IF ((err = noErr) AND (thePICTInfo.commentHandle <> NIL)) THEN
   BEGIN
      pCommentSpec := thePICTInfo.commentHandle^;
      FOR i := 1 TO thePICTInfo.uniqueComments DO
      BEGIN
         IF (pCommentSpec^.ID = CMBeginProfile) THEN
            BEGIN
               count := pCommentSpec^.count;
               LEAVE;
            END;
         pCommentSpec :=
            CommentSpecPtr(ORD4(pCommentSpec)+Sizeof(CommentSpec));
      END;
      {clean up allocations made by GetPictInfo}
      DisposeHandle(Handle(thePICTInfo.commentHandle));
   END;
   MyGetPICTProfileCount := err;
END;
```

If you want information about the colors of a picture or pixel map, you indicate to the Picture Utilities how many colors (up to 256) you want to know about, what method to use for selecting the colors, and whether you want the selected colors returned in a `Palette` record or `ColorTable` record.

The Picture Utilities provide two color-picking methods: one that gives you the most frequently used colors and one that gives you the widest range of colors. Each has advantages in different situations. For example, suppose the picture of a forest image contains 400 colors, of which 300 are greens, 80 are browns, and the rest are a scattering of golden sunlight effects. If you ask for the 250 most used colors, you will probably receive all greens. If you ask for a range of 250 colors, you will receive an assortment stretching from the greens and golds to the browns, including colors in between that might not actually appear in the image. You can also supply a color-picking method of your own, as described in "Application-Defined Routines" beginning on page 7-61.

Your application can then use the color information returned by the Picture Utilities in conjunction with the Palette Manager to provide the best selection of colors for displaying the picture on an 8-bit indexed device.

IMPORTANT

When you ask for color information about a picture, the Picture Utilities take into account only the version 2 and extended version 2 picture opcodes `RGBFgCol`, `RGBBkCol`, `BkPixPat`, `PnPixPat`, `FillPixPat`, and `HiliteColor` (as well as pixel map or bitmap data). Each occurrence of these opcodes is treated as one pixel, regardless of the number and sizes of the objects drawn with that color. If you need an accurate set of colors from a complex picture, create an image of the picture in an offscreen graphics world and call the `GetPixMapInfo` function to obtain color information about that pixel map for that graphics world. ▲

Pictures Reference

This section describes the data structures, routines, and resources provided by QuickDraw for creating and drawing pictures and by the Picture Utilities for gathering information about pictures and pixel maps.

"Data Structures" shows the Pascal data structures for the `Picture`, `OpenCPicParams`, `CommentSpec`, `FontSpec`, and `PictInfo` records.

"QuickDraw and Picture Utilities Routines" describes QuickDraw routines for creating, drawing, and disposing of pictures, and it describes Picture Utilities routines for collecting information about pictures, pixel maps, and bitmaps. "Application-Defined Routines" describes how you can define your own method for selecting colors from pictures and pixel maps.

"Resources" describes the picture (`'PICT'`) resource and the color-picking method (`'cpmt'`) resource.

See Appendix A at the back of this book for a list of picture opcodes.

Data Structures

This section shows the Pascal data structures for the `Picture`, `OpenCPicParams`, `CommentSpec`, `FontSpec`, and `PictInfo` records.

When you use the `OpenCPicture` or `OpenPicture` function, QuickDraw begins collecting your subsequent drawing commands in a `Picture` record. When you use the `GetPicture` function to retrieve a picture stored in a resource, `GetPicture` reads the resource into memory as a `Picture` record.

When you use the `OpenCPicture` function to begin creating a picture, you must pass it information in an `OpenCPicParams` record. This record provides a simple mechanism for specifying resolutions when creating images.

When you use the `GetPictInfo` function, it returns information in a `PictInfo` record. When you gather this information for multiple pictures, pixel maps, or bitmaps, the `RetrievePictInfo` function also returns a `PictInfo` record containing this information.

If you specify the `recordComments` constant in the `verb` parameter to the `GetPictInfo` function or `NewPictInfo` function, your application receives a `PictInfo` record that includes a handle to a `CommentSpec` record. A `CommentSpec` record contains information about the comments in a picture.

If you specify the `recordFontInfo` constant in the `verb` parameter to the `GetPictInfo` function or `NewPictInfo` function, the function returns a `PictInfo` record that includes a handle to a `FontSpec` record. A `FontSpec` record contains information about the fonts in a picture.

Picture

When you use the `OpenCPicture` or `OpenPicture` function (described on page 7-37 and page 7-39, respectively), QuickDraw begins collecting your subsequent drawing commands in a `Picture` record. (You use the `ClosePicture` procedure, described on page 7-42, to complete a picture definition.) When you use the `GetPicture` function (described on page 7-46) to retrieve a picture stored in a resource, `GetPicture` reads the resource into memory as a `Picture` record. (`'PICT'` resources are described on page 7-67.) By using the `DrawPicture` procedure (described on page 7-44), you can draw onscreen the picture defined by the commands stored in the `Picture` record.

A `Picture` record is defined as follows:

```
TYPE Picture =
RECORD
   picSize:    Integer; {for a version 1 picture: its size}
   picFrame:   Rect;    {bounding rectangle for the picture}
   {variable amount of picture data in the form of opcodes}
END;
```

Field descriptions

`picSize` The size of the rest of this record for a version 1 picture. To maintain compatibility with the version 1 picture format, the `picSize` field was not changed for the version 2 picture or extended version 2 formats. The information in this field is useful only for version 1 pictures, which cannot exceed 32 KB in size. Because version 2 and extended version 2 pictures can be much larger than the 32 KB limit imposed by the 2-byte `picSize` field, you should use the Memory Manager function `GetHandleSize` to determine the size of a picture in memory, the File Manager function `PBGetFInfo` to determine the size of a picture in a `'PICT'` file, and the Resource Manager function `MaxSizeResource` to determine the size of a `'PICT'` resource. (See *Inside Macintosh: Memory*, *Inside Macintosh: Files*, and *Inside Macintosh: More Macintosh Toolbox* for more information about these functions.)

`picFrame` The bounding rectangle for the picture defined in the rest of this record. The `DrawPicture` procedure uses this rectangle to scale the picture if you draw it into a destination rectangle of a different size.

Picture comments and compact drawing instructions in the form of picture opcodes compose the rest of this record.

A picture opcode is a number that the `DrawPicture` procedure uses to determine what object to draw or what mode to change for subsequent drawing. For debugging purposes, picture opcodes are listed in Appendix A at the back of this book. Your application generally should not read or write this picture data directly. Instead, your application should use the `OpenCPicture` (or `OpenPicture`), `ClosePicture`, and `DrawPicture` routines to process these opcodes.

The `Picture` record can also contain picture comments. Created by applications using the `PicComment` procedure, picture comments contain data or commands for special processing by output devices, such as PostScript printers. The `PicComment` procedure is described on page 7-40, and picture comments are described in greater detail in Appendix B in this book.

You can use File Manager routines to save the picture in a file of type `'PICT'`, you can use Resource Manager routines to save the picture in a resource of type `'PICT'`, and you can use the Scrap Manager procedure `PutScrap` to store the picture in `'PICT'` scrap format. See the chapter “File Manager” in *Inside Macintosh: Files* and the chapters “Resource Manager” and “Scrap Manager” in *Inside Macintosh: More Macintosh Toolbox* for more information about saving files, resources, and scrap data.

OpenCPicParams

When you use the `OpenCPicture` function (described on page 7-37) to begin creating a picture, you must pass it information in an `OpenCPicParams` record. This record provides a simple mechanism for specifying resolutions when creating images. For example, applications that create pictures from scanned images can specify resolutions higher than 72 dpi for these pictures in `OpenCPicParams` records.

An `OpenCPicParams` record is defined as follows:

```
TYPE OpenCPicParams =
RECORD
   srcRect:    Rect;       {optimal bounding rectangle for }
                           { displaying picture at resolution }
                           { indicated in hRes, vRes fields}
   hRes:       Fixed;      {best horizontal resolution; }
                           { $00480000 specifies 72 dpi}
   vRes:       Fixed;      {best vertical resolution; }
                           { $00480000 specifies 72 dpi}
   version:    Integer;    {set to -2}
   reserved1:  Integer;    {reserved; set to 0}
   reserved2:  LongInt;    {reserved; set to 0}
END;
```

Field descriptions

`srcRect` The optimal bounding rectangle for the resolution indicated by the next two fields. When you later call the `DrawPicture` procedure (described on page 7-44) to play back the saved picture, you supply a destination rectangle, and `DrawPicture` scales the picture so that it is completely aligned with the destination rectangle. To display a picture at a resolution other than that specified in the next two fields, your application should compute an appropriate destination rectangle by scaling its width and height by the following factor:

scale factor = destination resolution / source resolution

`hRes` The best horizontal resolution for the picture. Notice that this value is of type `Fixed`—a value of $0048000 specifies a horizontal resolution of 72 dpi.

`vRes` The best vertical resolution for the picture. Notice that this value is of type `Fixed`—a value of $00480000 specifies a horizontal resolution of 72 dpi.

`version` Always set this field to –2.

`reserved1` Reserved; set to 0.

`reserved2` Reserved; set to 0.

CommentSpec

If you specify the `recordComments` constant in the `verb` parameter to the `GetPictInfo` function (described on page 7-47) or the `NewPictInfo` function (described on page 7-53), you receive a `PictInfo` record (described beginning on page 7-32) that includes in its `commentHandle` field a handle to an array of `CommentSpec` records. The `uniqueComments` field of the `PictInfo` record indicates the number of `CommentSpec` records in this array.

The `CommentSpec` record is defined as follows:

```
TYPE CommentSpec =   {comment specification record}
RECORD
   count:   Integer; {number of times this type of comment }
                     { occurs in the picture or survey}
   ID:      Integer; {value identifying this type of comment}
END;
```

Field descriptions

`count` The number of times this kind of picture comment occurs in the picture specified to the `GetPictInfo` function or in all the pictures examined with the `NewPictInfo` function.

`ID` The value set in the `kind` parameter when the picture comment was created with the `PicComment` procedure. The `PicComment` procedure is described on page 7-40. The values of many common IDs are listed in Appendix B in this book.

When you are finished using the information returned in a `CommentSpec` record, you should use the `DisposeHandle` procedure (described in *Inside Macintosh: Memory*) to dispose of the memory allocated to it.

Listing 7-12 on page 7-25 illustrates how to count the number of picture comments by examining a `CommentSpec` record.

FontSpec

If you specify the `recordFontInfo` constant in the `verb` parameter to the `GetPictInfo` function (described on page 7-47) or the `NewPictInfo` function (described on page 7-53), your application receives a `PictInfo` record (described beginning on page 7-32) that includes in its `fontHandle` field a handle to an array of `FontSpec` records. The `uniqueFonts` field of the `PictInfo` record indicates the number of `FontSpec` records in this array. (For bitmap fonts, a font is a complete set of glyphs in one size, typeface, and style—for example, 12-point Geneva italic. For outline fonts, a font is a complete set of glyphs in one typeface and style—for example, Geneva italic.)

The FontSpec record is defined as follows:

```
TYPE FontSpec =         {font specification record}
RECORD
   pictFontID: Integer; {font ID as stored in the picture}
   sysFontID:  Integer; {font family ID}
   size:       ARRAY[0..3] OF LongInt;
                        {each bit set from 1 to 127 indicates a }
                        { point size at that value; if bit 0 is }
                        { set, then a size larger than 127 }
                        { points is found}
   style:      Integer; {styles used for this font family}
   nameOffset: LongInt; {offset to font name stored in the }
                        { data structure indicated by the }
                        { fontNamesHandle field of the PictInfo }
                        { record}
END;
```

Field descriptions

`pictFontID` The ID number of the font as it is stored in the picture.

`sysFontID` The number that identifies the resource file (of type `'FOND'`) that specifies the font family. Every font family—which consists of a complete set of fonts for one typeface including all available styles and sizes of the glyphs in that typeface—has a unique font family ID, in a range of values that determines the script system to which the font family belongs.

`size` The point sizes of the fonts in the picture. The field contains 128 bits, in which a bit is set for each point size encountered, from 1 to 127 points. Bit 0 is set if a size larger than 127 is found.

`style` The styles for this font family at any of its sizes. The values in this field can also be represented with the `Style` data type:

```
TYPE
   StyleItem = (bold, italic, underline, outline,
                shadow, condense, extend);
   Style = SET OF StyleItem;
```

`nameOffset` The offset into the list of font names (indicated by the `fontNamesHandle` field of the `PictInfo` record) at which the name for this font family is stored. A font name, such as Geneva, is given to a font family to distinguish it from other font families.

When you are finished using the information returned in a `FontSpec` record, you should use the `DisposeHandle` procedure (described in *Inside Macintosh: Memory*) to dispose of the memory allocated to it.

See the chapter "Font Manager" in *Inside Macintosh: Text* for more information about fonts.

PictInfo

When you use the `GetPictInfo` function (described on page 7-47) to collect information about a picture, or when you use the `GetPixMapInfo` function (described on page 7-50) to collect color information about a pixel map or bitmap, the function returns the information in a `PictInfo` record. When you gather this information for multiple pictures, pixel maps, or bitmaps, the `RetrievePictInfo` function (described on page 7-58) also returns a `PictInfo` record containing this information.

Initially, all of the fields in a new `PictInfo` record are set to `NIL`. Relevant fields are set to appropriate values depending on the information you request using the Picture Utilities functions as described in this chapter.

The `PictInfo` record is defined as follows:

```
TYPE PictInfo =
RECORD
   version:          Integer;          {Picture Utilities version number}
   uniqueColors:     LongInt;          {total colors in survey}
   thePalette:       PaletteHandle;    {handle to a Palette record--NIL for }
                                       { a bitmap in a basic graphics port}
   theColorTable:    CTabHandle;       {handle to a ColorTable record--NIL }
                                       { for a bitmap in a basic graphics }
                                       { port}
   hRes:             Fixed;            {best horizontal resolution (dpi)}
   vRes:             Fixed;            {best vertical resolution (dpi)}
   depth:            Integer;          {greatest pixel depth}
   sourceRect:       Rect;             {optimal bounding rectangle for }
                                       { picture for display at resolution }
                                       { specified in hRes and vRes fields}
   textCount:        LongInt;          {number of text strings in picture(s)}
   lineCount:        LongInt;          {number of lines in picture(s)}
   rectCount:        LongInt;          {number of rectangles in picture(s)}
   rRectCount:       LongInt;          {number of rounded rectangles in }
                                       { picture(s)}
   ovalCount:        LongInt;          {number of ovals in picture(s)}
   arcCount:         LongInt;          {number of arcs and wedges in }
                                       { picture(s)}
```

```
   polyCount:        LongInt;          {number of polygons in picture(s)}
   regionCount:      LongInt;          {number of regions in picture(s)}
   bitMapCount:      LongInt;          {number of bitmaps}
   pixMapCount:      LongInt;          {number of pixel maps}
   commentCount:     LongInt;          {number of comments in picture(s)}
   uniqueComments:   LongInt;          {number of different comments }
                                       { (by ID) in picture(s)}
   commentHandle:    CommentSpecHandle;{handle to an array of CommentSpec }
                                       { records for picture(s)}
   uniqueFonts:      LongInt;          {number of fonts in picture(s)}
   fontHandle:       FontSpecHandle;   {handle to an array of FontSpec }
                                       { records for picture(s)}
   fontNamesHandle:  Handle;           {handle to list of font names for }
                                       { picture(s)}
   reserved1:        LongInt;
   reserved2:        LongInt;
END;
```

Field descriptions

`version` The version number of the Picture Utilities, currently set to 0.

`uniqueColors` The number of colors in the picture specified to the `GetPictInfo` function, or the number of colors in the pixel map or bitmap specified to the `GetPixMapInfo` function, or the total number of colors for all the pictures, pixel maps, and bitmaps returned by the `RetrievePictInfo` function. The number of colors returned in this field is limited by the accuracy of the Picture Utilities' color bank for color storage. See "Application-Defined Routines" beginning on page 7-61 for information about the Picture Utility's color bank and about how you can create your own for selecting colors.

`thePalette` A handle to the resulting `Palette` record if you specified to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function that colors be returned in a `Palette` record. That `Palette` record contains either the number of colors you specified to the function or—if there aren't that many colors in the pictures, pixel maps, or bitmaps—the number of colors found. Depending on the constant you pass in the `verb` parameter to the function, the `Palette` record contains either the most used or the widest range of colors in the pictures, pixel maps, and bitmaps. On Macintosh computers running basic QuickDraw only, this field is always returned as `NIL`. See the chapter "Palette Manager" in *Inside Macintosh: Advanced Color Imaging* for more information about `Palette` records.

`theColorTable` A handle to the resulting `ColorTable` record if you specified to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function that colors be returned in a `ColorTable` record. If the pictures, pixel maps, or bitmaps contain fewer colors found than you specified to the function, the unused entries in the `ColorTable` record are filled with black. Depending on the constant you pass in the `verb` parameter to the function, the `ColorTable` record contains either the most used or the widest range of colors in the pictures, pixel maps, and bitmaps. The chapter "Color QuickDraw" in this book describes `ColorTable` records. On Macintosh computers running basic QuickDraw only, this field is always returned as `NIL`.

If a picture has more than 256 colors or has pixel depths of 32 bits, then Color QuickDraw translates the colors in the `ColorTable` record to 16-bit depths. In such a case, the returned colors might have a slight loss of resolution, and the `uniqueColors` field reflects the number of colors distinguishable at that pixel depth.

`hRes` The horizontal resolution of the current picture, pixel map, or bitmap retrieved by the `GetPictInfo` or `GetPixMapInfo` function; the greatest horizontal resolution from all pictures, pixel maps, and bitmaps retrieved by the `RetrievePictInfo` function.

`vRes` The vertical resolution of the current picture, pixel map, or bitmap retrieved by the `GetPictInfo` or `GetPixMapInfo` function; the greatest vertical resolution of all pictures, pixel maps, and bitmaps retrieved by the `RetrievePictInfo` function. Note that although the values of the `hRes` and `vRes` fields are usually the same, they don't have to be.

`depth` The pixel depth of the picture specified to the `GetPictInfo` function or the pixel map specified to the `GetPixMapInfo` function. When you use the `RetrievePictInfo` function, this field contains the deepest pixel depth of all pictures or pixel maps retrieved by the function.

`sourceRect` The optimal bounding rectangle for displaying the picture at the resolution indicated by the `hRes` and `vRes` fields. The upper-left corner of the rectangle is always (0,0). Pictures created with the `OpenCPicture` function have the `hRes`, `vRes`, and `sourceRect` fields built into their `Picture` records. For pictures created by `OpenPicture`, the `hRes` and `vRes` fields are set to 72 dpi, and the source rectangle is calculated using the `picFrame` field of the `Picture` record for the picture.

`textCount` The number of text strings in the picture specified to the `GetPictInfo` function, or the total number of text objects in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps specified to `GetPixMapInfo` or `RetrievePictInfo`, this field is set to 0.

`lineCount` The number of lines in the picture specified to the `GetPictInfo` function, or the total number of lines in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.

`rectCount`	The number of rectangles in the picture specified to the `GetPictInfo` function, or the total number of rectangles in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.
`rRectCount`	The number of rounded rectangles in the picture specified to the `GetPictInfo` function, or the total number of rounded rectangles in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.
`ovalCount`	The number of ovals in the picture specified to the `GetPictInfo` function, or the total number of ovals in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.
`arcCount`	The number of arcs and wedges in the picture specified to the `GetPictInfo` function, or the total number of arcs and wedges in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.
`polyCount`	The number of polygons in the picture specified to the `GetPictInfo` function, or the total number of polygons in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.
`regionCount`	The number of regions in the picture specified to the `GetPictInfo` function, or the total number of regions in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.
`bitMapCount`	The total number of bitmaps in the survey.
`pixMapCount`	The total number of pixel maps in the survey.
`commentCount`	The number of comments in the picture specified to the `GetPictInfo` function, or the total number of comments in all the pictures retrieved by the `RetrievePictInfo` function. This field is valid only if you specified to the `GetPictInfo` or `NewPictInfo` function that comments be returned in a `CommentSpec` record, described on page 7-30. For pixel maps and bitmaps, this field is set to 0.
`uniqueComments`	The number of picture comments that have different IDs in the picture specified to the `GetPictInfo` function, or the total number of picture comments with different IDs in all the pictures retrieved by the `RetrievePictInfo` function. (The values for many common IDs are listed in Appendix B, “Using Picture Comments for Printing,” in this book.) This field is valid only if you specify that comments be returned in a `CommentSpec` record. For pixel maps and bitmaps, this field is set to 0.
`commentHandle`	A handle to an array of `CommentSpec` records, described on page 7-30. For pixel maps and bitmaps, this field is set to `NIL`.

`uniqueFonts` The number of different fonts in the picture specified to the `GetPictInfo` function, or the total number of different fonts in all the pictures retrieved by the `RetrievePictInfo` function. For bitmap fonts, a font is a complete set of glyphs in one size, typeface, and style—for example, 12-point Geneva italic. For outline fonts, a font is a complete set of glyphs in one typeface and style—for example, Geneva italic.

This field is valid only if you specify that fonts be returned in a `FontSpec` record, which is described on page 7-30. For pixel maps and bitmaps, this field is set to 0.

`fontHandle` A handle to a list of `FontSpec` records, described on page 7-30. For pixel maps and bitmaps, this field is set to `NIL`.

`fontNamesHandle`
A handle to the names of the fonts in the picture retrieved by the `GetPictInfo` function or the pictures retrieved by the `RetrievePictInfo` function. The offset to a particular name is stored in the `nameOffset` field of the `FontSpec` record for that font. A font name is a name, such as Geneva, given to one font family to distinguish it from other font families.

When you are finished with this information, be sure to dispose of it. You can dispose of `Palette` records by using the `DisposePalette` procedure (which is described in the chapter "Palette Manager" in *Inside Macintosh: Advanced Color Imaging*). You can dispose of `ColorTable` records by using the `DisposeCTable` procedure (described in the chapter "Color QuickDraw" in this book). You can dispose of other allocations with the `DisposeHandle` procedure (described in *Inside Macintosh: Memory*).

QuickDraw and Picture Utilities Routines

This section describes QuickDraw routines for creating, drawing, and disposing of pictures, and it describes Picture Utilities routines for collecting information about pictures, pixel maps, and bitmaps.

Creating and Disposing of Pictures

Use the `OpenCPicture` function to begin defining a picture; `OpenCPicture` collects your subsequent drawing commands—which are described in the other chapters of this book—in a `Picture` record. You can use the `PicComment` procedure to include picture comments in your picture definition. To complete the collection of drawing and picture comment commands that define your picture, use the `ClosePicture` procedure. When you are finished using a picture not stored in a `'PICT'` resource, use the `KillPicture` procedure to release its memory. (To release the memory for a picture stored in a `'PICT'` resource, use the Resource Manager procedure `ReleaseResource`.)

The `OpenCPicture` function works on all Macintosh computers running System 7. Pictures created with the `OpenCPicture` function can be drawn on all versions of Macintosh system software. The `OpenPicture` function, which was created for earlier versions of system software, is described here for completeness.

OpenCPicture

To begin defining a picture in extended version 2 format, use the `OpenCPicture` function.

```
FUNCTION OpenCPicture (newHeader: OpenCPicParams): PicHandle;
```

`newHeader` An `OpenCPicParams` record, which is defined as follows (see page 7-29 for a description of the `OpenCPicParams` data type):

```
TYPE OpenCPicParams =
RECORD
   srcRect:    Rect;    {optimal bounding rectangle }
                        { for displaying picture at }
                        { resolution indicated in }
                        { hRes, vRes fields}
   hRes:       Fixed;   {best horizontal resolution; }
                        { $00480000 specifies 72 dpi}
   vRes:       Fixed;   {best vertical resolution; }
                        { $00480000 specifies 72 dpi}
   version:    Integer; {set to -2}
   reserved1:  Integer; {reserved; set to 0}
   reserved2:  LongInt; {reserved; set to 0}
END;
```

DESCRIPTION

The `OpenCPicture` function returns a handle to a new `Picture` record (described on page 7-27). Use the `OpenCPicture` function to begin defining a picture; `OpenCPicture` collects your subsequent drawing commands in this record. When defining a picture, you can use all other QuickDraw drawing routines described in this book, with the exception of `CopyMask`, `CopyDeepMask`, `SeedFill`, `SeedCFill`, `CalcMask`, and `CalcCMask`. (Nor can you use the `PlotCIcon` procedure, described in *Inside Macintosh: More Macintosh Toolbox*.)

You can also use the `PicComment` procedure (described on page 7-40) to include picture comments in your picture definition.

The `OpenCPicture` function creates pictures in the extended version 2 format. This format permits your application to specify resolutions when creating images.

Use the `OpenCPicParams` record you pass in the `newHeader` parameter to specify the horizontal and vertical resolution for the picture, and specify an optimal bounding rectangle for displaying the picture at this resolution. When you later call the `DrawPicture` procedure (described on page 7-44) to play back the saved picture, you supply a destination rectangle, and `DrawPicture` scales the picture so that it is completely aligned with the destination rectangle. To display a picture at a resolution other than that at which it was created, your application should compute an appropriate destination rectangle by scaling its width and height by the following factor:

scale factor = destination resolution / source resolution

For example, if a picture was created at 300 dpi and you want to display it at 75 dpi, then your application should compute the destination rectangle width and height as 1/4 of those of the picture's bounding rectangle.

The `OpenCPicture` function calls the `HidePen` procedure, so no drawing occurs on the screen while the picture is open (unless you call the `ShowPen` procedure just after `OpenCPicture`, or you called `ShowPen` previously without balancing it by a call to `HidePen`).

Use the handle returned by `OpenCPicture` when referring to the picture in subsequent routines, such as the `DrawPicture` procedure.

After defining the picture, close it by using the `ClosePicture` procedure, described on page 7-42. To draw the picture, use the `DrawPicture` procedure, described on page 7-44.

After creating the picture, your application can use the `GetPictInfo` function (described on page 7-47) to gather information about it. The `PictInfo` record (described on page 7-32) returned by `GetPictInfo` returns the picture's resolution and optimal bounding rectangle.

SPECIAL CONSIDERATIONS

When creating a picture, you should generally use the `ClosePicture` procedure to finish it before you open the Printing Manager with the `PrOpen` procedure. There are two main reasons for this. First, you should allow the printing driver to use as much memory as possible. Second, the Printing Manager creates its own type of graphics port—one that replaces the standard QuickDraw drawing operations stored in the `grafProcs` field of a `CGrafPort` or `GrafPort` record; to avoid unexpected results when creating a picture, you should draw into a graphics port created with QuickDraw instead of drawing into a printing port created by the Printing Manager.

After calling `OpenCPicture`, be sure to finish your picture definition by calling `ClosePicture` before you call `OpenCPicture` again. You cannot nest calls to `OpenCPicture`.

Always use the `ClipRect` procedure to specify a clipping region appropriate for your picture before you call `OpenCPicture`. If you do not use `ClipRect` to specify a clipping region, `OpenCPicture` uses the clipping region specified in the current graphics port. If the clipping region is very large (as it is when a graphics port is initialized) and you scale the picture when drawing it, the clipping region can become invalid when `DrawPicture` scales the clipping region—in which case, your picture will

not be drawn. On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when you draw it. Setting a clipping region equal to the port rectangle of the current graphics port, as shown in Listing 7-1 on page 7-11, always sets a valid clipping region.

The `OpenCPicture` function may move or purge memory.

SEE ALSO

The `PrOpen` procedure is described in the chapter "Printing Manager" in this book. The `ClipRect` procedure is described in the chapter "Basic QuickDraw" in this book. The `ShowPen` and `HidePen` procedures are described in the chapter "QuickDraw Drawing" in this book.

Listing 7-1 on page 7-11 illustrates the use of the `OpenCPicture` function.

OpenPicture

The `OpenPicture` function, which was created for earlier versions of system software, is described here for completeness. To create a picture, you should use the `OpenCPicture` function, which allows you to specify resolutions for your pictures, as explained in the previous routine description.

```
FUNCTION OpenPicture (picFrame: Rect): PicHandle;
```

`picFrame` The bounding rectangle for the picture. The `DrawPicture` procedure uses this rectangle to scale the picture if you draw it into a destination rectangle of a different size.

DESCRIPTION

The `OpenPicture` function returns a handle to a new `Picture` record (described on page 7-27). You can use the `OpenPicture` function to begin defining a picture; `OpenPicture` collects your subsequent drawing commands in this record. When defining a picture, you can use all other QuickDraw drawing routines described in this book, with the exception of `CopyMask`, `CopyDeepMask`, `SeedFill`, `SeedCFill`, `CalcMask`, and `CalcCMask`. (Nor can you use the `PlotCIcon` procedure, described in *Inside Macintosh: More Macintosh Toolbox*.) You can also use the `PicComment` procedure (described on page 7-40) to include picture comments in your picture definition.

The `OpenPicture` function creates pictures in the version 2 format on computers with Color QuickDraw when the current graphics port is a color graphics port. Pictures created in this format support color drawing operations at 72 dpi. On computers supporting only basic QuickDraw, or when the current graphics port is a basic graphics port, this function creates pictures in version 1 format. Pictures created in version 1 format support only black-and-white drawing operations at 72 dpi.

Use the handle returned by `OpenPicture` when referring to the picture in subsequent routines, such as the `DrawPicture` procedure.

The `OpenPicture` function calls the `HidePen` procedure, so no drawing occurs on the screen while the picture is open (unless you call the `ShowPen` procedure just after `OpenPicture` or you called `ShowPen` previously without balancing it by a call to `HidePen`).

After defining the picture, close it by using the `ClosePicture` procedure, described on page 7-42. To draw the picture, use the `DrawPicture` procedure, described on page 7-44.

SPECIAL CONSIDERATIONS

The version 2 and version 1 picture formats support only 72-dpi resolution. The `OpenCPicture` function creates pictures in the extended version 2 format. The extended version 2 format, which is created by the `OpenCPicture` function on all Macintosh computers running System 7, permits your application to specify additional resolutions when creating images.

See the description of the `OpenCPicture` function for its list of special considerations, all of which apply to `OpenPicture`.

Version 1 pictures are limited to 32 KB. You can determine the picture size while it's being formed by calling the Memory Manager function `GetHandleSize`.

PicComment

To insert a picture comment into a picture that you are defining or into your printing code, use the `PicComment` procedure.

```
PROCEDURE PicComment (kind,dataSize: Integer;
                      dataHandle: Handle);
```

`kind` The type of comment. Because the vast majority of picture comments are interpreted by printer drivers, the constants that you can supply in this parameter—and values they represent—are listed in Appendix B, "Using Picture Comments for Printing," in this book.

`dataSize` Size of any additional data passed in the `dataHandle` parameter. Data sizes for the various kinds of picture comments are listed in Appendix B, "Using Picture Comments for Printing," in this book. If no additional data is used, specify 0 in this parameter.

`dataHandle`
A handle to additional data, if used. If no additional data is used, specify `NIL` in this parameter.

DESCRIPTION

When used after your application begins creating a picture with the `OpenCPicture` (or `OpenPicture`) function, the `PicComment` procedure inserts the specified comment into the `Picture` record. When sent to a printer driver after your application uses the `PrOpenPage` procedure, `PicComment` passes the data or commands in the specified comment directly to the printer.

Picture comments contain data or commands for special processing by output devices, such as printers. For example, using the `SetLineWidth` comment, your application can draw hairlines—which are not available with standard QuickDraw calls—on any PostScript LaserWriter printer and on the QuickDraw LaserWriter SC printer.

Usually printer drivers process picture comments, but applications can also do so. For your application to process picture comments, it must replace the `StdComment` procedure pointed to by the `commentProc` field of the `CQDProcs` or `QDProcs` record, which in turn is pointed to by the `grafProcs` field of a `CGrafPort` or `GrafPort` record. The default `StdComment` procedure provided by QuickDraw does no comment processing whatsoever. You can use the `SetStdCProcs` procedure to assist you in changing the `CQDProcs` record, and you can use the `SetStdProcs` procedure to assist you in changing the `QDProcs` record.

If you create and process your own picture comments, you should define comments so that they contain information that identifies your application (to avoid using the same comments as those used by Apple or by other third-party products). You should define a comment as an `ApplicationComment` comment type with a `kind` value of 100. The first 4 bytes of the data for the comment should specify your application's signature. (See the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about application signatures.) You can use the next 2 bytes to identify the type of comment—that is, to specify a `kind` value to your own application.

Suppose your application signature were `'WAVE'`, and you wanted to use the value 128 to identify a `kind` value to your own application. You would supply values to the `kind` and `data` parameters to `PicComment` as follows:

`kind` = 100; `data` = `'WAVE'` [4 bytes] + 128 [2 bytes] + additional data [*n* bytes]

Your application can then parse the first 6 bytes of the comment to determine whether and how to process the rest of the data in the comment. It is up to you to publish information about your comments if you wish them to be understood and used by other applications.

SPECIAL CONSIDERATIONS

These former picture comments are now obsolete: `SetGrayLevel`, `ResourcePS`, `PostScriptFile`, and `TextIsPostScript`.

The `PicComment` procedure may move or purge memory.

SEE ALSO

See Appendix B, "Using Picture Comments for Printing," in this book for information about using picture comments to print with features that are unavailable with QuickDraw. See the chapter "QuickDraw Drawing" in this book for information about the `QDProcs` record and the `StdComment` and `SetStdProcs` procedures. See the chapter "Color QuickDraw" in this book for information about the `CQDProcs` record and the `SetStdCProcs` procedure.

ClosePicture

To complete the collection of drawing commands and picture comments that define your picture, use the `ClosePicture` procedure.

```
PROCEDURE ClosePicture;
```

DESCRIPTION

The `ClosePicture` procedure stops collecting drawing commands and picture comments for the currently open picture. You should perform one and only one call to `ClosePicture` for every call to the `OpenCPicture` (or `OpenPicture`) function.

The `ClosePicture` procedure calls the `ShowPen` procedure, balancing the call made by `OpenCPicture` (or `OpenPicture`) to the `HidePen` procedure.

SEE ALSO

The `ShowPen` and `HidePen` procedures are described in the chapter "QuickDraw Drawing" in this book.

Listing 7-1 on page 7-11 illustrates the use of the `ClosePicture` procedure.

KillPicture

To release the memory occupied by a picture not stored in a `'PICT'` resource, use the `KillPicture` procedure.

```
PROCEDURE KillPicture (myPicture: PicHandle);
```

`myPicture` A handle to the picture whose memory can be released.

DESCRIPTION

The `KillPicture` procedure releases the memory occupied by the picture whose handle you pass in the `myPicture` parameter. Use this only when you're completely finished with a picture.

SPECIAL CONSIDERATIONS

If you use the Window Manager procedure `SetWindowPic` to store a picture handle in the window record, you can use the Window Manager procedure `DisposeWindow` or `CloseWindow` to release the memory allocated to the picture; these procedures automatically call `KillPicture` for the picture.

If the picture is stored in a `'PICT'` resource, you must use the Resource Manager procedure `ReleaseResource` instead of `KillPicture`. The Window Manager procedures `DisposeWindow` and `CloseWindow` will not delete it; instead, you must call `ReleaseResource` before calling `DisposeWindow` or `CloseWindow`.

The `KillPicture` procedure may move or purge memory.

SEE ALSO

The `ReleaseResource` procedure is described in the chapter "Resource Manager" in *Inside Macintosh: Macintosh Toolbox*, and the `SetWindowPic`, `DisposeWindow`, and `CloseWindow` procedures are described in the chapter "Window Manager," in *Inside Macintosh: Macintosh Toolbox Essentials*.

Drawing Pictures

To draw a picture, use the `DrawPicture` procedure. You must access a picture through its handle. When creating pictures, the `OpenCPicture` and `OpenPicture` functions (described beginning on page 7-37) return their handles. You can use the `GetPicture` function to get a handle to a QuickDraw picture stored in a `'PICT'` resource.

Note

To get a handle to a QuickDraw picture stored in a `'PICT'` file, you must use File Manager routines, as described in "Drawing a Picture Stored in a 'PICT' File" beginning on page 7-13. To get a picture stored in the scrap, use the Scrap Manager procedure `GetScrap` to get a handle to its data and then coerce this handle to one of type `PicHandle`, as shown in Listing 7-6 on page 7-17. ◆

DrawPicture

To draw a picture on any type of output device, use the `DrawPicture` procedure.

```
PROCEDURE DrawPicture (myPicture: PicHandle; dstRect: Rect);
```

`myPicture` A handle to the picture to be drawn.

`dstRect` A destination rectangle, specified in coordinates local to the current graphics port, in which to draw the picture. The `DrawPicture` procedure shrinks or expands the picture as necessary to align the borders of its bounding rectangle with the rectangle you specify in this parameter. To display a picture at a resolution other than that at which it was created, your application should compute an appropriate destination rectangle by scaling its width and height by the following factor:

scale factor = destination resolution / source resolution

For example, if a picture was created at 300 dpi and you want to display it at 75 dpi, then your application should compute the destination rectangle width and height as 1/4 of those of the picture's bounding rectangle. Your application can use the `GetPictInfo` function (described on page 7-47) to gather information about a picture. The `PictInfo` record (described on page 7-32) returned by `GetPictInfo` returns the picture's resolution in its `hRes` and `vRes` fields. The `sourceRect` field contains the bounding rectangle for displaying the image at its optimal resolution.

DESCRIPTION

Within the rectangle that you specify in the `dstRect` parameter, the `DrawPicture` procedure draws the picture that you specify in the `myPicture` parameter.

The `DrawPicture` procedure passes any picture comments to the `StdComment` procedure pointed to by the `commentProc` field of the `CQDProcs` or `QDProcs` record, which in turn is pointed to by the `grafProcs` field of a `CGrafPort` or `GrafPort` record. The default `StdComment` procedure provided by QuickDraw does no comment processing whatsoever. If you want to process picture comments when drawing a picture, you can use the `SetStdCProcs` procedure to assist you in changing the `CQDProcs` record, and you can use the `SetStdProcs` procedure to assist you in changing the `QDProcs` record.

SPECIAL CONSIDERATIONS

Always use the `ClipRect` procedure to specify a clipping region appropriate for your picture before defining it with the `OpenCPicture` (or `OpenPicture`) function. If you do not use `ClipRect` to specify a clipping region, `OpenCPicture` uses the clipping region specified in the current graphics port. If the clipping region is very large (as it is when a graphics port is initialized) and you want to scale the picture, the clipping region

can become invalid when `DrawPicture` scales the clipping region—in which case, your picture will not be drawn. On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when `DrawPicture` draws it. Setting a clipping region equal to the port rectangle of the current graphics port, as shown in Listing 7-1 on page 7-11, always sets a valid clipping region.

When it scales, `DrawPicture` changes the size of the font instead of scaling the bits. However, the widths used by bitmap fonts are not always linear. For example, the 12-point width isn't exactly 1/2 of the 24-point width. This can cause lines of text to become slightly longer or shorter as the picture is scaled. The difference is often insignificant, but if you are trying to draw a line of text that fits exactly into a box (a spreadsheet cell, for example), the difference can become noticeable to the user—most typically, at print time. The easiest way to avoid such problems is to specify a destination rectangle that is the same size as the bounding rectangle for the picture. Otherwise, your application may need to directly process the opcodes in the picture instead of using `DrawPicture`.

You may also have disappointing results if the fonts contained in an image are not available on the user's system. Before displaying a picture, your application may want to use the Picture Utilities to determine what fonts are contained in the picture, and then use Font Manager routines to determine whether the fonts are available on the user's system. If they are not, you can use Dialog Manager routines to display an alert box warning the user of display problems.

If there is insufficient memory to draw a picture in Color QuickDraw, the `QDError` function (described in the chapter "Color QuickDraw" in this book) returns the result code `noMemForPictPlaybackErr`.

The `DrawPicture` procedure may move or purge memory.

SEE ALSO

Listing 7-1 on page 7-11 illustrates how to use `DrawPicture` after creating a picture while your application is running; Listing 7-2 on page 7-13 illustrates how to use `DrawPicture` after reading in a picture stored in a `'PICT'` file; Listing 7-6 on page 7-17 illustrates how to use `DrawPicture` after reading in a picture stored in the scrap; and Listing 7-8 on page 7-20 illustrates how to use `DrawPicture` after reading in a picture stored in a `'PICT'` resource.

See the chapter "Font Manager" in *Inside Macintosh: Text* for information about Font Manager routines; see the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about Dialog Manager routines.

GetPicture

Use the `GetPicture` function to get a handle to a picture stored in a `'PICT'` resource.

```
FUNCTION GetPicture (picID: Integer): PicHandle;
```

`picID` The resource ID for a `'PICT'` resource.

DESCRIPTION

The `GetPicture` function returns a handle to the picture stored in the `'PICT'` resource with the ID that you specify in the `picID` parameter. You can pass this handle to the `DrawPicture` procedure (described on page 7-44) to draw the picture stored in the resource.

The `GetPicture` function calls the Resource Manager procedure `GetResource` as follows:

```
GetResource('PICT',picID)
```

If the resource can't be read, `GetPicture` returns `NIL`.

SPECIAL CONSIDERATIONS

To release the memory occupied by a picture stored in a `'PICT'` resource, use the Resource Manager procedure `ReleaseResource`.

The `GetPicture` function may move or purge memory.

SEE ALSO

The `GetResource` and `ReleaseResource` procedures are described in the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*. The `'PICT'` resource is described on page 7-67.

Collecting Picture Information

You can use the Picture Utilities routines described in this section to gather extensive information about pictures and to gather color information about pixel maps and bitmaps. On an 8-bit indexed device, for example, you might want your application to determine the 256 most-used colors in a picture composed of millions of colors. Your application can then use the Palette Manager (as described in *Inside Macintosh: Advanced Color Imaging*) to make these colors available for the window in which your application needs to draw the picture.

You use the `GetPictInfo` function to gather information about a single picture, and you use the `GetPixMapInfo` function to gather color information about a single pixel map or bitmap. Each of these functions returns color and resolution information in a `PictInfo` record. A `PictInfo` record for a picture also contains additional information, such as the resolution of the picture, and information about the fonts and comments contained in the picture.

You can also survey multiple pictures, pixel maps, and bitmaps for this information. Use the `NewPictInfo` function to begin collecting pictures, pixel maps, and bitmaps for your survey. You also use `NewPictInfo` to specify how you would like the color, comment, and font information for the survey returned to you.

To add the information for a picture to your survey, use the `RecordPictInfo` function. To add the information for a pixel map or a bitmap to your survey, use the `RecordPixMapInfo` function. The `RetrievePictInfo` function collects the information about the pictures, pixel maps, and bitmaps that you have added to the survey. The `RetrievePictInfo` function returns this information in a `PictInfo` record.

When you are finished with this information, use the `DisposePictInfo` function to dispose of the private data structures allocated by the `NewPictInfo` function.

Note

The Picture Utilities also collect information from black-and-white pictures and bitmaps, and they are supported in System 7 even by computers running only basic QuickDraw. However, when collecting color information on a computer running only basic QuickDraw, the Picture Utilities return `NIL` instead of a handle to a `Palette` or `ColorTable` record. ◆

GetPictInfo

Use the `GetPictInfo` function to gather information about a single picture.

```
FUNCTION GetPictInfo (thePictHandle: PicHandle;
                      VAR thePictInfo: PictInfo; verb: Integer;
                      colorsRequested: Integer;
                      colorPickMethod: Integer;
                      version: Integer): OSErr;
```

`thePictHandle`
A handle to a picture.

`thePictInfo`
A pointer to a `PictInfo` record, which will hold information about the picture. The `PictInfo` record is described on page 7-32.

verb A value indicating what type of information you want GetPictInfo to return in the PictInfo record. You can use any or all of the following constants or the sum of the integers they represent:

```
CONST
returnColorTable  = 1;  {return a ColorTable record}
returnPalette     = 2;  {return a Palette record}
recordComments    = 4;  {return comment information}
recordFontInfo    = 8;  {return font information}
suppressBlackAndWhite
                  = 16; {don't include black and }
                        { white with returned colors}
```

Because the Palette Manager adds black and white when creating a Palette record, you can specify the number of colors you want minus 2 in the colorsRequested parameter and specify the suppressBlackAndWhite constant in the verb parameter when gathering colors destined for a Palette record or a screen.

colorsRequested

From 1 to 256, the number of colors you want in the ColorTable or Palette record returned via the PictInfo record. If you are not requesting colors (that is, if you pass the recordComments or recordFontInfo constant in the verb parameter), this function does not return colors, in which case you may instead pass 0 here.

colorPickMethod

The method by which colors are selected for the ColorTable or Palette record returned via the PictInfo record. You can use one of the following constants or the integer it represents:

```
CONST
systemMethod   = 0;  {let Picture Utilities choose }
                     { the method (currently they }
                     { always choose popularMethod)}
popularMethod  = 1;  {return most frequently used }
                     { colors}
medianMethod   = 2;  {return a weighted distribution }
                     { of colors}
```

You can also create your own color-picking method in a resource file of type 'cpmt' and pass its resource ID in the colorPickMethod parameter. The resource ID must be greater than 127.

version Always set this parameter to 0.

DESCRIPTION

In the `PictInfo` record to which the parameter `thePictInfo` points, the `GetPictInfo` function returns information about the picture you specify in the `thePictHandle` parameter. Initially, all of the fields in a new `PictInfo` record are set to `NIL`. Relevant fields are set to appropriate values depending on the information you request using the `GetPictInfo` function.

Use the `verb` parameter to specify whether you want color information (in a `ColorTable` record, a `Palette` record, or both), whether you want picture comment information, and whether you want font information. If you want color information, be sure to use the `colorPickMethod` parameter to specify the method by which to select colors.

The Picture Utilities provide two color-picking methods: one (specified by the `popularMethod` constant) that gives you the most frequently used colors and one (specified by the `medianMethod` constant) that gives you the widest range of colors. Each has advantages in different situations. For example, suppose the picture of a forest image contains 400 colors, of which 300 are greens, 80 are browns, and the rest are a scattering of golden sunlight effects. If you ask for the 250 most used colors, you will probably receive all greens. If you ask for a range of 250 colors, you will receive an assortment stretching from the greens and golds to the browns, including colors in between that might not actually appear in the image. If you specify the `systemMethod` constant, the Picture Utilities choose the method; currently they always choose `popularMethod`. You can also supply a color-picking method of your own.

If your application uses more than one color-picking method, it should present the user with a choice of which method to use.

When you are finished with the information in the `PictInfo` record, be sure to dispose of it. Use the Memory Manager procedure `DisposeHandle` to dispose of the `PictInfo`, `CommentSpec`, and `FontSpec` records. Dispose of the `Palette` record by using the `DisposePalette` procedure. Dispose of the `ColorTable` record by using the `DisposeCTable` procedure.

SPECIAL CONSIDERATIONS

When you ask for color information, `GetPictInfo` takes into account only the version 2 and extended version 2 picture opcodes `RGBFgCol`, `RGBBkCol`, `BkPixPat`, `PnPixPat`, `FillPixPat`, and `HiliteColor` (as well as pixel map or bitmap data). Each occurrence of these opcodes is treated as 1 pixel, regardless of the number and sizes of the objects drawn with that color. If you need an accurate set of colors from a complex picture, create an image of the picture in an offscreen pixel map, and then call the `GetPixMapInfo` function (described on page 7-50) to obtain color information about that pixel map.

The `GetPictInfo` function returns a bit depth of 1 on QuickTime-compressed `'PICT'` files. However, when QuickTime is installed, QuickTime decompresses and displays the image correctly.

The `GetPictInfo` function may move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `GetPictInfo` function are

Trap macro	Selector
`_Pack15`	$0800

RESULT CODES

`pictInfoVersionErr`	–11000	Version number not 0
`pictInfoVerbErr`	–11002	Invalid verb combination specified
`cantLoadPickMethodErr`	–11003	Custom pick method not in resource chain
`colorsRequestedErr`	–11004	Number out of range or greater than that passed to `NewPictInfo`
`pictureDataErr`	–11005	Invalid picture data

SEE ALSO

The `PictInfo` record is described on page 7-32, the `CommentSpec` record is described on page 7-30, and the `FontSpec` record is described on page 7-30. The `ColorTable` record is described in the chapter "Color QuickDraw" in this book; the `Palette` record is described in the chapter "Palette Manager" in *Inside Macintosh: Advanced Color Imaging*. See "Application-Defined Routines" beginning on page 7-61 for more information about creating your own color-picking method for the `colorPickMethod` parameter.

The `DisposePalette` procedure is described in the chapter "Palette Manager" in *Inside Macintosh: Advanced Color Imaging*. The `DisposeCTable` procedure is described in the chapter "Color QuickDraw" in this book. The `DisposeHandle` procedure is described in the chapter "Memory Manager" in *Inside Macintosh: Memory*.

Listing 7-12 on page 7-25 illustrates the use of the `GetPictInfo` function.

GetPixMapInfo

Use the `GetPixMapInfo` function to gather color information about a single pixel map or bitmap.

```
FUNCTION GetPixMapInfo (thePixMapHandle: PixMapHandle;
                        VAR thePictInfo: PictInfo; verb: Integer;
                        colorsRequested: Integer;
                        colorPickMethod: Integer;
                        version: Integer): OSErr;
```

`thePixMapHandle`
A handle to a pixel map or bitmap.

`thePictInfo`
A pointer to a `PictInfo` record, which will hold information about a pixel map or bitmap. The `PictInfo` record is described on page 7-32.

`verb` A value indicating whether you want color information returned in a `ColorTable` record, a `Palette` record, or both. You can also request that black and white not be included among the returned colors. You can use any or all of the following constants or the sum of the integers they represent:

```
CONST
returnColorTable  = 1;  {return a ColorTable record}
returnPalette     = 2;  {return a Palette record}
suppressBlackAndWhite
                  = 16; {don't include black and }
                        { white with returned colors}
```

Because the Palette Manager adds black and white when creating a `Palette` record, you can specify the number of colors you want minus 2 in the `colorsRequested` parameter and specify the constant `suppressBlackAndWhite` in the `verb` parameter when gathering colors destined for a `Palette` record or a screen.

`colorsRequested`
From 1 to 256, the number of colors you want in the `ColorTable` or `Palette` record returned via the `PictInfo` record.

`colorPickMethod`
The method by which colors are selected for the `ColorTable` or `Palette` record returned via the `PictInfo` record. You can use one of the following constants or the integer it represents:

```
CONST
systemMethod   = 0;  {let Picture Utilities choose }
                     { the method (currently they }
                     { always choose popularMethod)}
popularMethod  = 1;  {return most frequently used }
                     { colors}
medianMethod   = 2;  {return a weighted distribution }
                     { of colors}
```

You can also create your own color-picking method in a resource file of type `'cpmt'` and pass its resource ID in the `colorPickMethod` parameter. The resource ID must be greater than 127.

`version` Always set this parameter to 0.

DESCRIPTION

For the pixel map (or bitmap) whose handle you pass in the `thePixMapHandle` parameter, the `GetPixMapInfo` function returns color information in the `PictInfo` record that you point to in the parameter `thePictInfo`. Initially, all of the fields in a new `PictInfo` record are set to `NIL`. Relevant fields are set to appropriate values depending on the information you request using the `GetPixMapInfo` function.

Use the `verb` parameter to specify whether you want color information returned in a `ColorTable` record, a `Palette` record, or both, and use the `colorPickMethod` parameter to specify the method by which to select colors.

The Picture Utilities provide two color-picking methods: one (specified by the `popularMethod` constant) that gives you the most frequently used colors and one (specified by the `medianMethod` constant) that gives you the widest range of colors. If you specify the `systemMethod` constant, the Picture Utilities choose the method; currently they always choose `popularMethod`. You can also supply a color-picking method of your own.

When you are finished with the information in the `PictInfo` record, be sure to dispose of it. Use the Memory Manager procedure `DisposeHandle` to dispose of the `PictInfo` record. Dispose of the `Palette` record by using the `DisposePalette` procedure. Dispose of the `ColorTable` record by using the `DisposeCTable` procedure.

SPECIAL CONSIDERATIONS

The `GetPixMapInfo` function may move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `GetPixMapInfo` function are

Trap macro	Selector
`_Pack15`	$0801

RESULT CODES

`pictInfoVersionErr`	–11000	Version number not 0
`pictInfoVerbErr`	–11002	Invalid verb combination specified
`cantLoadPickMethodErr`	–11003	Custom pick method not in resource chain
`colorsRequestedErr`	–11004	Number out of range or greater than that passed to `NewPictInfo`

SEE ALSO

See "Application-Defined Routines" beginning on page 7-61 for more information about creating your own color-picking method for the `colorPickMethod` parameter. The `PictInfo` record is described on page 7-32; the `PixMapHandle` data type and the `ColorTable` record are described in the chapter "Color QuickDraw" in this book; the `Palette` record is described in *Inside Macintosh: Advanced Color Imaging*.

The `DisposePalette` procedure is described in *Inside Macintosh: Advanced Color Imaging*. The `DisposeCTable` procedure is described in the chapter "Color QuickDraw" in this book. The `DisposeHandle` procedure is described in the chapter "Memory Manager" in *Inside Macintosh: Memory*.

NewPictInfo

You can survey multiple pictures for such information as colors, picture comments, and fonts, and you can survey multiple pixel maps and bitmaps for color information. Use the `NewPictInfo` function to begin collecting pictures, pixel maps, and bitmaps for your survey.

```
FUNCTION NewPictInfo (VAR thePictInfoID: PictInfoID;
                      verb: Integer; colorsRequested: Integer;
                      colorPickMethod: Integer;
                      version: Integer): OSErr;
```

`thePictInfoID`
A unique value that denotes your collection of pictures, pixel maps, or bitmaps.

`verb` A value indicating what type of information you want the `RetrievePictInfo` function (described on page 7-58) to return in a `PictInfo` record (described on page 7-32). When collecting information about pictures, you can use any or all of the following constants or the sum of the integers they represent:

```
CONST
returnColorTable  = 1;  {return a ColorTable record}
returnPalette     = 2;  {return a Palette record}
recordComments    = 4;  {return comment information}
recordFontInfo    = 8;  {return font information}
suppressBlackAndWhite
                  = 16; {don't include black and }
                        { white with returned colors}
```

The constants `recordComments` and `recordFontInfo` and the values they represent have no effect when gathering information about the pixel maps and bitmaps included in your survey.

Because the Palette Manager adds black and white when creating a palette, you can specify the number of colors you want minus 2 in the `colorsRequested` parameter and specify the constant `suppressBlackAndWhite` in the `verb` parameter when gathering colors destined for a `Palette` record or a screen.

colorsRequested
: From 1 to 256, the number of colors you want included in the `ColorTable` or `Palette` record returned by the `RetrievePictInfo` function via a `PictInfo` record.

colorPickMethod
: The method by which colors are selected for the `ColorTable` or `Palette` record included in the `PictInfo` record returned by the `RetrievePictInfo` function. You can use one of the following constants or the integer it represents:

```
CONST
systemMethod   = 0;  {let Picture Utilities choose }
                     { the method (currently they }
                     { always choose popularMethod)}
popularMethod  = 1;  {return most frequently used }
                     { colors}
medianMethod   = 2;  {return a weighted distribution }
                     { of colors}
```

You can also create your own color-picking method in a resource file of type `'cpmt'` and pass its resource ID in the `colorPickMethod` parameter. The resource ID must be greater than 127.

version
: Always set this parameter to 0.

DESCRIPTION

In the `thePictInfoID` parameter, the `NewPictInfo` function returns a unique ID number for use when surveying multiple pictures, pixel maps, and bitmaps for information.

To add the information for a picture to your survey, use the `RecordPictInfo` function, which is described next. To add the information for a pixel map or a bitmap to your survey, use the `RecordPixMapInfo` function, which is described on page 7-57. For each of these functions, you identify the survey with the ID number returned by `NewPictInfo`.

Use the `RetrievePictInfo` function (described on page 7-58) to return information about the pictures, pixel maps, and bitmaps in the survey. Again, you identify the survey with the ID number returned by `NewPictInfo`. The `RetrievePictInfo` function returns your requested information in a `PictInfo` record.

Use the `verb` parameter for `NewPictInfo` to specify whether you want to gather comment or font information for the pictures in the survey. If you want to gather color information, use the `verb` parameter for `NewPictInfo` to specify whether you want this information in a `ColorTable` record, a `Palette` record, or both. The `PictInfo` record returned by the `RetrievePictInfo` function will then include a handle to a `ColorTable` record or a `Palette` record, or handles to both. If you want color information, be sure to use the `colorPickMethod` parameter to specify the method by which to select colors.

The Picture Utilities provide two color-picking methods: one (specified by the `popularMethod` constant) that gives you the most frequently used colors and one (specified by the `medianMethod` constant) that gives you the widest range of colors. If you specify the `systemMethod` constant, the Picture Utilities choose the method; currently they always choose `popularMethod`. You can also supply a color-picking method of your own.

SPECIAL CONSIDERATIONS

The `NewPictInfo` function may move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `NewPictInfo` function are

Trap macro	Selector
`_Pack15`	$0602

RESULT CODES

`pictInfoVersionErr`	–11000	Version number not 0
`pictInfoVerbErr`	–11002	Invalid verb combination specified
`cantLoadPickMethodErr`	–11003	Custom pick method not in resource chain
`colorsRequestedErr`	–11004	Number out of range or greater than that passed to `NewPictInfo`

SEE ALSO

The `PictInfo` record is described on page 7-32, the `CommentSpec` record is described on page 7-30, and the `FontSpec` record is described on page 7-30. The `ColorTable` record is described in the chapter "Color QuickDraw" in this book; the `Palette` record is described in *Inside Macintosh: Advanced Color Imaging*. See "Application-Defined Routines" beginning on page 7-61 for more information about creating your own color-picking method for the `colorPickMethod` parameter.

RecordPictInfo

To add a picture to an informational survey of multiple pictures, use the `RecordPictInfo` function.

```
FUNCTION RecordPictInfo (thePictInfoID: PictInfoID;
                         thePictHandle: PicHandle): OSErr;
```

`thePictInfoID`
The ID number—returned by the `NewPictInfo` function—that identifies the survey to which you are adding the picture. The `NewPictInfo` function is described on page 7-53.

`thePictHandle`
A handle to the picture being added to the survey.

DESCRIPTION

The `RecordPictInfo` function adds the picture you specify in the parameter `thePictHandle` to the survey of pictures identified by the parameter `thePictInfoID`. Use `RecordPictInfo` repeatedly to add additional pictures to your survey.

After you have collected all of the pictures you need, use the `RetrievePictInfo` function, described on page 7-58, to return information about pictures in the survey.

SPECIAL CONSIDERATIONS

When you ask for color information, `RecordPictInfo` takes into account only the version 2 and extended version picture opcodes `RGBFgCol`, `RGBBkCol`, `BkPixPat`, `PnPixPat`, `FillPixPat`, and `HiliteColor`. Each occurrence of these opcodes is treated as 1 pixel, regardless of the number and sizes of the objects drawn with that color. If you need an accurate set of colors from a complex picture, create an image of the picture in an offscreen pixel map, and then call the `GetPixMapInfo` function (described on page 7-50) to obtain color information about that pixel map.

The `RecordPictInfo` function may move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `RecordPictInfo` function are

Trap macro	Selector
`_Pack15`	$0403

RESULT CODES

`pictInfoIDErr`	–11001	Invalid picture information ID
`pictureDataErr`	–11005	Invalid picture data

RecordPixMapInfo

To add a pixel map or bitmap to an informational survey of multiple pixel maps and bitmaps, use the `RecordPictInfo` function.

```
FUNCTION RecordPixMapInfo (thePictInfoID: PictInfoID;
                           thePixMapHandle: PixMapHandle): OSErr;
```

`thePictInfoID`
The ID number—returned by the `NewPictInfo` function—that identifies the survey to which you are adding the pixel map or bitmap. The `NewPictInfo` function is described on page 7-53.

`thePixMapHandle`
A handle to a pixel map (or bitmap) to be added to the survey.

DESCRIPTION

The `RecordPixMapInfo` function adds the pixel map or bitmap you specify in the parameter `thePixMapHandle` to the survey identified by the parameter `thePictInfoID`. Use `RecordPictInfo` repeatedly to add additional pixel maps and bitmaps to your survey.

After you have collected all of the images you need, use the `RetrievePictInfo` function, described on page 7-58, to return information about all the images in the survey.

SPECIAL CONSIDERATIONS

The `RecordPixMapInfo` function may move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `RecordPixMapInfo` function are

Trap macro	Selector
`_Pack15`	$0404

RESULT CODES

`pictInfoIDErr`	–11001	Invalid picture information ID
`pictureDataErr`	–11005	Invalid picture data

RetrievePictInfo

Use the `RetrievePictInfo` function to return information about all the pictures, pixel maps, and bitmaps included in a survey.

```
FUNCTION RetrievePictInfo (thePictInfoID: PictInfoID;
                           VAR thePictInfo: PictInfo;
                           colorsRequested: Integer): OSErr;
```

`thePictInfoID`
The ID number—returned by the `NewPictInfo` function—that identifies the survey of pictures, pixel maps, and bitmaps. The `NewPictInfo` function is described on page 7-53.

`thePictInfo`
A pointer to the `PictInfo` record that holds information about the pictures or images in the survey. The `PictInfo` record is described on page 7-32.

`colorsRequested`
From 1 to 256, the number of colors you want returned in the `ColorTable` or `Palette` record included in the `PictInfo` record.

DESCRIPTION

In a `PictInfo` record that you point to in the parameter `thePictInfo`, the `RetrievePictInfo` function returns information about all of the pictures and images collected in the survey that you specify in the parameter `thePictInfoID`.

After using the `NewPictInfo` function to create a new survey, and then using `RecordPictInfo` to add pictures to your survey and `RecordPixMapInfo` to add pixel maps and bitmaps to your survey, you can call `RetrievePictInfo`.

When you are finished with the information in the `PictInfo` record, be sure to dispose of it. You can dispose of the `Palette` record by using the `DisposePalette` procedure. You can dispose of the `ColorTable` record by using the `DisposeCTable` procedure. You can dispose of other allocations with the `DisposeHandle` procedure. You should also use the `DisposePictInfo` function (described next) to dispose of the private data structures created by the `NewPictInfo` function.

SPECIAL CONSIDERATIONS

The `RetrievePictInfo` function may move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `RetrievePictInfo` function are

Trap macro	Selector
`_Pack15`	$0505

RESULT CODES

`pictInfoIDErr`	–11001	Invalid picture information ID
`colorsRequestedErr`	–11004	Number out of range or greater than that passed to `NewPictInfo`

SEE ALSO

The `DisposePalette` procedure is described in *Inside Macintosh: Advanced Color Imaging*. The `DisposeCTable` procedure is described in the chapter "Color QuickDraw" in this book. The `DisposeHandle` procedure is described in the chapter "Memory Manager" in *Inside Macintosh: Memory*.

DisposePictInfo

When you are finished gathering information from a survey of pictures, pixel maps, or bitmaps, use the `DisposePictInfo` function to dispose of the private data structures allocated by the `NewPictInfo` function. The `DisposePictInfo` function is also available as the `DisposPictInfo` function.

```
FUNCTION DisposePictInfo (thePictInfoID: PictInfoID): OSErr;
```

`thePictInfoID`
The unique identifier returned by `NewPictInfo`.

DESCRIPTION

The `DisposePictInfo` function disposes of the private data structures allocated by the `NewPictInfo` function, which is described on page 7-53.

The `DisposePictInfo` function does not dispose of any of the handles returned to you in a `PictInfo` record by the `RetrievePictInfo` function, which is described on page 7-58. Instead, you can dispose of a `Palette` record by using the `DisposePalette` procedure. You can dispose of a `ColorTable` record by using the `DisposeCTable` procedure. You can dispose of other allocations with the `DisposeHandle` procedure.

SPECIAL CONSIDERATIONS

The `DisposePictInfo` function may move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `DisposePictInfo` function are

Trap macro	Selector
`_Pack15`	$0206

RESULT CODE

`pictInfoIDErr`	–11001	Invalid picture information ID

SEE ALSO

The `DisposePalette` procedure is described in *Inside Macintosh: Advanced Color Imaging*. The `DisposeCTable` procedure is described in the chapter "Color QuickDraw" in this book. The `DisposeHandle` procedure is described in the chapter "Memory Manager" in *Inside Macintosh: Memory*.

Application-Defined Routines

As described in "Collecting Picture Information" beginning on page 7-46, your application can use the `GetPictInfo`, `GetPixMapInfo`, and `NewPictInfo` functions to gather information about pictures, pixel maps, and bitmaps. Each of these functions can gather up to 256 colors in `ColorTable` and `Palette` records. In the `colorPickMethod` parameter to these functions, you specify how they should select which colors to gather. These Picture Utilities functions provide two color-picking methods: the first selects the most frequently used colors, and the second selects a weighted distribution of the existing colors.

You can also create your own color-picking method. You must compile it as a resource of type `'cpmt'` and write its entry point in assembly language. To use this color-picking method (`'cpmt'`) resource, pass its resource ID (which must be greater than 127) in the `colorPickMethod` parameter to these Picture Utilities functions. These functions call your color-picking method's entry point and pass one of four possible selectors in register D0. These functions pass their parameters on the stack. As shown in Table 7-1, each selector requires your color-picking method to call a different routine, which should return its results in register D0.

Table 7-1 Routine selectors for an application-defined color-picking method

D0 value	Routine to call
0	`MyInitPickMethod`
1	`MyRecordColors`
2	`MyCalcColorTable`
3	`MyDisposeColorPickMethod`

Your color-picking method (`'cpmt'`) resource should include a routine that specifies its **color bank** (that is, the structure into which all the colors of a picture, pixel map, or bitmap are gathered) and allocates whatever data your color-picking method needs. This routine is explained next as a Pascal function declared as `MyInitPickMethod`.

Your `MyInitPickMethod` function can let the Picture Utilities generate a color bank consisting of a **histogram** (that is, frequency counts of each color) to a resolution of 5 bits per color. Or, your `MyInitPickMethod` function can specify that your application has its own custom color bank—for example, a histogram to a resolution of 8 bits per color.

If you create your own custom color bank, your `'cpmt'` resource should include a routine that gathers and stores colors; this routine is described on page 7-64 as a Pascal function declared as `MyRecordColors`.

For the number of colors your application requests using the Picture Utilities, your `'cpmt'` resource should include a routine that determines which colors to select from the color bank and then fills an array of `ColorSpec` records with those colors; this routine is described on page 7-65 as a Pascal function declared as `MyCalcColorTable`. The Picture Utilities function that your application initially called then returns these

colors in a `Palette` record or `ColorTable` record, as specified by your application when it first called the Picture Utilities function.

Your `'cpmt'` resource should include a routine that releases the memory allocated by your `MyInitPickMethod` routine. The routine that releases memory is described on page 7-67 as a Pascal function declared as `MyDisposeColorPickMethod`.

If your routines return an error, that error is passed back to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function, which in turn passes the error to your application as a function result.

MyInitPickMethod

Your color-picking method (`'cpmt'`) resource should include a routine that specifies its color bank and allocates whatever data your color-picking method needs. Here is how you would declare this routine if it were a Pascal function named `MyInitPickMethod`:

```
FUNCTION MyInitPickMethod (colorsRequested: Integer;
                           VAR dataRef: LongInt;
                           VAR colorBankType: Integer): OSErr;
```

`colorsRequested`
The number of colors requested by your application to be gathered for examination in a `ColorTable` or `Palette` record.

`dataRef` A handle to any data needed by your color-picking method; that is, if your application allocates and uses additional data, it should return a handle to it in this parameter.

`colorBankType`
The type of color bank your color-picking method uses. Your `MyInitPickMethod` routine should return one of three valid color bank types, which it can represent with one of these constants:

```
CONST
colorBankIsCustom       = -1; {gathers colors into a }
                              { custom color bank}
colorBankIsExactAnd555  = 0;  {gathers exact colors }
                              { if there are less }
                              { than 256 unique }
                              { colors in picture; }
                              { otherwise gathers }
                              { colors for picture }
                              { in a 5-5-5 histogram}
colorBankIs555          = 1;  {gathers colors into a }
                              { 5-5-5 histogram}
```

Return the `colorBankIs555` constant in this parameter if you want to let the Picture Utilities gather the colors for a picture or a pixel map into a 5-5-5 histogram. When you return the `colorBankIs555` constant, the Picture Utilities call your `MyCalcColorTable` routine with a pointer to the color bank (that is, to the 5-5-5 histogram). Your `MyCalcColorTable` routine (described on page 7-65) selects whatever colors it needs from this color bank. Then the Picture Utilities function called by your application returns these colors in a `Palette` record, a `ColorTable` record, or both, as requested by your application.

Return the `ColorBankIsExactAnd555` constant in this parameter to make the Picture Utilities return exact colors if there are less than 256 unique colors in the picture; otherwise, the Picture Utilities gather the colors for the picture in a 5-5-5 histogram, just as they do when you return the `colorBankIs555` constant. If the picture or pixel map has fewer colors than your application requests when it calls a Picture Utilities function, the Picture Utilities function returns all of the colors contained in the color bank. If the picture or pixel map contains more colors than your application requests, the Picture Utilities call your `MyCalcColorTable` routine to select which colors to return.

Return the `colorBankIsCustom` constant in this parameter if you want to implement your own color bank for storing the colors in a picture or a pixel map. For example, because the 5-5-5 histogram that the Picture Utilities provide gathers colors to a resolution of 5 bits per color, your application may want to create a histogram with a resolution of 8 bits per color. When you return the `colorBankIsCustom` constant, the Picture Utilities call your `MyRecordColors` routine (explained in the next routine description) to create this color bank. The Picture Utilities also call your `MyCalcColorTable` routine to select colors from this color bank.

DESCRIPTION

Your `MyInitPickMethod` routine should allocate whatever data your color-picking method needs and store a handle to your data in the location pointed to by the `dataRef` parameter. In the `colorBankType` parameter, your `MyInitPickMethod` routine must also return the type of color bank your color-picking method uses for color storage. If your `MyInitPickMethod` routine generates any error, it should return the error as its function result.

The 5-5-5 histogram that the Picture Utilities provide if you return the `ColorBankIs555` or `ColorBankIsExactAnd555` constant in the `colorBankType` parameter is like a reversed `cSpecArray` record, which is an array of `ColorSpec` records. (The `cSpecArray` and `ColorSpec` records are described in the chapter "Color QuickDraw" in this book.) This 5-5-5 histogram is an array of 32,768 integers, where the *index* into the array is the color: 5 bits of red, followed by 5 bits of green, followed by 5 bits of blue. Each *entry* in the array is the number of colors in the picture that are approximated by the index color for that entry.

For example, suppose there were three instances of the following color in the pixel map:

Red = %1101 1010 1010 1110

Green = %0111 1010 1011 0001

Blue = %0101 1011 0110 1010

This color would be represented by index % 0 11011-01111-01011 (in hexadecimal, $6DEB), and the value in the histogram at this index would be 3, because there are three instances of this color.

MyRecordColors

When you return the `colorBankIsCustom` constant in the `colorBankType` parameter to your `MyInitPickMethod` function (described in the preceding section), your color-picking method (`'cpmt'`) resource must include a routine that creates this color bank; for example, your application may want to create a histogram with a resolution of 8 bits per color. Here is how you would declare this routine if it were a Pascal function named `MyRecordColors`:

```
FUNCTION MyRecordColors (dataRef: LongInt;
                         colorsArray: RGBColorArray;
                         colorCount: LongInt;
                         VAR uniqueColors: LongInt): OSErr;
```

`dataRef` A handle to any data your method needs. Your application initially creates this handle using the `MyInitPickMethod` routine (explained in the preceding section).

`colorsArray`
An array of `RGBColor` records. (`RGBColor` records are described in the chapter "Color QuickDraw" in this book.) Your `MyRecordColors` routine should store the color information for this array of `RGBColor` records in a data structure of type `RGBColorArray`. You should define the `RGBColorArray` data type as follows:

```
TYPE RGBColorArray  =  ARRAY[0..0] OF RGBColor;
```

`colorCount`
The number of colors in the array specified in the `colorsArray` parameter.

`uniqueColors`
: Upon input: the number of unique colors already added to the array in the `colorsArray` parameter. (The Picture Utilities functions call your `MyRecordColors` routine once for every color in the picture, pixel map, or bitmap.) Your `MyRecordColors` routine must calculate the number of unique colors (to the resolution of the color bank) that are added by this call. Your `MyRecordColors` routine should add this amount to the value passed upon input in this parameter and then return the sum in this parameter.

DESCRIPTION

Your `MyRecordColors` routine should store each color encountered in a picture or pixel into its own color bank. The Picture Utilities call `MyRecordColors` only if your `MyInitPickMethod` routine returns the constant `colorBankIsCustom` in the `colorBankType` parameter. The Picture Utilities functions call `MyRecordColors` for all the colors in the picture, pixel map, or bitmap. If your `MyRecordColors` routine generates any error, it should return the error as its function result.

MyCalcColorTable

Your color-picking method (`'cpmt'`) resource should include a routine that selects as many colors as are requested by your application from the color bank for a picture or pixel map and then fills these colors into an array of `ColorSpec` records.

Here is how you would declare this routine if it were a Pascal function named `MyCalcColorTable`:

```
FUNCTION MyCalcColorTable (dataRef: LongInt;
                           colorsRequested: Integer;
                           colorBankPtr: Ptr;
                           VAR resultPtr: CSpecArray): OSErr;
```

`dataRef`
: A handle to any data your method needs. Your application initially creates this handle using the `MyInitPickMethod` routine (explained on page 7-62).

`colorsRequested`
: The number of colors requested by your application to be gathered for examination in a `ColorTable` or `Palette` record.

`colorBankPtr`

If your `MyInitPickMethod` routine (described on page 7-62) returned either the `colorBankIsExactAnd555` or `colorBankIs555` constant, then this parameter contains a pointer to the 5-5-5 histogram that describes all of the colors in the picture, pixel map, or bitmap being examined. (The format of the 5-5-5 histogram is explained in the routine description for the `MyInitPickMethod` routine.) Your `MyCalcColorTable` routine should examine these colors and then, using its own criterion for selecting the colors, fill in an array of `ColorSpec` records with the number of colors specified in the `colorsRequested` parameter.

If your `MyInitPickMethod` routine returned the `colorBankIsCustom` constant, then the value passed in this parameter is invalid. In this case, your `MyCalcColorTable` routine should use the custom color bank that your application created (as explained in the routine description for the `MyRecordColors` routine on page 7-64) for filling in an array `of ColorSpec` records with the number of colors specified in the `colorsRequested` parameter.

Your `MyCalcColorTable` function should return a pointer to this array of `ColorSpec` records in the next parameter.

`resultPtr` A pointer to the array of `ColorSpec` records to be filled with the number of colors specified in the `colorsRequested` parameter. The Picture Utilities function that your application initially called places these colors in a `Palette` record or `ColorTable` record, as specified by your application.

DESCRIPTION

Selecting from the color bank created for the picture, bitmap, or pixel map being examined, your `MyCalcColorTable` routine should fill an array of `ColorSpec` records with the number of colors requested in the `colorsRequested` parameter and return this array in the `resultPtr` parameter. If your `MyCalcColorTable` routine generates any error, it should return the error as its function result.

If more colors are requested than the picture contains, fill the remaining entries with black (0000 0000 0000).

The `colorBankPtr` parameter is of type `Ptr` because the data stored in the color bank is of the type specified by your `MyInitPickMethod` routine (described on page 7-62). Thus, if you specified `colorBankIs555` in the `colorBankType` parameter, the color bank would be an array of integers. However, if the Picture Utilities support other data types in the future, the `colorBankPtr` parameter could point to completely different data types.

SPECIAL CONSIDERATIONS

Always coerce the value passed in the `colorBankPtr` parameter to a pointer to an integer. In the future you may need to coerce this value to a pointer of the type you specify in your `MyInitPickMethod` function.

MyDisposeColorPickMethod

Your `'cpmt'` resource should include a routine that releases the memory allocated by your `MyInitPickMethod` routine (which is described on page 7-62). Here is how you would declare this routine if it were a Pascal function named `MyDisposeColorPickMethod`:

```
FUNCTION MyDisposeColorPickMethod (dataRef: LongInt): OSErr;
```

`dataRef` A handle to any data your method needs. Your application initially creates this handle using the `MyInitPickMethod` routine.

DESCRIPTION

Your `MyDisposeColorPickMethod` routine should release any memory that you allocated in your `MyInitPickMethod` routine. If your `MyDisposeColorPickMethod` routine generates any error, it should return the error as its function result.

Resources

This section describes the picture (`'PICT'`) resource and the color-picking method (`'cpmt'`) resource. You can use the `'PICT'` resource to save pictures in the resource fork of your application or document files. You can assemble your own color-picking method for use by the Picture Utilities in a `'cpmt'` resource.

The Picture Resource

A picture (`'PICT'`) resource contains QuickDraw drawing instructions that can be played back using the `DrawPicture` procedure.

You may find it useful to store pictures in the resource fork of your application or document file. For example, when the user chooses the About command in the Apple menu for your application, you might wish to display a window containing your company's logo. Or, if yours is a page-layout application, you might want to store all the images created by the user for a document as resources in the document file.

You can use high-level tools like the ResEdit resource editor, available from APDA, to create and store images as `'PICT'` resources for distribution with your files.

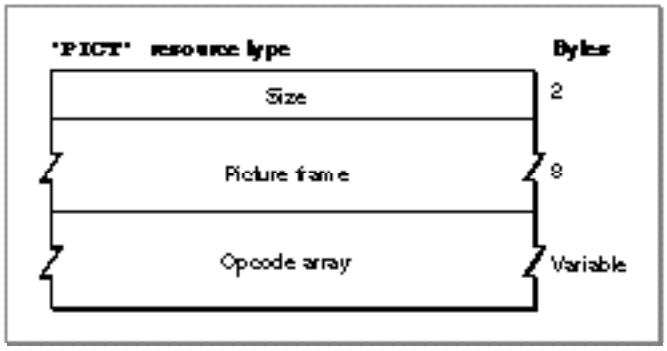
To save a picture in a `'PICT'` resource while your application is running, you should use Resource Manager routines, such as `FSpOpenResFile` (to open your application's resource fork), `ChangedResource` (to change an existing `'PICT'` resource), `AddResource` (to add a new `'PICT'` resource), `WriteResource` (to write the data to the resource), and `CloseResFile` and `ReleaseResource` (to conclude saving the resource). These routines are described in the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.

All 'PICT' resources must be marked purgeable, and they must have resource IDs greater than 127.

If you examine the compiled version of a 'PICT' resource, as represented in Figure 7-4, you find that it contains the following elements:

- The size of the resource—if the resource contains a picture created in the version 1 format. Because version 2 and extended version 2 pictures can be much larger than the 32 KB limit imposed by the size of this element, you should use the Resource Manager function MaxSizeResource (described in *Inside Macintosh: More Macintosh Toolbox*) to determine the size of a picture in version 2 and extended version 2 format.
- The bounding rectangle for the picture. The DrawPicture procedure uses this rectangle to scale the picture when you draw it into a destination rectangle of a different size.
- An array of picture opcodes. A picture opcode is a number that the DrawPicture procedure uses to determine what object to draw or what mode to change for subsequent drawing. For debugging purposes, picture opcodes are listed in Appendix A at the back of this book. Your application generally should not read or write this picture data directly. Instead, you should use the OpenCPicture (or OpenPicture), ClosePicture, and DrawPicture routines to process these opcodes.

Figure 7-4 Structure of a compiled picture ('PICT') resource



To retrieve a 'PICT' resource, specify its resource ID to the GetPicture function, described on page 7-46, which returns a handle to the picture. Listing 7-8 on page 7-20 illustrates an application-defined routine that retrieves and draws a picture stored as a resource.

Appendix A, "Picture Opcodes," shows examples of disassembled picture resources.

The Color-Picking Method Resource

The resource type for an assembled color-picking routine is 'cpmt'. It must have a resource ID greater than 127. The resource data is the assembled code of the routine. See "Application-Defined Routines" beginning on page 7-61 for information about creating a color-picking method resource.

Summary of Pictures and the Picture Utilities

Pascal Summary

Constants

```
CONST
   {color-picking methods}
   systemMethod   = 0;  {let Picture Utilities choose the method (currently }
                        { they always choose popularMethod)}
   popularMethod  = 1;  {return most frequently used colors}
   medianMethod   = 2;  {return a weighted distribution of colors}

   {picture information to be returned by Picture Utilities}
   returnColorTable        = 1;  {return a ColorTable record}
   returnPalette           = 2;  {return a Palette record}
   recordComments          = 4;  {return comment information}
   recordFontInfo          = 8;  {return font information}
   suppressBlackAndWhite   = 16; {don't include black and }
                                 { white with returned colors}

   {color bank types}
   colorBankIsCustom       = -1; {gathers colors into a custom color bank}
   colorBankIsExactAnd555  = 0;  {gathers exact colors if there are less }
                                 { than 256 unique colors in picture; }
                                 { otherwise gathers colors for picture }
                                 { in a 5-5-5 histogram}
   colorBankIs555          = 1;  {gathers colors in a 5-5-5 histogram}
```

Data Types

```
TYPE
   PicPtr      = ^Picture;
   PicHandle   = ^PicPtr;
   Picture     =
   RECORD                  {picture record}
      picSize:    Integer; {for a version 1 picture: its size}
```

```
   picFrame:   Rect;    {bounding rectangle for the picture}
   {variable amount of picture data in the form of opcodes}
END;

OpenCPicParams =
RECORD
   srcRect:    Rect;       {optimal bounding rectangle for displaying }
                           { picture at resolution indicated in hRes, }
                           { vRes fields}
   hRes:       Fixed;      {best horizontal resolution; }
                           { $00480000 specifies 72 dpi}
   vRes:       Fixed;      {best vertical resolution; }
                           { $00480000 specifies 72 dpi}
   version:    Integer;    {set to -2}
   reserved1:  Integer;    {reserved; set to 0}
   reserved2:  LongInt;    {reserved; set to 0}
END;

CommentSpecHandle = ^CommentSpecPtr;
CommentSpecPtr    = ^CommentSpec;
CommentSpec       =     {comment specification record}
RECORD
   count:   Integer;    {number of times this type of comment }
                        { occurs in the picture or survey}
   ID:      Integer;    {value identifying this type of comment}
END;

FontSpecHandle = ^FontSpecPtr;
FontSpecPtr    = ^FontSpec;
FontSpec       =          {font specification record}
RECORD
   pictFontID: Integer;    {font ID as stored in the picture}
   sysFontID:  Integer;    {font family ID}
   size:       ARRAY[0..3] OF LongInt;
                           {each bit set from 1 to 127 indicates a }
                           { point size at that value; if bit 0 is }
                           { set, then a size larger than 127 }
                           { points is found}
   style:      Integer;    {styles used for this font family}
   nameOffset: LongInt;    {offset to font name stored in the }
                           { data structure indicated by the }
                           { fontNamesHandle field of the PictInfo }
                           { record}
END;
```

```
PictInfoID     = LongInt;

PictInfoHandle = ^PictInfoPtr;
PictInfoPtr    = ^PictInfo;
PictInfo       =                        {picture information record}
RECORD
   version:          Integer;           {Picture Utilities version number}
   uniqueColors:     LongInt;           {total colors in survey}
   thePalette:       PaletteHandle;     {handle to a Palette record--NIL }
                                        { for a bitmap in a basic }
                                        { graphics port}
   theColorTable:    CTabHandle;        {handle to a ColorTable record-- }
                                        { NIL for a bitmap in a basic }
                                        { graphics port}
   hRes:             Fixed;             {best horizontal resolution (dpi)}
   vRes:             Fixed;             {best vertical resolution (dpi)}
   depth:            Integer;           {greatest pixel depth}
   sourceRect:       Rect;              {optimal bounding rectangle for }
                                        { picture for display at }
                                        { resolution specified in hRes }
                                        { and vRes fields}
   textCount:        LongInt;           {number of text strings in }
                                        { picture(s)}
   lineCount:        LongInt;           {number of lines in picture(s)}
   rectCount:        LongInt;           {number of rectangles in }
                                        { picture(s)}
   rRectCount:       LongInt;           {number of rounded rectangles in }
                                        { picture(s)}
   ovalCount:        LongInt;           {number of ovals in picture(s)}
   arcCount:         LongInt;           {number of arcs and wedges in }
                                        { picture(s)}
   polyCount:        LongInt;           {number of polygons in picture(s)}
   regionCount:      LongInt;           {number of regions in picture(s)}
   bitMapCount:      LongInt;           {number of bitmaps}
   pixMapCount:      LongInt;           {number of pixel maps}
   commentCount:     LongInt;           {number of comments in picture(s)}
   uniqueComments:   LongInt;           {number of different comments }
                                        { (by ID) in picture(s)}
   commentHandle:    CommentSpecHandle;
                                        {handle to array of CommentSpec }
                                        { records for picture(s)}
   uniqueFonts:      LongInt;           {number of fonts in picture(s)}
   fontHandle:       FontSpecHandle;    {handle to an array of FontSpec }
                                        { records for picture(s)}
```

```
    fontNamesHandle:  Handle;          {handle to list of font names for }
                                       { picture(s)}
    reserved1:        LongInt;
    reserved2:        LongInt;
  END;
```

Routines

Creating and Disposing of Pictures

```
FUNCTION OpenCPicture       (newHeader: OpenCPicParams): PicHandle;
FUNCTION OpenPicture        (picFrame: Rect): PicHandle;
PROCEDURE PicComment        (kind,dataSize: Integer; dataHandle: Handle);
PROCEDURE ClosePicture;
PROCEDURE KillPicture       (myPicture: PicHandle);
```

Drawing Pictures

```
PROCEDURE DrawPicture       (myPicture: PicHandle; dstRect: Rect);
FUNCTION GetPicture         (picID: Integer): PicHandle;
```

Collecting Picture Information

```
/* DisposePictInfo is also spelled as DisposPictInfo */
FUNCTION GetPictInfo        (thePictHandle: PicHandle;
                             VAR thePictInfo: PictInfo; verb: Integer;
                             colorsRequested: Integer;
                             colorPickMethod: Integer;
                             version: Integer): OSErr;
FUNCTION GetPixMapInfo      (thePixMapHandle: PixMapHandle;
                             VAR thePictInfo: PictInfo; verb: Integer;
                             colorsRequested: Integer;
                             colorPickMethod: Integer;
                             version: Integer): OSErr;
FUNCTION NewPictInfo        (VAR thePictInfoID: PictInfoID; verb: Integer;
                             colorsRequested: Integer;
                             colorPickMethod: Integer;
                             version: Integer): OSErr;
FUNCTION RecordPictInfo     (thePictInfoID: PictInfoID;
                             thePictHandle: PicHandle): OSErr;
```

```
FUNCTION RecordPixMapInfo   (thePictInfoID: PictInfoID;
                             thePixMapHandle: PixMapHandle): OSErr;
FUNCTION RetrievePictInfo   (thePictInfoID: PictInfoID;
                             VAR thePictInfo: PictInfo;
                             colorsRequested: Integer): OSErr;
FUNCTION DisposePictInfo    (thePictInfoID: PictInfoID): OSErr;
```

Application-Defined Routines

```
FUNCTION MyInitPickMethod   (colorsRequested: Integer;
                             VAR dataRef: LongInt;
                             VAR colorBankType: Integer): OSErr;
FUNCTION MyRecordColors     (dataRef: LongInt; colorsArray: RGBColorArray;
                             colorCount: LongInt;
                             VAR uniqueColors: LongInt): OSErr;
FUNCTION MyCalcColorTable   (dataRef: LongInt; colorsRequested: Integer;
                             colorBankPtr: Ptr;
                             VAR resultPtr: CSpecArray): OSErr;
FUNCTION MyDisposeColorPickMethod
                            (dataRef: LongInt): OSErr;
```

C Summary

Constants

```
/* color-picking methods */
#define systemMethod    0  /* let Picture Utilities choose the method
                              (currently they always choose popularMethod) */
#define popularMethod   1  /* return most frequently used colors */
#define medianMethod    2  /* return a weighted distribution of colors */

/* picture information to be returned by Picture Utilities */
#define returnColorTable   ((short) 0x0001)  /* return a ColorTable record */
#define returnPalette      ((short) 0x0002)  /* return a Palette record */
#define recordComments     ((short) 0x0004)  /* return comment information */
#define recordFontInfo     ((short) 0x0008)  /* return font information */
#define suppressBlackAndWhite
                           ((short) 0x0010)  /* don't include black and
                                                white with returned colors */
```

```
/* color bank types */
#define ColorBankIsCustom        -1    /* gathers colors into a custom
                                          color bank */
#define ColorBankIsExactAnd555   0     /* gathers exact colors if there are
                                          less than 256 unique colors in
                                          picture; otherwise gathers colors
                                          for picture in a 5-5-5 histogram */
#define ColorBankIs555           1     /* gathers colors in a 5-5-5
                                          histogram */
```

Data Types

```
struct Picture {
   short    picSize;    /* for a version 1 picture: its size */
   Rect     picFrame;   /* bounding rectangle for the picture */
   /* variable amount of picture data in the form of opcodes */
};
typedef struct Picture Picture;
typedef Picture *PicPtr, **PicHandle;

struct OpenCPicParams {
   Rect  srcRect;    /* optimal bounding rectangle for displaying picture at
                        resolution indicated in hRes, vRes fields */
   Fixed hRes;       /* best horizontal resolution; $00480000 specifies
                        72 dpi */
   Fixed vRes;       /* best vertical resolution; $00480000 specifies
                        72 dpi */
   short version;    /* set to -2 */
   short reserved1;  /* reserved; set to 0 */
   long  reserved2;  /* reserved; set to 0 */
};

struct CommentSpec {
   short    count;   /* number of times this type of comment occurs in
                        the picture or survey */
   short    ID;      /* value identifying this type of comment */
};

typedef struct CommentSpec CommentSpec;
typedef CommentSpec *CommentSpecPtr, **CommentSpecHandle;
```

```
struct FontSpec {       /* font specification record */
   short    pictFontID; /* font ID as stored in the picture */
   short    sysFontID;  /* font family ID */
   long     size[4];    /* each bit set from 1 to 127 indicates a point
                           size at that value; if bit 0 is set, then a size
                           larger than 127 is found */
   short    style;      /* styles used for this font family */
   long     nameOffset; /* offset to font name stored in the data structure
                           indicated by the fontNamesHandle field of the
                           PictInfo record */
};
typedef struct FontSpec FontSpec;
typedef FontSpec *FontSpecPtr, **FontSpecHandle;

struct  PictInfo {
   short             version;       /* Picture Utilities version number */
   long              uniqueColors;  /* total colors in survey */
   PaletteHandle     thePalette;    /* handle to a Palette record--NIL for
                                       a bitmap in a basic graphics port */
   CTabHandle        theColorTable; /* handle to a ColorTable record--NIL for
                                       a bitmap in a basic graphics port */
   Fixed             hRes;          /* best horizontal resolution (dpi)} */
   Fixed             vRes;          /* best vertical resolution (dpi) */
   short             depth;         /* greatest pixel depth */
   Rect              sourceRect;    /* optimal bounding rectangle for
                                       picture for display at resolution
                                       specified in hRes and vRes fields */
   long              textCount;     /* number of text strings in
                                       picture(s) */
   long              lineCount;     /* number of lines in picture(s) */
   long              rectCount;     /* number of rectangles in picture(s) */
   long              rRectCount;    /* number of rounded rectangles in
                                       picture(s) */
   long              ovalCount;     /* number of ovals in picture(s) */
   long              arcCount;      /* number of arcs and wedges in
                                       picture(s) */
   long              polyCount;     /* number of polygons in picture(s) */
   long              regionCount;   /* number of regions in picture(s) */
   long              bitMapCount;   /* number of bitmaps */
   long              pixMapCount;   /* number of pixel maps */
   long              commentCount;  /* number of comments in picture(s) */
   long              uniqueComments;
                                    /* number of different comments (by ID)
                                       in picture(s) */
```

```
   CommentSpecHandle commentHandle; /* handle to an array of CommentSpec
                                       structures for picture(s) */
   long              uniqueFonts;   /* number of fonts in picture(s) */
   FontSpecHandle    fontHandle;    /* handle to an array of FontSpec
                                       structures for picture(s) */
   Handle            fontNamesHandle;
                                    /* handle to list of font names for
                                       picture(s) */
   long              reserved1;
   long              reserved2;
};
typedef struct PictInfo PictInfo;
typedef PictInfo *PictInfoPtr,**PictInfoHandle;

typedef long PictInfoID;
```

Functions

Creating and Disposing of Pictures

```
pascal PicHandle OpenCPicture
                           (const OpenCPicParams *newHeader);
pascal PicHandle OpenPicture
                           (const Rect *picFrame);
pascal void PicComment     (short kind, short dataSize, Handle dataHandle);
pascal void ClosePicture   (void);
pascal void KillPicture    (PicHandle myPicture);
```

Drawing Pictures

```
pascal void DrawPicture    (PicHandle myPicture, const Rect *dstRect);
pascal PicHandle GetPicture (Integer picID);
```

Collecting Picture Information

```
pascal OSErr GetPictInfo    (PicHandle thePictHandle,
                             PictInfo *thePictInfo, short verb,
                             short colorsRequested, short colorPickMethod,
                             short version);
pascal OSErr GetPixMapInfo  (PixMapHandle thePixMapHandle,
                             pictInfo *thePictInfo, short verb,
                             short colorsRequested, short colorPickMethod,
                             short version);
pascal OSErr NewPictInfo    (PictInfoID *thePictInfoID, short verb,
                             short colorsRequested, short colorPickMethod,
                             short version);
pascal OSErr RecordPictInfo (PictInfoID thePictInfoID,
                             PicHandle thePictHandle);
pascal OSErr RecordPixMapInfo
                            (PictInfoID thePictInfoID,
                             PixMapHandle thePixMapHandle);
pascal OSErr RetrievePictInfo
                            (PictInfoID thePictInfoID,
                             PictInfo *thePictInfo, short colorsRequested);
pascal OSErr DisposePictInfo
                            (PictInfoID thePictInfoID);
```

Application-Defined Functions

```
pascal OSErr MyInitPickMethod
                            (short colorsRequested, long *dataRef,
                             short *colorBankType);
pascal OSErr MyRecordColors (long dataRef, RGBColorArray colorsArray,
                             long colorCount, long *uniqueColors);
pascal OSErr MyCalcColorTable
                            (long dataRef, short colorsRequested,
                             Ptr colorBankPtr, CSpecArray *resultPtr);
pascal OSErr MyDisposeColorPickMethod
                            (long dataRef);
```

Assembly-Language Summary

Data Structures

Picture Data Structure

0	`picSize`	word	for a version 1 picture: its size
2	`picFrame`	8 bytes	bounding rectangle for the picture
10	`picData`	variable	variable amount of picture data

OpenCPicParams Data Structure

0	`srcRect`	8 bytes	optimal bounding rectangle for displaying picture at `hRes`, `vRes`
8	`hRes`	long	best horizontal resolution
12	`vRes`	long	best vertical resolution
16	`version`	word	always set to –2
18	`reserved1`	word	reserved; set to 0
20	`reserved2`	long	reserved; set to 0

CommentSpec Data Structure

0	`count`	long	number of times this type of comment occurs in picture or survey
4	`ID`	long	value identifying this type of comment

FontSpec Data Structure

0	`pictFontID`	word	font ID as stored in the picture
2	`sysFontID`	word	font family ID
4	`size`	8 bytes	each bit set from 1 to 127 indicates a point size at that value; if bit 0 is set, then a size larger than 127 points is found
12	`style`	word	styles used for this font family
14	`nameOffset`	long	offset to font name stored in the data structure indicated by the `fontNamesHandle` field of the `PictInfo` record

PictInfo Data Structure

0	version	word	Picture Utilities version number
2	uniqueColors	long	total number of colors in survey
6	thePalette	long	handle to a Palette record—NIL for a bitmap in a basic graphics port
10	theColorTable	long	handle to a ColorTable record—NIL for a bitmap in a basic graphics port
14	hRes	long	best horizontal resolution (dpi)
18	vRes	long	best vertical resolution (dpi)
22	depth	word	greatest pixel depth
24	sourceRect	8 bytes	optimal bounding rectangle for picture for display at resolution specified in hRes and vRes fields
32	textCount	long	number of text strings in picture(s)
36	lineCount	long	number of lines in picture(s)
40	rectCount	long	number of rectangles in picture(s)
44	rRectCount	long	number of rounded rectangles in picture(s)
48	ovalCount	long	number of ovals in picture(s)
52	arcCount	long	number of arcs and wedges in picture(s)
56	polyCount	long	number of polygons in picture(s)
60	regionCount	long	number of regions in picture(s)
64	bitMapCount	long	number of bitmaps
68	pixMapCount	long	number of pixel maps
72	commentCount	long	number of comments in picture(s)
76	uniqueComments	long	number of different comments (by ID) in picture(s)
80	commentHandle	long	handle to an array of CommentSpec records for picture(s)
84	uniqueFonts	long	number of fonts in picture(s)
88	fontHandle	long	handle to an array of FontSpec records for picture(s)
92	fontNamesHandle	long	handle to list of font names for picture(s)
96	reserved1	long	reserved
100	reserved2	long	reserved

Trap Macros

Trap Macros Requiring Routine Selectors

_Pack15

Selector	Routine
$0206	DisposePictInfo
$0403	RecordPictInfo
$0404	RecordPixMapInfo
$0505	RetrievePictInfo
$0602	NewPictInfo
$0800	GetPictInfo
$0801	GetPixMapInfo

Result Codes

pictInfoVersionErr	–11000	Version number not 0
pictInfoIDErr	–11001	Invalid picture information ID
pictInfoVerbErr	–11002	Invalid verb combination specified
cantLoadPickMethodErr	–11003	Custom pick method not in resource chain
colorsRequestedErr	–11004	Number out of range or greater than that passed to NewPictInfo
pictureDataErr	–11005	Invalid picture data