NFF

NAME:	NFF
ALSO KNOWN AS:	Haines NFF, Neutral File Format
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Eric Haines
PLATFORM:	All
SUPPORTING APPLICATIONS:	Standard Procedural Database (SPD), MTV, others
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	MTV, Pixar RIB, POV, PRT, QRT, Radiance, Rayshade, RTrace

USAGE: Modeling of rendering algorithms, ray-trace applications.

COMMENTS: A simple scene description language incorporating most of the basics, which would be informative for anyone thinking about designing yet another ray-trace scene-description format.

Overview

NFF (Neutral File Format) is the creation of Eric Haines. Mr. Haines, the publisher of *Ray Tracing News*, has been active in the high-end graphics community for a number of years, He is well-known through these efforts, particularly on the Internet. As a consequence of Mr. Haines' visibility, although NFF was originally designed to test rendering algorithms, the format has played a role in the evolution of other, more sophisticated, scene description languages and formats.

Eric Haines describes NFF as follows:

> The NFF (Neutral File Format) is designed as a minimal scene description language. The language was designed in order to test various

Ex.1007 / Page 648 of 1158
APPLE INC.

rendering algorithms and efficiency schemes. It is meant to describe the geometry and basic surface characteristics of objects, the placement of lights, and the viewing frustum for the eye. Some additional information is provided for aesthetic reasons (such as the color of the objects, which is not strictly necessary for testing the efficiency of rendering algorithms).

Note that NFF has minimal support for lighting and shading.

File Organization

NFF files consist of lines of ASCII text. Each line describes an object called an entity. The first field of each line describes the entity's type, and subsequent fields on the same line, and possibly subsequent lines, contain further information on the entity.

File Details

The information in this section is extracted from the NFF documentation kindly provided by Eric Haines.

By providing a minimal interface, NFF is meant to act as a simple format to allow the programmer to quickly write filters to move from NFF to the local file format. Presently, the following entities are supported:

- Simple perspective frustum
- Background color description
- Positional (versus directional) light-source description
- Surface-properties description
- Polygon, polygonal patch, cylinder/cone, and sphere descriptions

Entities are coded as follows:

"v"	Viewpoint location (viewing vectors and angles)
"b"	Background color
"l"	Positional light location
"f"	Object material properties
"c"	Cone or cylinder primitive
"s"	Sphere primitive
"p"	Polygon primitive
"pp"	Polygonal patch primitive

Ex.1007 / Page 649 of 1158
APPLE INC.

These are explained in the following sections.

Viewpoint Location

The viewpoint location entity is coded as follows:

"v"
"from> Fx Fy Fz
"at" Ax Ay Az
"up" Ux Uy Uz
"angle" angle
"hither" hither
"resolution" xres yres

Format:

```
v
from %g %g %g
at %g %g %g
up %g %g %g
angle %g
hither %g
resolution %d %d
```

Parameters

from	Eye location in XYZ
at	Position to be at the center of the image, in XYZ world coordinates (a.k.a. "lookat")
up	Vector defining which direction is up, as an XYZ vector
angle	In degrees, defined as the angle from the center of top pixel row to bottom pixel row and left column to right column
hither	Distance of the hither plane (if any) from the eye. Mostly needed for hidden surface algorithms.
resolution	In pixels, in x and in y

Note that no assumptions are made about normalizing the data (e.g., the from-at distance does not have to be 1). Also, vectors are not required to be perpendicular to each other.

Ex.1007 / Page 650 of 1158
APPLE INC.

For all databases, some viewing parameters are always the same:

yon is "at infinity."
aspect ratio is 1.0.

A view entity must be defined before any objects are defined. (This requirement is so that NFF files can be displayed on-the-fly by hidden-surface machines.)

Background Color

A color is simply RGB, with values between 0 and 1:

"b" R G B

Format:

```
b %g %g %g
```

If no background color is set, assume that RGB = [0,0,0].

Positional Light Location

A light is defined by XYZ position:

"l" X Y Z [R G B]

Format:

```
l %g %g %g [%g %g %g]
```

All light entities must be defined before any objects are defined. (This requirement is so that NFF files can be used by hidden surface machines). Lights have a non-zero intensity of no particular value, if not specified (i.e., the program can determine a useful intensity as desired); the red/green/blue color of the light can optionally be specified.

Object Material Properties (Fill Color and Shading Parameters)

Object material properties (fill color and shading parameters) are coded as follows:

"f" red green blue Kd Ks Shine T index_of_refraction

Format:

```
f %g %g %g %g %g %g %g %g
```

RGB is in terms of 0.0 to 1.0.

Ex.1007 / Page 651 of 1158
APPLE INC.

Parameters

Kd	Diffuse component
Ks	Specular
Shine	Phong cosine power for highlights
T	Transmittance (fraction of contribution of the transmitting ray).

Usually, 0 <= Kd <= 1 and 0 <= Ks <= 1, though it is not required that Kd + Ks = 1. Note that transmitting objects (T > 0) are considered to have two sides for algorithms that need these (normally, objects have one side).

The fill color is used to color the objects following it until a new color is assigned.

Objects (Cone or Cylinder Primitive)

All objects are considered one-sided, unless the second side is needed for transmittance calculations (e.g., you cannot throw out the second intersection of a transparent sphere in ray tracing).

A cylinder is defined as having a radius and an axis defined by two points, which also define the top and bottom edge of the cylinder.

A cone is defined in similar fashion; the difference is that the apex and base radii are different. The apex radius is defined as being smaller than the base radius. Note that the surface exists without endcaps. The cone or cylinder description is shown below:

```
"c"
base.x base.y base.z base_radius
apex.x apex.y apex.z apex_radius
```

Format:

```
c
%g %g %g %g
%g %g %g %g
```

A negative value for both radii means that only the inside of the object is visible (objects are normally considered one-sided, with the outside visible). Note that the base and apex cannot be coincident for a cylinder or cone. Making them coincident could be used to define endcaps, but none of the SPD scenes currently make use of this definition.

Ex.1007 / Page 652 of 1158
APPLE INC.

Sphere

A sphere is defined by a radius and center position, as shown below:

"s" center.x center.y center.z radius

Format:

```
s %g %g %g %g
```

If the radius is negative, then only the sphere's inside is visible (objects are normally considered one-sided, with the outside visible). Currently none of the SPD scenes makes use of negative radii.

Polygon

A polygon is defined by a set of vertices. With these databases, a polygon is defined to have all points coplanar. A polygon has only one side; the order of the vertices is counterclockwise as you face the polygon (right-handed coordinate system). The first two edges must form a non-zero convex angle, so that the normal and side visibility can be determined by using just the first three vertices.

A polygon is defined as shown below:

"p" total_vertices
vert1.x vert1.y vert1.z
[etc. for total_vertices vertices]

Format:

```
p %d
[ %g %g %g ] ← for total_vertices vertices
```

Polygonal Patch

A patch is defined by a set of vertices and their normals. With these databases, a patch is defined to have all points coplanar. A patch has only one side, with the order of the vertices being counterclockwise as you face the patch (right-handed coordinate system). The first two edges must form a non-zero convex angle, so that the normal and side visibility can be determined.

A polygonal patch is defined as shown below:

"pp" total_vertices
vert1.x vert1.y vert1.z norm1.x norm1.y norm1.z
[etc. for total_vertices vertices]

Ex.1007 / Page 653 of 1158
APPLE INC.

Format:

```
pp %d
[ %g %g %g %g %g %g ] ← for total_vertices vertices
```

Comment

A comment is defined as shown below:

"#" [string]

Format:

```
# [ string ]
```

As soon as a # character is detected, the rest of the line is considered a comment.

For Further Information

For further information about the NFF format, see the specification included on the CD-ROM that accompanies this book. You can also contact the NFF author:

Eric Haines
3D/EYE Inc.
1050 Craft Road
Ithaca, NY 14850
Email: *erich@eye.com*

NFF is also used in conjunction with the Standard Procedural Database (SPD) software, a package designed to create a variety of databases for testing rendering schemes. For more information about SPD, see the following paper:

"A Proposal for Standard Graphics Environments," *IEEE Computer Graphics and Applications*, vol. 7, no. 11, November 1987, pp. 3–5.

SPD is available by anonymous FTP from:

ftp://wuarchive.wustl.edu/graphics/graphics/objects/
ftp://princeton.edu/pub/Graphics/

Images of the databases are available from (among other places):

ftp://ftp.eye.com/pub/graphics/SPD/
ftp://gondwana.ecr.mu.oz.au/pub/images/haines/

Ex.1007 / Page 654 of 1158
APPLE INC.

NAME:	OFF
ALSO KNOWN AS:	Object File Format
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Randi Rost
PLATFORM:	UNIX
SUPPORTING APPLICATIONS:	dxmodel, others
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes
IMAGES ON CD:	No
SEE ALSO:	MTV, Pixar RIB, POV, PRT, QRT, Radiance, RTrace

USAGE: Description of 3D scenes for later rendering.

COMMENTS: The OFF format is designed to support easy and flexible description of 3D objects for later manipulation and rendering.

Overview

OFF (Object File Format) was developed in 1986 at Digital Equipment Corporation's Workstations Systems Engineering by Randi Rost and was subsequently made available for public distribution. OFF is partly derived from an object file format used at Ohio State University. OFF was designed from the start to support data interchange and archiving; in this case, the interchange and archiving of 3D objects. Although this plan originally bore fruit inside Digital, OFF has seen use in the 3D modeling community, partly because of its wide availability on the Internet.

The OFF author thought carefully about how to establish libraries of laboriously-produced 3D objects so that the labor that went into construction of the objects could be amortized through reuse. OFF files consist of lines of ASCII

Ex.1007 / Page 655 of 1158
APPLE INC.

text describing objects, implementing part of the author's goal of making OFF independent of language, device, and operating system.

The format is well described in the original specification documents included on the CD-ROM, and only a summary of that format is included here.

File Organization and Details

An OFF file consists of a number of ASCII lines. The following are usually found at the beginning of these lines:

Name — Short descriptive name of object defined in the file

Description — Fuller description of the object defined in the file

Author — Actual author or company owner

Copyright — Distribution information

Type — Object type; currently, only polygons are supported

Following this information is a series of lines, each defining an object attribute. Each attribute consists of:

Property name
: Uniquely describes the property; currently, conventions exist for geometry, polygon colors, vertex colors, back faces, vertex order, diffusion coefficients, specular coefficients, and specular power

Property types
: One of the following: default, generic, indexed, or indexed_poly

Data format
: String of characters indicating the order and type of the data to follow

Filename or data
: The file indicated here may contain more elaborate data than might be appropriate in this file.

Associated with the original OFF system are an include file, *objects.h,* and a library file, *off.a* (on UNIX systems). Together, these implement a subroutine library for reading and writing OFF files. You can adapt these for operating systems other than UNIX.

Ex.1007 / Page 656 of 1158
APPLE INC.

For Further Information

For further information about the OFF format, see the specification included on the CD-ROM that accompanies this book.

> Rost, Randi, *OFF—A 3D Object File Format*, 6 November 1986, updated 12 October 1989.

You can also obtain the OFF archive (containing the distribution format, tools, and objects) via FTP from:

> *ftp://gatekeeper.dec.com/pub/DEC/*

The OFF author, Randi Rost, is not currently supporting OFF or enhancing its tools (nor is Digital Equipment, where Mr. Rost developed OFF). The archive can be used freely, but comes with no express or implied warranties. You must adhere to the usage guidelines outlined in the copyright sections of the individual files.

For further information, contact:

Randi Rost
Kubota Pacific Computer, Inc.
2630 Walsh Avenue
Santa Clara, CA 95051
Email: *rost@kpc.com*

Ex.1007 / Page 657 of 1158
APPLE INC.

OS/2 Bitmap

NAME:	OS/2 Bitmap*
ALSO KNOWN AS:	BMP, Presentation Manager Bitmap, PM Bitmap, PM BMP, PM DIB
TYPE:	Bitmap
COLORS:	Mono, 4-bit, 8-bit, 24-bit
COMPRESSION:	RLE, Huffman 1D, uncompressed
MAXIMUM IMAGE SIZE:	64K×64K and 4G×4G pixels
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	Microsoft Corporation, IBM
PLATFORM:	Intel machines running OS/2, Microsoft Windows, Windows NT, Windows 95, and MS-DOS
SUPPORTING APPLICATIONS:	Too numerous to list
SPECIFICATION ON CD:	Yes (part of the Presentation Manager Metafile specification)
CODE ON CD:	Yes
IMAGES ON CD:	Yes
SEE ALSO:	Microsoft Windows Bitmap

USAGE: Used as the standard bitmap, pointer, and icon storage format in the IBM OS/2 Presentation Manager environment.

COMMENTS: A well-defined format for programmers having access to the IBM Developer's Toolkit for OS/2 Warp and Presentation Manager Software Development Kit. Its simple RLE compression scheme is rather inefficient for complex, deep-pixel images. Its many variations and differences from the Windows BMP format can be confusing. It is also less widely used than the Windows BMP file format.

* Our thanks to David Charlap for his contributions to this article.

Ex.1007 / Page 658 of 1158
APPLE INC.

Overview

The IBM OS/2 Bitmap (BMP) file format is one of several graphics file formats supported by the OS/2 operating system. BMP is the native bitmap format of OS/2 and is used to store several types of bitmap data, including icons and pointers. Most graphics and imaging applications operating under OS/2 support the creation and display of BMP format files. BMP is also found in MS-DOS and Microsoft Windows and originated in that environment.

BMP was originally developed for the Microsoft Windows environment. Microsoft shared responsibility with IBM for the development of early versions of IBM's OS/2 operating system. When Presentation Manager, the OS/2 Graphical User Interface, required a bitmap format, the Windows BMP format was used. Thus, the Windows 2.*x* and OS/2 1.*x* BMP formats are identical.

The structure of BMP format files is closely tied to the API of both OS/2 and Windows. In this regard, BMP was never meant to be a portable format, or used for bitmap data interchange between different operating systems. As each of these operating system APIs has changed, the BMP format has changed along with them.

OS/2 BMP files are much less common than Windows BMP files. This is mostly due to the wider distribution and acceptance of the Microsoft Windows environment over the IBM OS/2 operating system, as well as to the greater number of graphics-using Windows applications.

Because OS/2 BMP files are encountered less frequently, many graphics file display programs do not support the reading, writing, and displaying of OS/2 BMP files but do support Microsoft BMP. However, because both of these formats use the same file extension, a non-OS/2-aware file reader will not be able to discern whether it is reading a Windows or an OS/2 BMP file. Reading an OS/2 BMP file as if it were a Windows BMP file will produce unpredictable results from the software application.

There are currently two versions of BMP under OS/2 (1.*x* and 2.*x*) and six variations; two with the file extension .ICO, two with .PTR, and two using .BMP. This article describes the two versions and their variations used under IBM OS/2. For a discussion of the Microsoft Windows BMP format versions, see the article about the Microsoft Bitmap format.

All of the BMP versions originated on Intel-based machines and thus share a common little-endian heritage. The current BMP format is otherwise hardware-independent and can accommodate images with up to 24-bit color. Its basic

Ex.1007 / Page 659 of 1158
APPLE INC.

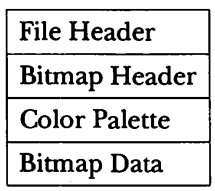
design makes it a good general purpose format that can be used for color or black-and-white image storage if file size is not a factor. Its main virtues are its simplicity and widespread support in the PC marketplace.

The compression method used is a type of run-length encoding (RLE), although most BMP files to date have been stored uncompressed. Although the BMP RLE scheme is lossless and quickly decompressed, it is not considered a superior compression method.

The OS/2 BMP format is documented in the appendix of the OS/2 Presentation Manager *Programmer's Reference* manual and in the online Presentation Manager reference from the OS/2 Warp v3 toolkit.

File Organization

OS/2 BMP files occur in two major revisions (v1.*x* and v2.*x*) and in six different variants: Bitmap, Bitmap Array, Icon, Color Icon, Pointer, and Color Pointer. The typical BMP file contains four sections of data: the file header, the bitmap header, the color palette, and the bitmap data.



As previously mentioned, OS/2 1.*x* BMP files are identical to Microsoft Windows 2.*x* BMP files. This fact reflects the common ancestry of the OS/2 operating system and the Microsoft Windows operating environment.

The major difference between v1.*x* and v2.*x* files is the size of the bitmap header. This header is 14 bytes in length in v1.*x* BMP files and is 64 bytes in length in v2.*x* files.

File Details

All OS/2 BMP files begin with a file header 14 bytes in length with the following format:

```
typedef struct _Os2BmpFileHeader
{
    WORD   FileType;      /* File type identifier */
```


Ex.1007 / Page 660 of 1158
APPLE INC.

```
    DWORD  FileSize;       /* Size of the file in bytes */
    WORD   XHotSpot;       /* X coordinate of hotspot */
    WORD   YHotSpot;       /* Y coordinate of hotspot */
    DWORD  BitmapOffset;   /* Starting position of image data in bytes */
} OS2BMPFILEHEADER;
```

FileType holds a 2-byte magic value used to identify the file type. This value is 4D42h (or "BM" in ASCII) for a single bitmap, 4142h ("BA") for a bitmap array, 4349h ("IC") for an icon, 4943h ("CI") for a color icon, 5450h ("PT") for a pointer, and 5043h ("CP") for a color pointer. If your application only uses "BM" type BMP files, make sure to always check this field before attempting to use any of the data read from the file.

Size is the combined size of the file header plus the bitmap header in bytes. This value is typically zero in many BMP files.

XHotSpot and YHotSpot store the coordinates of the central point of the hotspot on the bitmap for icons and pointers. These coordinates are relative to the lower-left corner of the bitmap. If there is no hotspot, then these values will be zero. These values are not used in bitmap and bitmap array BMP files.

BitmapOffset is the offset, in bytes, from the beginning of the file to the pixel data that corresponds to this header.

BMP Version 1.x (IBM OS/2 1.x)

Following the file header in v1.*x* BMP files is a second header called the bitmap header. This header contains information specific to the bitmap data. This header is 12 bytes in length and has the following format:

```
typedef struct _Os21xBitmapHeader
{
    DWORD Size;            /* Size of this header in bytes */
    WORD  Width;           /* Image width in pixels */
    WORD  Height;          /* Image height in pixels */
    WORD  NumPlanes;       /* Number of color planes */
    WORD  BitsPerPixel;    /* Number of bits per pixel */
} OS21XBITMAPHEADER;
```

Size is the size of the header in bytes. For OS/2 1.*x* BMP files, this value is always 12.

Width and Height are the width and height of the image in pixels, respectively. Width does not include any scan-line boundary padding.

Ex.1007 / Page 661 of 1158
APPLE INC.

NumPlanes is the number of color planes used to represent the bitmap data. OS/2 BMP files contain only one color plane, so this value is always 1. The apparent size of a plane in bits is calculated by:

```
Width * Height * BitsPerPixel
```

The actual size of a plane includes scan-line padding.

BitsPerPixel is the number of bits per pixel in each plane. This value is in the range 1 to 24; the values 1, 4, 8, and 24 are the only values considered legal by the OS/2 1.*x* API.

Following the header is the color palette data. A color palette is always present in a BMP file if the bitmap data contains 1-, 4-, or 8-bit data. Twenty-four-bit bitmap data never uses a color palette (nor does it ever need to). Each element of the palette is three bytes in length and has the following structure:

```
typedef struct _Os21xPaletteElement
{
    BYTE Blue;      /* Blue component */
    BYTE Green;     /* Green component */
    BYTE Red;       /* Red component */
} OS21XPALETTEELEMENT;
```

Blue, Green, and Red hold the color component values for a pixel, each in the range 0 to 255.

The size of the color palette is calculated from the BitsPerPixel value. The color palette has 2, 16, 256, or 0 entries for a BitsPerPixel of 1, 4, 8, and 24, respectively. The number of color palette entries is calculated as follows:

```
NumberOfEntries = 1 << BitsPerPixel;
```

To detect the presence of a color palette in a BMP file (rather than just assuming that a color palette does exist), you can calculate the number of bytes between the bitmap header and the bitmap data and divide this number by the size of a single palette element. Assuming that your code is compiled using 1-byte structure element alignment, the calculation would be:

```
NumberOfEntries = (BitmapOffset - sizeof(OS2BMPFILEHEADER) -
    sizeof(OS21XBITMAPHEADER)) / sizeof(OS21XPALETTEELEMENT);'
```

If NumberOfEntries is zero, then there is no palette data, otherwise you have the number of elements in the color palette.

Ex.1007 / Page 662 of 1158
APPLE INC.

BMP Version 2.x (IBM OS/2 2.x)

Version 2.*x* BMP files begin with the same 14-byte header as v1.*x* BMP files. The file header is also followed by a bitmap header, which is an expanded version of the v1.*x* bitmap header. It is typically 64 bytes in size and contains up to 14 additional fields:

```
typedef struct _Os22xBitmapHeader
{
    DWORD  Size;              /* Size of this structure in bytes */
    DWORD  Width;             /* Bitmap width in pixels */
    DWORD  Height;            /* Bitmap height in pixel */
    WORD   NumPlanes;         /* Number of bit planes (color depth) */
    WORD   BitsPerPixel;      /* Number of bits per pixel per plane */

    /* Fields added for OS/2 2.x follow this line */

    DWORD  Compression;       /* Bitmap compression scheme */
    DWORD  ImageDataSize;     /* Size of bitmap data in bytes */
    DWORD  XResolution;       /* X resolution of display device */
    DWORD  YResolution;       /* Y resolution of display device */
    DWORD  ColorsUsed;        /* Number of color table indices used */
    DWORD  ColorsImportant;   /* Number of important color indices */
    WORD   Units;             /* Type of units used to measure resolution */
    WORD   Reserved;          /* Pad structure to 4-byte boundary */
    WORD   Recording;         /* Recording algorithm */
    WORD   Rendering;         /* Halftoning algorithm used */
    DWORD  Size1;             /* Reserved for halftoning algorithm use */
    DWORD  Size2;             /* Reserved for halftoning algorithm use */
    DWORD  ColorEncoding;     /* Color model used in bitmap */
    DWORD  Identifier;        /* Reserved for application use */

} OS22XBITMAPHEADER;
```

Size is the size of the header in bytes. This header can vary in size, so the reader must be careful only to read the number of bytes indicated by this value. Fields not included in the Size value are not stored in the BMP file, and their values are assumed to be zero. For example, if Size is 16, then only the first five fields (16 bytes) of the header are present. The color palette will begin starting on the 17th byte. If Size is 40, then only the first eleven fields are present. If this value is 64, then the entire header is present in the file. A value larger than 64 indicates a later version of the OS/2 BMP format.

Width and Height are the width and height of the image in pixels, respectively. Width does not include any scan-line boundary padding.

Ex.1007 / Page 663 of 1158
APPLE INC.

NumPlanes is the number of color planes used to represent the bitmap data. OS/2 BMP files contain only one color plane, so this value is always 1. The apparent size of a plane in bits is calculated by:

```
Width * Height * BitsPerPixel
```

The actual size of a plane includes scan-line padding.

BitsPerPixel is the number of bits per pixel in each plane. This value is in the range 1 to 24; the values 1, 4, 8, and 24 are the only values considered legal by the OS/2 2.*x* API.

Compression indicates the type of encoding method used to compress the bitmap data. 0 indicates that the data is uncompressed; 1 indicates that the 8-bit RLE algorithm was used; 2 indicates that the 4-bit RLE algorithm was used; 3 indicates that the Huffman 1D algorithm was used; and 4 indicates that the 24-bit RLE algorithm was used. (See the section called "Image Data and Compression" below for more information on BMP RLE encoding.)

ImageDataSize is the length of the pixel data in bytes, as stored in the file. This value may be zero for uncompressed bitmaps, in which case the bitmap size is calculated from:

```
Width * Height * BitsPerPixel
```

This value may never be zero for uncompressed bitmaps.

XResolution and YResolution are the horizontal and vertical resolutions of the bitmap. These values are used to help a BMP reader choose a proper resolution when printing or displaying a BMP file. The units used for the values in these fields are defined in the Units field.

ColorsUsed is the number of colors present in the palette. If this value is zero and the value of BitsPerPixel is less than 16, then the number of entries is equal to the maximum size possible for the color map. (BMP files with a BitsPerPixel value of 16 or greater will not have a color palette.) This value is calculated by using the value of the BitsPerPixel field:

```
ColorsUsed = 1 << BitsPerPixel;
```

ColorsImportant is the number of significant colors in the palette, determined by their frequency of appearance in the bitmap data; the more frequent the occurrence of a color, the more important it is. This field is used to provide as accurate a display as possible when using graphics hardware that supports fewer colors than defined in the image. For example, an 8-bit image with 142 colors might only have a dozen or so colors making up the bulk of the image.

Ex.1007 / Page 664 of 1158
APPLE INC.

If these colors could be identified, a display adapter with only 16-color capability would be able to display the image more accurately using the 16 most frequently occurring colors in the image. The most important colors are always stored first in the palette; ColorsImportant is 0 if all of the colors in the palette are to be considered important.

Units indicates the type of units used to interpret the values of the XResolution and YResolution fields. The only valid value is 0, indicating pixels per meter.

Reserved is unused and is always set to a value of zero.

Recording specifies how the bitmap scan lines are stored. The only valid value for this field is 0, indicating that the bitmap is stored from left to right and from the bottom up, with the origin being in the lower-left corner of the display.

Rendering specifies the halftoning algorithm used on the bitmap data. A value of 0 indicates that no halftoning algorithm was used; 1 indicates error diffusion halftoning; 2 indicates Processing Algorithm for Noncoded Document Acquisition (PANDA); and 3 indicates super-circle halftoning.

Size1 and Size2 are reserved fields used only by the halftoning algorithm. If error diffusion halftoning is used, Size1 is the error damping as a percentage in the range 0 through 100. A value of 100 percent indicates no damping, and a value of 0 percent indicates that any errors are not diffused. Size2 is not used by error diffusion halftoning. If PANDA or super-circle halftoning is specified, Size1 is the X dimension and Size2 is the Y dimension of the pattern used in pixels.

ColorEncoding specifies the color model used to describe the bitmap data. The only valid value is 0, indicating the RGB encoding scheme.

Identifier is a field reserved for application use and may contain an application-specific value.

The color palette that may follow the bitmap header is basically the same as the v1.*x* palette, but adds an extra byte of padding to increase its size to four bytes. This allows palette entries to be read as 4-byte values, making these values more efficient to read in memory and easier to see in a hex dump or debugger.

```
typedef struct _Os22xPaletteElement
{
    BYTE Blue;      /* Blue component */
    BYTE Green;     /* Green component */
```


Ex.1007 / Page 665 of 1158
APPLE INC.

```
    BYTE Red;       /* Red component */
    BYTE Reserved;  /* Padding (always 0) */
} OS22XPALETTEELEMENT;
```

Blue, Green, and Red hold the color component values for a pixel, each in the range 0 to 255.

Reserved pads the structure to end on an even-byte boundary and is always zero.

When identifying BMP files you must make sure that the first two bytes of the file are 4Dh 42h and only read the number of bytes from the bitmap header as indicated by the Size field. This value will be 12 for OS/2 1.*x* BMP files and less than or equal to 64 for OS/2 2.*x* BMP files. The file header is identical for both versions; the only difference in the bitmap header is that the Width and Height fields are WORDs in v1.*x* files and DWORDs in v2.*x* files.

Color Palette

As we have seen, a BMP color palette is an array of structures that specify the red, green, and blue intensity values of each color in a display device's color palette. Each pixel in the bitmap data stores a single value used as an index into the color palette. The color information stored in the element at that index specifies the color of that pixel.

One-, 4-, and 8-bit BMP files are expected to always contain a color palette. Twenty-four-bit BMP files never contain color palettes.

You must be sure to check the Size field of the bitmap header to know if you are reading 3-byte or 4-byte color palette elements. A Size value of 12 indicates an OS/2 1.*x* (or possibly a Windows 2.*x*) BMP file with 3-byte elements. Larger numbers, such as 64, indicate later versions of BMP, which all use 4-byte color palette elements.

OS/2 BMP File Types

Each new version of BMP has added new information to the bitmap header. In some cases, the newer versions have changed the size of the color palette and added features to the format itself. Unfortunately, a field wasn't included in the header to easily indicate the version of the file's format or the type of operating system that created the BMP file. If we add Windows' four versions of BMP to OS/2's two versions—each with four possible variations—we find that as many as twelve different related file formats all have the file extension ".BMP".

Ex.1007 / Page 666 of 1158
APPLE INC.

It is clear that you cannot know the internal format of a BMP file based on the file extension alone. But fortunately, you can use a short algorithm to determine the internal format of BMP files.

The FileType field of the file header is where we start. If these two byte values are 424Dh ("BM"), then you have a single-image BMP file that may have been created under Windows or OS/2. If FileType is the value 4142h ("BA"), 4349h ("IC"), 4943h ("CI"), 5450h ("PT"), or 5043h ("CP"), you have an OS/2 variation of the BMP format, the internals of which we discuss below.

If your file type is "BM", then you must now read the Size field of the bitmap header to determine the version of the file. Size is 12 for Windows 2.*x* BMP and OS/2 1.*x* BMP, 40 for Windows 3.*x* and Windows NT BMP, 12 to 64 for OS/2 2.*x* BMP, and 108 for Windows 4.*x* BMP. A Windows NT BMP file will always have a Compression value of 3; otherwise, it is read as a Windows 3.*x* BMP file.

Note that the only difference between Windows 2.*x* BMP and OS/2 1.*x* BMP is the data type of the Width and Height fields. For Windows 2.*x* they are signed shorts and for OS/2 1.*x* they are unsigned shorts. Windows 3.*x*, Windows NT, and OS/2 2.*x* BMP files only vary in the number of fields in the bitmap header and in the interpretation of the Compression field.

In addition to standard bitmaps, an OS/2 file supports the storage of icon, color icon, pointer (also called a *cursor*), color pointer, and bitmap array data. As we have noted, each of these BMP file types have a different image FileType value as follows:

Bitmap Array	4142h	'BA'
Bitmap	4D42h	'BM'
Color Icon	4943h	'CI'
Color Pointer	5043h	'CP'
Icon	4349h	'IC'
Pointer	5450h	'PT'

The icon, pointer, color icon, and color pointer BMP files have the same file and bitmap headers as bitmap BMP files, but differ in how the headers and bitmap data are arranged.

Icon and pointer BMP files store only 1-bit monochrome data and do not use color palette data, although a color palette may be present in the file. The bitmap data is normally uncompressed and stores two bitmap masks used to display the icon or pointer image.

Ex.1007 / Page 667 of 1158
APPLE INC.

The top half of the bitmap is the AND mask and the bottom half is the XOR mask (remember that the bitmap is stored from the bottom-up). The icon or pointer is displayed by first ANDing the top half of the bitmap with the pixels on the display and then XORing the bottom half of the bitmap with the same pixels. The four possible values produced from this masking will result in the background, foreground, transparent, and inverse values defined for the monochrome display.

Also note that because the two masks each occupy one-half of the total size of the bitmap, the icon or pointer displayed will be one-half the height specified in the Height field of the file header.

Display Pixel	AND Mask Bit	OR Mask Bit	Resulting Value
X	0	0	0 (background)
X	0	1	1 (foreground)
X	1	0	X (transparent)
X	1	1	~X (inverse)

Color icon and color pointer BMP files are basically the same as icon and pointer files but have an additional color bitmap added. The headers and color table for the monochrome bitmap are immediately followed by the headers and color palette for the color bitmap. The monochrome bitmap data and color bitmap data then follow, as shown in Figure OS/2 Bitmap-1.

Also note that this is the most common arrangement of these sections. The bitmap data can actually appear in any order as long the BitmapOffset field in the file headers point to the proper bitmap data. And the FileType field in other OS2BMPFILEHEADER sections will have the same value (either 'CI' or 'CP'). The NumPlanes and BitsPerPixel fields of the OS22XBITMAPHEADER header will indicate which is the monochrome and the color bitmap. For compatibility with existing software, the monochrome image data should always be written to the BMP file first.

Color icons and color pointers are displayed using the same process as their monochrome counterparts, with the color pixel values used to set the color of the icon or pointer pixels on the display.

Color icon and color pointers are typically based on the v2.*x* OS/2 BMP format, as the OS/2 32-bit icon editor only generates these file types, even when monochrome images are stored in them.

Ex.1007 / Page 668 of 1158
APPLE INC.

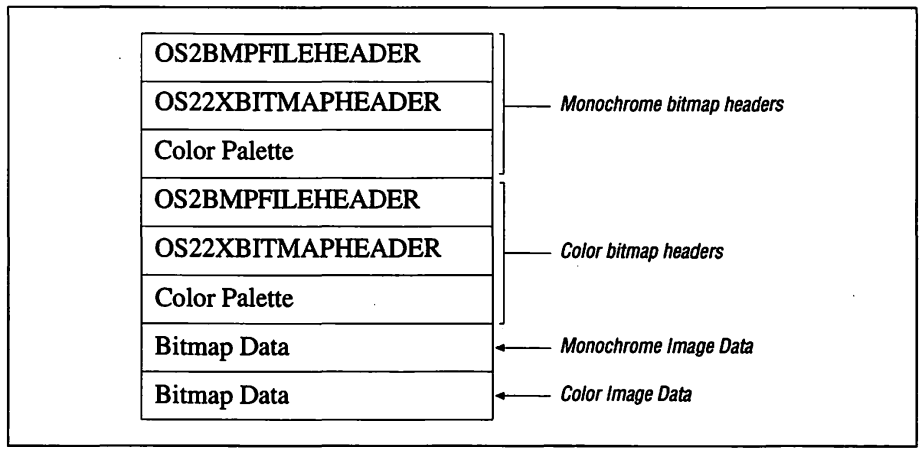


FIGURE OS/2 BITMAP-1: *Monochrome/color data format*

Display Pixel	AND Mask Bit	OR Mask Bit	Color Pixel		Resulting Value
X	0	0	C	C	(color)
X	0	1	C	C	(color)
X	1	0	C	X	(transparent)
X	1	1	C	~X	(inverse)

The bitmap array flavor of BMP allows the storage of one or more bitmaps within a single BMP file. Bitmap array files are typically used to hold multiple resolution bitmaps of the same image. This BMP variation is similar in concept and use to the Multiple Resolution Bitmap (.MRB) files found in the Microsoft Windows environment.

When multiple images are stored in a single OS/2 BMP file, it is assumed that each image has different metrics (dimensions, color depth, resolution) and that all the images are different renderings of the same object.

Bitmap array files start with a series of headers and color palettes. There is one set of headers for each bitmap stored in the file. The first header is a variation of the BMP file header, called a bitmap array header. This header is 14 bytes in size and has the following format:

Ex.1007 / Page 669 of 1158
APPLE INC.

```
typedef struct _Os2BmpArrayHeader
{
    WORD  Type;          /* Header type, always 4142h ("BA") */
    DWORD Size;          /* Size of this header */
    DWORD OffsetToNext;  /* Offset to next OS2BMPARRAYFILEHEADER */
    WORD  ScreenWidth;   /* Width of the image display in pixels */
    WORD  ScreenHeight;  /* Height of the image display in pixels */
} OS2BMPARRAYHEADER;
```

Type holds the 2-byte value 4142h, or 'BA' in ASCII. This value identifies the start of all bitmap array headers. File readers should not attempt to read 'BA' files as if they are single-bitmap ('BM') BMP files.

Size is the size of this bitmap array header and the headers that follow it. This value really has no practical use and is often set to zero. It is included in this header mainly for compatibility with other BMP headers starting with both Type and Size fields.

OffsetToNext contains the byte offset of the next OS2BMPARRAY-FILEHEADER structure in the file. This offset is calculated from the starting byte of the file. A value of zero indicates that this header is for the last image in the array list.

ScreenWidth and ScreenHeight indicate the resolution of the image in pixels. These values are used in conjunction with the Width, Height, NumPlanes, and BitsPerPixel values of the bitmap header to determine which bitmap in the file to display.

The headers that follow the bitmap array file header and the color palette data are the same used by the standard OS/2 BMP file format. Each image stored in the array bitmap file will have a complete set of these headers, as shown in Figure OS/2 Bitmap-2, and be linked together by the OffsetToNext value in the bitmap array header.

A display program should look through the list of OS2BMPARRAYHEADER structures in a bitmap array file and choose the image that most closely fits the resolution and metrics of the display device. If a proper image is not available, the first image in the list is displayed by default. Device-independent images should therefore always be placed first in any bitmap array list. The value of the Size field of the bitmap header will indicate if this is an OS/2 1.*x* or 2.*x* bitmap array file.

OS/2 attempts to find an exact match of the requested size (image dimensions). For icons and pointers, if a bitmap of the requested size cannot be

Ex.1007 / Page 670 of 1158
APPLE INC.

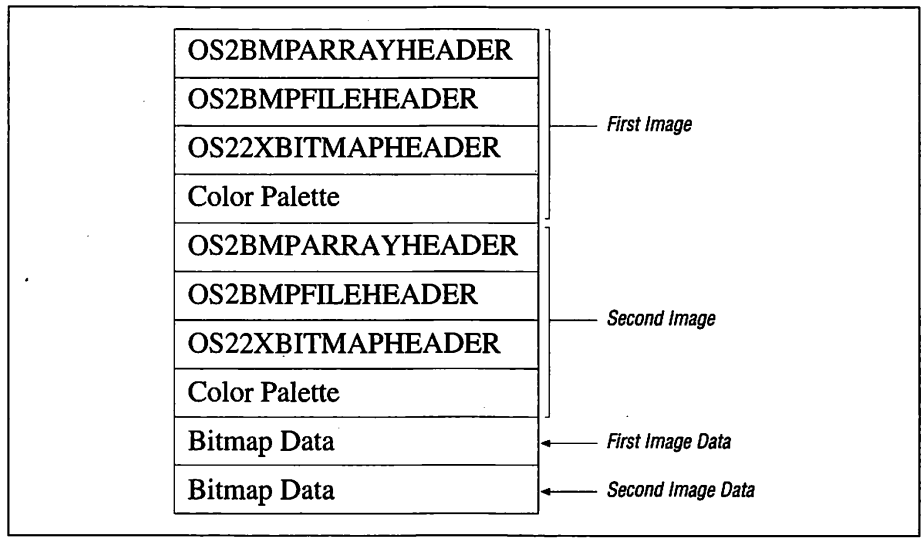


FIGURE OS/2 BITMAP-2: *Multi-image data format*

found, a device-independent icon or pointer bitmap is used. If the requested size cannot be found for standard bitmaps, a bitmap whose display size matches the output device's size is used, and a device-independent bitmap is used if neither is available.

If multiple bitmaps match the requested size (or if there are multiple matches for the screen size or if there are multiple device-independent images, etc.), then the image whose color depth most closely matches the output device's color depth is used.

The icon/pointer size that is requested depends on the display driver. The following versions are most commonly used:

40×40	4bpp	16 color
32×32	4bpp	16 color
32×32	1bpp	monochrome
20×20	1bpp	monochrome
16×16	1bpp	monochrome

The size 40×40 is used for 8514/a, XGA, and other displays with a resolution of 1024×768 or higher.

Ex.1007 / Page 671 of 1158
APPLE INC.

The size 32×32 is used for VGA, EGA, and most other devices of resolution less than 1024×768.

The sizes 20×20 and 16×16 are half-size variants of the 40×40 and 32×32 images. They are intended for small-icon views and use in window title bars.

If you ever encounter a user with a CGA (640×200) display, the icon size is 32×16 1bpp (monochrome).

Image Data and Compression

Uncompressed data is a series of values representing either color palette indices or actual RGB color values. Pixels are packed into bytes and arranged as scan lines. Each scan line must end on a 4-byte boundary, so one, two, or three bytes of padding may follow each scan line. Scan lines are always stored from the bottom up in OS/2 BMP files with the origin in the lower-left corner of the display.

Monochrome bitmaps contain one bit per pixel, eight pixels per byte (with the most significant bit being the leftmost pixel), and have a 2-element color palette. If a BMP reader chooses to ignore the color palette, all "one" bits are set to the display's foreground color and all "zero" bits are set to the background color.

Four-bit pixels are packed two per byte with the most significant nibble being the leftmost pixel. Eight-bit pixels are stored one per byte. Both 4- and 8-bit pixel values are indices into color palettes 16 and 256 elements in size respectively.

The OS/2 BMP format supports a simple run-length encoded (RLE) compression scheme for compressing 4-, 8-, and 24-bit bitmap data. One-bit monochrome data may be compressed using the Huffman 1D algorithm common to facsimile transmission applications. (For a discussion of the Huffman 1D algorithm, see Chapter 9, *Data Compression.*)

BMP uses a two-value RLE scheme. The first value contains a count of the number of pixels in the run, and the second value contains the value of the pixel. Runs of up to 255 identical pixel values may be encoded as only two bytes of data. Actually, it's a bit more complex than this. In addition to encoded runs, there are unencoded runs, delta markers, end-of-scan-line markers, and an end-of-RLE data marker.

Ex.1007 / Page 672 of 1158
APPLE INC.

The 8-bit RLE algorithm (RLE8) stores repeating pixel values as encoded runs. The first byte of an encoded run will be in the range of 1 to 255. The second byte is the value of the 8-bit pixels in the run. For example, an encoded run of 05 18 would decode into five pixels, each with the value 18, or 18 18 18 18 18.

When a scan line does not contain enough pixel runs to achieve a significant amount of compression, contiguous pixel values may be stored as literal or unencoded runs. An unencoded run may contain from 3 to 255 pixel values. The first byte of an unencoded run is always zero. This makes it possible to tell the difference between the start of an encoded and the start of an unencoded run. The second byte value is the number of unencoded pixel values that follow. If the number of pixels is odd, then a 00 padding value also follows. This padding value is not part of the original pixel data and should not be written to the decoded data stream. Here are some examples of encoded and unencoded data streams:

Encoded Bytes	**Decoded Bytes**
05 10	10 10 10 10 10
00 05 23 65 34 56 45 00	23 65 34 56 45
0A 0A	0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
00 04 46 57 68 79	46 57 68 79

Three marker values may also be found in the RLE data. Each of these markers also begins with a zero-byte value. The second byte value indicates the type of marker. These markers specify positional information relating to the decoded bitmap data and do not generate any data themselves.

The first marker is the end-of-scan-line marker and is identified by two byte values 00 and 00. This marker is an indication that the end of data for the current scan line has been reached. Encoded data occurring after this marker is decoded starting at the beginning of the next scan line. If an end-of-scan-line marker is not present in the encoded data, then the pixels will automatically wrap from the end of one scan line to the start of the next.

This marker is only used when you want to force the decoding of a scan line to end at a particular place. If the end-of-line marker occurs in the middle of a scan line, all remaining pixels in the decoded bitmap for the line are ignored. This "short scan line" technique is used to omit unneeded portions of scan lines. Most often, it is found in icon and pointer BMP files.

Ex.1007 / Page 673 of 1158
APPLE INC.

The next marker is the end of RLE data marker. It is identified by the two byte values 00 and 01. This marker occurs only as the last two bytes of the RLE data. This marker is an indication that the reader should stop decoding data.

The last marker is the run offset marker, also called a *delta* or *vector code.* This marker is four bytes in size, with the first two bytes being the values 00 and 02, and the last two values specifying a pixel address using unsigned X and Y values as an offset from the current bitmap cursor position. The X value is the number of pixels across the scan line, and the Y value is the number of rows forward in the bitmap.

This run offset marker indicates the location in the bitmap where the next decoded run of pixels should be written. For example, a run offset marker value of 00 02 05 03 would indicate that the offset of the bitmap cursor should move five pixels down the scan line, three rows forward, and write out the next run. The cursor then continues writing decoded data from its new position moving forwards.

Run offset markers are used when a bitmap may contain a large amount of "don't care" pixels. For example, if the BMP file holds a bitmap used as a mask (such as used with icons and pointers), many of the pixels in the rectangular bitmap may not be used. Rather than store these unused pixels in the BMP file, only the significant pixels are stored, and the delta markers are used as "jumps" to skip over the parts of the bitmap not actually used in the mask.

Here is another example. In color icon and color pointer BMP files, it is unnecessary to store color values for pixels whose AND mask value is 1, since the screen color will be determined by the XOR mask for those pixels and not by the color value. In this case, delta markers and end-of-scan-line markers would be used to skip over and ignore pixels not used in the image and therefore not stored in the bitmap.

The following are the BMP RLE markers:

00 00	End of scan line
00 01	End of bitmap data
00 02 XX YY	Run offset marker

Here is an example of decoding an 8-bit data stream. Each of the values is an 8-bit index value into the color palette, not an actual color value.

Ex.1007 / Page 674 of 1158
APPLE INC.

Encoded Bytes	Decoding Description	Decoded Bytes
04 16	Four bytes of value 16	16 16 16 16
08 45	Eight bytes of value 45	45 45 45 45 45 45 45 45
00 00	End of scan line	None
00 02 04 02	Move to offset four pixels forward and two rows up	None
03 E4	Three bytes of value E4	E4 E4 E4
00 03 12 A4 46 00	Three bytes of unencoded data	12 A4 46
00 00	End of scan line	None
00 01	End of RLE data	None

The 4-bit RLE algorithm (RLE4) stores repeating pixel values in a very similar manner to RLE8. All of the markers are the same. The only real difference is that two pixel values are packed per byte, and these pixel values alternate when decoded. For example, an RLE4-encoded data stream of 07 48 would decode to seven pixels, alternating in value as 04 08 04 08 04 08 04.

If this looks bizarre, it's because you rarely see alternating runs of pixel values in bitmaps eight bits or greater in depth. Four-bit (16-color) bitmaps, however, usually contain a lot of dithering. Most dithering algorithms will yield relatively large runs of alternating pixels. Runs of repeating sequences of three and four pixels are also fairly common output from many dithering algorithms. But the ability for efficiently encoding these types of pixel runs is not currently supported in the BMP RLE scheme.

Runs of identical pixel values may be encoded by RLE4 as well. For example, a run of twelve pixels, all of the value 9, would be RLE4-encoded as 0C 99 and would decode to the run 09 09 09 09 09 09 09 09 09 09 09 09.

Padding is added to unencoded pixel runs that are an odd number of bytes, rather than pixels, in length. And an unused final nibble in odd-length runs is set to zero. For example, the six pixel values 1 3 5 7 9 0 would be stored as the unencoded run 00 06 13 57 90 00, while the five pixel values 1 3 5 7 9 would be stored as the unencoded run 00 05 13 57 90 00.

Following is an example of decoding a 4-bit data stream. Each of the values is a 4-bit index value into the color palette, not an actual color value.

Ex.1007 / Page 675 of 1158
APPLE INC.

OS/2 Bitmap (cont'd)

Encoded Bytes	Decoding Description	Decoded Bytes
04 16	Four values alternating 1 and 6	1 6 1 6
08 44	Eight values alternating 4 and 4	4 4 4 4 4 4 4 4
00 00	End of scan line	None
00 02 04 02	Move to offset four pixels forward and two rows up	None
03 E4	Three values alternating E and 4	E 4 E
00 06 12 A4 46 00	Six values of unencoded data	1 2 A 4 4 6
00 00	End of scan line	None
00 01	End of RLE data	None

And, as you probably guessed, the 24-bit RLE algorithm (RLE24) encodes pixel values using nearly the same algorithm as RLE4 and RLE8. The only difference is that run values encoded are three bytes in size (one byte per RGB color component), rather than four or eight bits in size.

Here is an example of decoding a 24-bit data stream. Each of the values is an actual color value found in the bitmap data. Run length values must be a multiple of three, and odd-length encoded runs are padded with a final zero byte, which is not part of the decoded data.

Encoded Bytes	Decoding Description	Decoded Bytes
04 16 D4 6F	Four pixels of value 16, D4, 6F	16 D4 6F 16 D4 6F 16 D4 6F 16 D4 6F
02 00 00 00	Two pixels of value 0,0,0	00 00 00 00 00 00
00 00	End of scan line	None
00 02 09 0C	Move to offset nine pixels forward and twelve rows up	None
03 E4 E4 E4	Three pixels of value E4,E4,E4	E4 E4 E4 E4 E4 E4 E4 E4 E4
00 01 12 A4 46 00	One pixel of unencoded data	12 A4 46
00 00	End of scan line	None
00 01	End of RLE data	None

Ex.1007 / Page 676 of 1158
APPLE INC.

Here is a summary of OS/2 BMP data characteristics:

Pixel Depth	Pixel Size	Compression	Color Palette
1 bit	1 bit	0,3	Yes
4 bits	4 bits	0,2	Yes
8 bits	1 byte	0,1	Yes
24 bits	3 bytes	0,4	No

For Further Information

The OS/2 BMP file format is maintained by IBM. Version 1.*x* of the OS/2 BMP format is documented in the Presentation Manager Software Development Kit (SDK). Later versions of OS/2 BMP are found in the IBM Developer's Toolkit for OS/2 Warp, version 3:

OS/2 V3 Technical Library: Presentation Manager Programming Reference (IBM part number G25H-7105).

and in the online Presentation Manager reference.

For further information, contact:

IBM Corporation
Attn: Independent Vendor League
Mail Stop 147
150 Kettletown Road
Southbury CT 06488
Voice: 203-266-2000
WWW: *http://www.ibm.com/*

IBM documentation can be ordered from:

USA: 1-800-426-7282
Canada: 1-800-465-1234
Hours: 8am to 8pm EST, Monday through Friday

Information about the OS/2 BMP format can also be found in the following references:

Charlap, David, "The BMP File Format: Part I," *Dr. Dobb's Journal*, vol. 20, no. 228, March 1995.

Charlap, David, "The BMP File Format: Part II," *Dr. Dobb's Journal*, vol. 20, no. 229, April 1995.

Ex.1007 / Page 677 of 1158
APPLE INC.

IBM Corporation, *OS/2 2.0 Programmer's Toolkit: Presentation Manager Reference* (online manual).

Luse, Marv, "The BMP File Format," *Dr. Dobb's Journal*, vol. 9, no. 219, September 1994, pp. 18–22.

Petzold, Charles, "Preserving a Device-Independent Bitmap: The Packed-DIB Format," *PC Magazine*, July 1991, pp. 433–439.

Petzold, Charles, "What's New in Bitmap Formats: A Look at Windows and OS/2," *PC Magazine*, 11 September 1990, pp. 403–410.

The code for the above issues of *Dr. Dobb's Journal* are available at:

ftp://ftp.mv.com/pub/ddj/1994/1194.09/bmp.zip
ftp://ftp.mv.com/pub/ddj/1995/1195.03/bmp.zip

The two *Dr. Dobb's Journal* articles by David Charlap contain a complete collection of source code for working with Windows 2.*x*, 3.*x*, NT, and OS/2 BMP file formats. It is available at the above FTP site and on this book's CD-ROM.

You might also be able to find some helpful information at the OS/2 shareware BBS:

Voice: 703-385-0201
BBS: 703-385-4325
WWW: *http://www.os2bbs.com/*
Telnet: *bbs.os2bbs.com*

Or at IBM's OS/2 homepage:

WWW: *http://www.austin.ibm.com/pspinfo/os2.html*

Also, look at these other online resources:

http://axion.physics.ubc.ca/os2/os2.html
OS/2 Resource listing

ftp://ftp-os2.cdrom.com/
OS/2 Archive at Walnut Creek CD-ROM

Ex.1007 / Page 678 of 1158
APPLE INC.

NAME:	P3D
ALSO KNOWN AS:	Pittsburgh Supercomputer Center 3D Metafile
TYPE:	3D scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Carnegie Mellon University
PLATFORM:	All
SUPPORTING APPLICATIONS:	P3D
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	None

USAGE: Description and storage of 3D objects.

COMMENTS: A powerful format that implements its own language and provides support for a number of common and useful renderers.

Overview

P3D is a system that originated at Carnegie Mellon University's Pittsburgh Supercomputing Center, which retains the copyright for the system. The P3D format used by the P3D system was intended for the storage of 3D models and was designed to be portable, flexible, compact, and extensible. The authors wished to create a format that would be compatible with applications, renderers in particular, on a number of platforms.

The P3D format implements a sophisticated description language, which consists of a set of extensions to Common Lisp. Perhaps to avoid implementation dependencies, only a subset of Common Lisp, called Alisp, is used. While this can be a boon to developers who already know a Lisp dialect, it provides a barrier to entry to those who don't. As a consequence, the P3D format has not

Ex.1007 / Page 679 of 1158
APPLE INC.

been used much outside academic circles, which is a shame, because it is otherwise a powerful model and a well-thought-out specification.

File Organization

A P3D file consists of a number of ASCII lines that are usually Common Lisp statements. Extensions to the language are mainly in an idiosyncratic terminology that is unfortunately at odds with most of the rest of the computer graphics world.

File Details

A P3D file stores a model, which is a set of objects; these may be geometrical structures or things more like architectural primitives, such as directed acyclic graphs. P3D normally supports the specification of spheres, cylinders, tori, polygons, polylines, polymarkers, lists of triangles, meshes, spline surfaces, font objects, and lighting objects. Various attributes may be associated with objects.

There are also procedural features designed to trigger actions in the application processing a P3D file. An example is the snap function, which can be used to trigger rendering.

A P3D file contains descriptions of one or more graphical objects, called *gobs* in the documentation. These may be primitive objects, like spheres or triangle lists, or they may consist of numbers of primitive objects. A gob is generally a directed acyclic graph, the nodes of which may be other gobs.

The P3D documentation describes gobs as follows:

> Gobs are defined either by invoking a function which returns a primitive gob, or by listing the "children" and possibly the attributes the new gob is to have. A gob can be saved either by binding it to a name (for example, via a Lisp setq function) or by including it directly into the list of children of another gob. Color, material type, and backface cullability are examples of attributes that might be associated with a gob.

There is intrinsic support in P3D for mathematical entities such as vectors, points, and lines, which may be manipulated in an arbitrary fashion. After the initial definition, a gob may be referenced repeatedly. Each reference instance may be associated with attributes and transformations independent of the those inherent in the original definition. It is the programmer's responsibility to make sure that no gob is its own descendant.

Ex.1007 / Page 680 of 1158
APPLE INC.

Coordinate System

The P3D system assumes a right-hand coordinate system. Coordinate transformations are effected by manipulation with 4×4 matrices as follows:

Rotation:

```
[ R11 R12 R13 0 ]
[ R21 R22 R23 0 ]
[ R31 R32 R33 0 ]
[  0   0   0  1 ]
```

Translation:

```
[  1   0   0  Tx ]
[  0   1   0  Ty ]
[  0   0   1  Tz ]
[  0   0   0  1  ]
```

Scale:

```
[ Sx   0   0  0 ]
[  0  Sy   0  0 ]
[  0   0  Sz  0 ]
[  0   0   0  1 ]
```

Points are defined in the following manner:

```
(defstruct point
  (x 0.0)     ;x coordinate
  (y 0.0)     ;y coordinate
  (z 0.0))    ;z coordinate
```

They can also be made with the function make-point, which takes three floating-point arguments (x,y,z) that default to zero.

```
(setq origin (make-point))
```

is the definition of the standard symbol "origin."

Vectors are defined as follows:

```
(defstruct vector
  (x 0.0)     ;x coordinate
  (y 0.0)     ;y coordinate
  (z 0.0))    ;z coordinate
```


Ex.1007 / Page 681 of 1158
APPLE INC.

Vectors can also be created through the function make-vector. Thus:

```
(setq x-vec (make-vector :x 1.0 :y 0.0 :z 0.0))
```

is the definition of the standard symbol "x-vec."

The structure holding a color is as follows:

```
(defstruct color
  (r 0.8)    ;red intensity
  (g 0.8)    ;green intensity
  (b 0.8)    ;blue intensity
  (a 1.0))   ;opacity
```

The color "red," for instance, can be made as follows:

```
(setq red (make-color :r 1.0 :g 0.0 :b 0.0))
```

A vertex may be formed in a similar manner:

```
(defstruct vertex
  (x 0.0)        ;x coordinate
  (y 0.0)        ;y coordinate
  (z 0.0)        ;z coordinate
  (clr nil)      ;local color
  (normal nil))  ;local surface normal
```

For example:

```
(setq red-origin (make-vertex :clr red))
```

creates vertex "red-origin."

Structured Fields

This discussion of structured fields in a P3D file is extracted from the P3D documentation.

The structure used to hold a material (a set of properties used with attributes like color to determine the appearance of an object) is represented as a structure with at least the following fields:

Field	Type	Meaning
:ka	float	Ambient light weighting factor
:kd	float	Diffuse light weighting factor
:ks	float	Specular light weighting factor
:exp	float	Specular exponent
:reflect	float	Reflection coefficient

Ex.1007 / Page 682 of 1158
APPLE INC.

Field	Type	Meaning
:refract	float	Index of refraction
:energy	color	Energy density (for radiosity)

Other structure fields may exist, but they are maintained by P3D and should not be modified by the programmer. A material should always be created with the "def-material" function:

```
( def-material :ka ka-value :kd kd-value :ks ks-value
  :exp exp-value :reflect reflect-value
  :refract refract-value :energy energy-color )
```

Parameters are listed below:

:ka ka-value	optional	Ambient light weighting factor
:kd kd-value	optional	Diffuse light weighting factor
:ks ks-value	optional	Specular light weighting factor
:exp exp-value	optional	Specular exponent
:reflect reflect-value	optional	Reflection coefficient
:refract refract-value	optional	Index of refraction
:energy energy-color	optional	Energy density for radiosity

This function returns material with the given characteristics.

All of the keyword-field pairs are optional. Fields that are not specified are assigned specific default values; see the specification for the "default-material" predefined symbol at the end of this document for the default values of each field.

Cameras

Cameras are defined as follows:

```
(defstruct camera
  (lookfrom origin)        ;eye point
  (lookat origin)          ;point to look at
  (up y-vec)               ;view's 'up' direction
  (fovea 56.0)             ;view included angle
  (hither -0.01)           ;hither clipping distance
  (yon -100.0)             ;yon clipping distance
  (background black))      ;background color
```


Ex.1007 / Page 683 of 1158
APPLE INC.

Gob Structures

A gob is represented as a structure with at least the following options:

Option	Type	Meaning
:attr	assoc-list	Attribute-value pairs for this gob
:transform	transformation	Coordinate transformation
:children	list	List of gobs to be children

Other structure slots may exist, but they are maintained by P3D and should not be modified by the programmer. All of the fields default to nil.

A gob should always be created with "def-gob," or with one of the geometrical primitive generators (see below). If "def-gob" is used, the definition should include a ":children" option or the gob will have no descendants in the DAG and thus be useless.

```
( def-gob :attr attrlist
  :transform transformation
  :children childlist )
```

Parameters are the following:

:children childlist	required	List of children of this gob
:transform transformation	optional	Coordinate transformation for this gob
:attr attrlist	optional	Association list of attribute and value pairs for this gob

This function returns a gob with the given children, coordinate transformation, and attributes.

For Further Information

For further information about the P3D format, see the P3D specification on the CD-ROM that accompanies this book. For up-to-date P3D documentation, see:

http://pscinfo.psc.edu/general/software/packages/p3d/p3d.html
http://pscinfo.psc.edu/general/software/categories/graphics.html

Ex.1007 / Page 684 of 1158
APPLE INC.

You can obtain P3D generators and translators via FTP from:

ftp://ftp.psc.edu/pub/p3d/

You can also contact:

Joel Welling
Pittsburgh Supercomputer Center
4400 Fifth Avenue
Pittsburgh, PA 15213
Email: *welling@psc.edu*

Ex.1007 / Page 685 of 1158
APPLE INC.

PBM, PGM, PNM, and PPM

NAME:	PBM, PGM, PNM, PPM
ALSO KNOWN AS:	Portable Bitmap Utilities, pbmplus
TYPE:	Bitmap
COLORS:	Up to 24-bit
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Jef Poskanzer
PLATFORM:	UNIX, Intel-based PCs
SUPPORTING APPLICATIONS:	pbmplus, others
SPECIFICATION ON CD:	Yes (man pages)
CODE ON CD:	Yes (in pbmplus package)
IMAGES ON CD:	No
SEE ALSO:	Most of the formats in this book

USAGE: File format conversion through an intermediary least-common-denominator format.

COMMENTS: PBM, PGM, PNM, and PPM are intermediate formats used in the conversion of many little known formats via pbmplus, the Portable Bitmap Utilities. These formats are mainly available under UNIX and on Intel-based PCs.

Overview

The Portable Bitmap Utilities (PBM) is a collection of programs organized, maintained, and primarily written by Jef Poskanzer. Although owned and copyrighted by Mr. Poskanzer, they are freely available in both source and executable form on the Internet and on many BBS systems. The "bitmap" in PBM is used in the older sense to refer to monochrome images. There are actually three other sets of programs encompassed by the PBM utilities. These are the Portable Greymap Utilities (PGM), the Portable Pixmap Utilities (PPM), and the Portable Anymap Utilities (PNM). PBM programs manipulate monochrome bitmaps, and PGM and PPM programs manipulate gray-scale bitmaps and color bitmaps, respectively. PNM programs operate on all of the bitmaps produced

Ex.1007 / Page 686 of 1158
APPLE INC.

by the other programs. There is no file format associated with PNM itself. Most people call the overall set of programs PBM and the newer version pbmplus, however, and we'll follow this convention.

Associated with pbmplus are three least-common-denominator intermediate formats. When converting a graphics file from one format to another, we speak of the *source file* (in the current format) and the *destination file* (in the desired new format). pbmplus works by taking a source file and converting it into one of the intermediate formats. That intermediate format file is then converted into the destination format.

To see how this works, here are the steps necessary to convert a Microsoft Windows Bitmap (BMP) format file named *testfile.bmp* to a GIF format file. These are 256-color files, so we use the PPM utilities *bmptoppm* and *ppmtogif*:

bmptoppm testfile.bmp	This produces *testfile.ppm*
ppmtogif testfile.ppm	This produces *testfile.gif*

The latest version of *pbmplus* is available on the CD-ROM that accompanies this book.

File Organization

The PBM, PGM, and PPM formats are each designed to be as simple as possible. Each starts out with a header, and the bitmap data follows immediately after. The header is always written in ASCII, and data items are separated by white space (blanks, tabs, carriage returns, or linefeeds). The data portion of each file type can be written in either ASCII or binary form.

File Details

There are two versions of each of the the PBM, PGM, and PPM headers. Although all the headers are in ASCII format, one is used for the ASCII version of the format, and the other is used for the binary version.

PBM Header

A PBM header consists of the following entries, each separated by white space:

MagicValue	Literally P1 for ASCII version, P4 for binary
ImageWidth	Width of image in pixels (ASCII decimal value)
ImageHeight	Height of image in pixels (ASCII decimal value)

Ex.1007 / Page 687 of 1158
APPLE INC.

PGM Header

A PGM header consists of the following entries, each separated by white space:

MagicValue	Literally P2 for ASCII version, P5 for binary
ImageWidth	Width of image in pixels (ASCII decimal value)
ImageHeight	Height of image in pixels (ASCII decimal value)
MaxGrey	Maximum gray value (ASCII decimal value)

PPM Header

A PPM header consists of the following entries, each separated by white space:

MagicValue	Literally P3 for ASCII version, P6 for binary
ImageWidth	Width of image in pixels (ASCII decimal value)
ImageHeight	Height of image in pixels (ASCII decimal value)
MaxGrey	Maximum color value (ASCII decimal value)

Image Data

After the header is a series of lines describing width×height pixels. For PPM, each pixel contains three ASCII decimal values between 0 and the specified maximum value, starting at the top-left corner of the pixmap, proceeding in normal English reading order. The three values for each pixel represent red, green, and blue, respectively; a value of 0 means that color is turned off, and the maximum value means that color is "maxxed out."

For PBM and PGM, there is only one ASCII decimal value per pixel. For PBM, the maximum value is implicitly 1.

Here is an example of a small pixmap in this format:

```
P3
# feep.ppm
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
```

You can include comments in the PBM file. Characters from a # character to the next end-of-line are ignored. There is a suggested maximum of 70 characters per line, but this is not an actual restriction.

Ex.1007 / Page 688 of 1158
APPLE INC.

Mr. Poskanzer cautions that programs that read this format should be as lenient as possible, accepting anything that looks remotely like a pixmap.

RAWBITS Variant

There is also a variant on the format, available by setting the RAWBITS option at compile time. This variant differs from the traditional format in the following ways:

- The "magic numbers" are as follows:

Format	Normal	RAWBITS Variant
PBM	P1	P4
PGM	P2	P5
PPM	P3	P6

- The pixel values are stored as plain bytes, instead of ASCII decimal:

PBM	RAWBITS is eight pixels per byte
PGM	RAWBITS is one pixel per byte
PPM	RAWBITS is three bytes per pixel

- White space is not allowed in the pixel area, and only a single character of white space (typically a newline) is allowed after the MaxGrey value.
- The files are smaller and many times faster to read and write.
- Bit order within the byte is most significant bit (MSB) first.

Note that this raw format can only be used for maximum values less than or equal to 255. If you use the PPM library and try to write a file with a larger maximum value, it automatically uses the slower, but more general, plain format.

For Further Information

For further information about the PBM, PGM, PNM, and PPM utilities, see the documentation on the CD-ROM that accompanies this book. See also the code and documentation for the pbmplus utilities, also included on the CD-ROM.

For more information about PBM, PGM, PNM, and PPM, you can contact:

Jef Poskanzer
Email: *jef@well.sf.ca.us*

Ex.1007 / Page 689 of 1158
APPLE INC.

PCX

NAME:	PCX
ALSO KNOWN AS:	PC Paintbrush File Format, DCX, PCC
TYPE:	Bitmap
COLORS:	Mono, 4-bit, 8-bit, 24-bit
COMPRESSION:	RLE, uncompressed
MAXIMUM IMAGE SIZE:	64K×64K pixels
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	ZSoft, Microsoft
PLATFORM:	MS-DOS, Windows, UNIX, others
SUPPORTING APPLICATIONS:	Too numerous to list
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes
IMAGES ON CD:	Yes
SEE ALSO:	FAX formats

USAGE: PCX is used in Microsoft Windows and Windows-based products but has found wide acceptance mainly in the MS-DOS world. It is mainly an exchange and storage format.

COMMENTS: A partially documented format in wide use, which is quick and easy to read and decompress. It lacks, however, a superior compression scheme, making it unsuitable for the storage of deep-pixel images.

Overview

PCX is one of the most widely used storage formats. It originated with ZSoft's MS-DOS-based PC Paintbrush, and because of this, PCX is sometimes referred to as the PC Paintbrush format. ZSoft entered into an OEM arrangement with Microsoft, which allowed Microsoft to bundle PC Paintbrush with various products, including a version called Microsoft Paintbrush for Windows; this product was distributed with every copy of Microsoft Windows sold. This distribution established the importance of PCX, not only on Intel-based MS-DOS platforms, but industry-wide.

Ex.1007 / Page 690 of 1158
APPLE INC.

PCX has been used by manufacturers of computer-based FAX boards and also as a general format for the storage of clip art targeted at the desktop publishing aftermarket.

The original PCX format (starting with v2.5 of PC Paintbrush) stored graphics and images with no more than 16 colors, due to the limitations of Enhanced Graphics Adapter (EGA) display technology produced by IBM. When IBM introduced the Virtual Graphics Array (VGA) display adapter, the PCX format was revised to store graphics and images with up to 256 colors.

The latest revision of the PCX format now includes the ability to store 24-bit color images. This allows the PCX format to be used for the storage of images created by the most advanced graphics, imaging, and video technology available today.

PCX is hardware-dependent in the sense that it was originally designed to accommodate a specific type of display hardware. Data may be stored either plane- or pixel-oriented, to accommodate the hardware design of the plane-oriented IBM EGA or the pixel-oriented IBM VGA display adapters.

Image data is encoded using an RLE variant, which is simple and somewhat quick in its operation, if not terribly efficient in actually reducing the size of the data. As with other RLE schemes, how much the PCX compression scheme reduces the size of a given image is difficult to say, because the reduction factor is dependent largely upon the content of the image (how "busy" the image is) and how many colors are actually used. Generally, an image incorporating 16 or fewer colors will be reduced by 40 to 70 percent from the original data, whereas a 64- to 256-color image from a scanner or video source may be reduced by only 10 to 30 percent. It is possible for an image to be so complex that the PCX compression scheme actually causes the data to increase in size after compression. (For further discussion of these and other topics, please see Chapter 9, *Data Compression.*)

File Organization

PCX files are organized into three major sections: the header, the image data, and the color palette. The color palette normally contains entries for 256 colors and is associated with the VGA display adapter. This VGA color palette is only found in later versions of the PCX image file format.

Ex.1007 / Page 691 of 1158
APPLE INC.

File Details

This section describes the major sections of PCX files and methods of reading, compressing, encoding, and decoding these files.

Header

The first 128 bytes of every PCX file is the header, which has the following format:

```
typedef struct _PcxHeader
{
  BYTE  Identifier;        /* PCX Id Number (Always 0x0A) */
  BYTE  Version;           /* Version Number */
  BYTE  Encoding;          /* Encoding Format */
  BYTE  BitsPerPixel;      /* Bits per Pixel */
  WORD  XStart;            /* Left of image */
  WORD  YStart;            /* Top of Image */
  WORD  XEnd;              /* Right of Image
  WORD  YEnd;              /* Bottom of image */
  WORD  HorzRes;           /* Horizontal Resolution */
  WORD  VertRes;           /* Vertical Resolution */
  BYTE  Palette[48];       /* 16-Color EGA Palette */
  BYTE  Reserved1;         /* Reserved (Always 0) */
  BYTE  NumBitPlanes;      /* Number of Bit Planes */
  WORD  BytesPerLine;      /* Bytes per Scan-line */
  WORD  PaletteType;       /* Palette Type */
  WORD  HorzScreenSize;    /* Horizontal Screen Size */
  WORD  VertScreenSize;    /* Vertical Screen Size */
  BYTE  Reserved2[54];     /* Reserved (Always 0) */
} PCXHEAD;
```

Identifier is an identification value defined by the PCX specification as always being 10h. This value has no real meaning other than to indicate that the file is a ZSoft PCX file. PCX readers should always check that this byte contains the proper value, even though the file may have the extension PCX. However, it is possible that a non-PCX format file might also begin with the value 10h, so the remainder of the header information should be read, and the information fields be checked for the proper values before trying to decode any image data in the file. In other words, don't just jump to byte offset 128 and start decoding what you think is encoded image data.

Version contains the version of Paintbrush that created the PCX file. ZSoft has released updated revisions of the PCX format to keep up with the increasing functionality of its PC Paintbrush program and the burgeoning display adapter technology available for the PC. Each PCX file version has separate requirements for handling and displaying its image. Prior to v2.5 of PC Paintbrush, the

Ex.1007 / Page 692 of 1158
APPLE INC.

PCX image file format was considered proprietary information by ZSoft Corporation.

Possible values for Version are shown as follows:

Value	PC Paintbrush Version and Description
0	Version 2.5 with fixed EGA palette information
2	Version 2.8 with modifiable EGA palette information
3	Version 2.8 without palette information
4	PC Paintbrush for Windows
5	Version 3.0 of PC Paintbrush, PC Paintbrush Plus, PC Paintbrush Plus for Windows, Publisher's Paintbrush, and all 24-bit image files

Encoding indicates the type of encoding used on the image data. The only encoding algorithm currently supported by the PCX specification is a simple byte-wise run-length encoding (RLE) scheme indicated by a value of 1 in this byte. It would seem to follow that if a PCX file held unencoded image data this value would be 0. PCX files, however, always contain encoded image data, and currently the only valid value for the encoding field is 1.

BitsPerPixel is the number of bits per pixel per plane in the image data. The possible values are 1, 2, 4, and 8 for 2-, 4-, 16-, and 256-color images. The planar data in a scan line is often padded with extra data to align the scan line on an even byte boundary to prevent aliasing (the "jaggies"). PCX paint and conversion programs use this value to find where in a scan line pixel data stops and extra padding begins.

XStart, YStart, XEnd, and YEnd store the size of the image in pixels. These four values are the rectangular dimensions of the visible part of the PCX image (sometimes called the *picture dimension window*) and its position relative to the physical display screen. Using these dimensions, the largest PCX image that can be stored is 65,535×65,535 pixels in size. The dimensions are the location of the upper-left and lower-right corners of the PCX image on the display screen. The upper-left corner of the screen is considered to be at location 0,0, and any PCX image with an XStart and YStart of 0 will start displaying at this location. If the XStart and YStart are values greater than zero, then a display program should start displaying the PCX image starting at those pixel coordinates. However, this is a feature rarely supported by PCX display programs.

Ex.1007 / Page 693 of 1158
APPLE INC.

Any PCX image may contain extra bytes of padding at the end of each scan line or extra scan lines added to the bottom of the image. To prevent this extra data from becoming visible, only the image data within the picture dimension window coordinates is displayed.

HorzRes and VertRes are the horizontal and vertical size of the stored image in pixels per line or dots per inch (DPI). Scanned images have the DPI value of the device that created them. Typical DPI values for a scanned image may be 100×100 DPI or 300×300 DPI. An image produced by a FAX card can have a resolution of 100×200 DPI or 200×200 DPI. Images created by paint or screen dump programs will have pixel resolution values that reflect the resolution of the display mode under which they were created. For example, a typical VGA paint program saves images with a horizontal resolution of 320 pixels and a vertical resolution of 200 pixels. However, these values are not used when decoding image data.

Palette is a 48-byte array of 8-bit values that make up a 16-color EGA color palette. The earliest version of PC Paintbrush was not able to use a modifiable EGA palette and, therefore, used only the standard palette of the EGA. Subsequent versions have allowed the use of a modifiable palette enabling a PCX image file writer to choose which 16 (or fewer) of the 64 colors available to the EGA to use.

Reserved1 is not currently used and should have a value of 00h. Older versions of PCX used this field for file identification or to hold the mode value of the display screen on which the PCX image was created. Several paint and graphics display programs will, in fact, claim that a PCX file is invalid if this field is not set to 00h.

NumBitPlanes is the number of color planes that contains the image data. The number of planes is usually 1, 3, or 4 and is used in conjunction with the BitsPerPixel value to determine the proper video mode in which to display the image. PCX video display modes are shown as follows:

Color Planes	Bits per Pixel per Plane	Maximum Number of Colors	Video Mode
1	1	2	Monochrome
1	2	4	CGA
3	1	8	EGA
4	1	16	EGA and VGA

Ex.1007 / Page 694 of 1158
APPLE INC.

Color Planes	Bits per Pixel per Plane	Maximum Number of Colors	Video Mode
1	8	256	Extended VGA
3	8	16,777,216	Extended VGA and XGA

NumBitPlanes is also used to determine the maximum number of colors a PCX image may have. The number of bits per pixel per plane is multiplied by the number of color planes and shifted to the left by one:

```
MaxNumberOfColors = (1L << (BitsPerPixel * NumBitPlanes));
```

BytesPerLine is a 16-bit value indicating the size in bytes of a color plane in an unencoded scan line. This value may be multiplied by the NumBitPlanes value to find the total length of an unencoded scan line in bytes:

```
ScanLineLength = (BytesPerLine * NumBitPlanes);
```

PaletteType contains an indicator of information held in the color palette. A value of 1 indicates color or monochrome information, while a 2 indicates gray-scale information. This value is actually an indicator of whether the image should be displayed in color or gray-scale. (Only VGA is capable of displaying true gray-scale images.) PC Paintbrush and most other programs that use PCX files ignore this value.

HorzScreenSize and VertScreenSize were added to the PCX format starting with PC Paintbrush 4.0 and 4.0 Plus. These horizontal and vertical screen-size values represent the resolution of the screen on which the image was created. This allows graphics display programs to adjust their video mode to allow for proper display of the PCX image. Because these fields were added after the release of PC Paintbrush 3.0, there is no way to know if these fields contain valid information or are part of the Reserved2 field. Therefore, always check these values to be sure they are reasonable before you use them.

Reserved2 is the last field in the header and is a run of bytes with the value 00h. This filler field is used to pad the header out to a full 128 bytes and to save room for additional fields that might be added to the header in future revisions of the PCX format. The size of this field will be either 54 or 58 bytes, depending on whether or not the header contains the HorizScreenSize and VertScreenSize fields.

Ex.1007 / Page 695 of 1158
APPLE INC.

Palette

The color palette information within a PCX file varies depending upon the version of the PCX file.

16-color EGA palette

The first version of the PCX format did not support a modifiable color palette, so the values of the standard EGA color palette were always used. Later versions of PC Paintbrush could work with or without a modifiable palette, so two more versions of the PCX format appeared, one with palette information (modifiable palette) and one without palette information (standard EGA palette).

The EGA palette is a 48-byte array of 16 RGB triples. Each color triple contains a red, green, and blue value, each with a range of 0 to 255. The palette will contain entries for 2, 4, 8, or 16 color triples with any remaining entries being set to 00h. No interpretation is necessary for display adapters using this format of color values. The EGA, however, has only four possible values for each RGB color (0 through 3), so each RGB value is shifted to the right by six to obtain the proper value. To extract the proper values to load into the EGA palette registers, the following code is used:

```
EgaColor0Red   = EgaPalette[0] >> 6;
EgaColor0Green = EgaPalette[1] >> 6;
EgaColor0Blue  = EgaPalette[2] >> 6;
EgaColor1Red   = EgaPalette[3] >> 6;
EgaColor1Green = EgaPalette[4] >> 6;
EgaColor1Blue  = EgaPalette[5] >> 6;
 ⋮
```

4-color CGA palette

The EGA color palette is also used for displaying CGA images. Two- or four-color images may be displayed on the CGA using one of eight possible color palettes, each consisting of three foreground colors and one background color.

The most significant four bits of the first byte of the EGA color palette contains the background color and is in the range of 0 to 15.

The most significant three bits of the fourth byte of the color palette contains the foreground color. The three bits of the foreground color correspond to the Color Burst Enable, Palette, and Intensity settings of the CGA, as shown below.

Ex.1007 / Page 696 of 1158
APPLE INC.

Color Burst Enable (Bit 7)	Palette (Bit 6)	Intensity (Bit 5)
0 (color)	0 (yellow)	0 (normal)
1 (monochrome)	1 (white)	1 (bright)

Code used to extract the CGA color-level data from the EGA color palette is shown below:

```
/* Get the CGA background color */
CgaBackgroundColor  = EgaPalette[0] >> 4;    /* 0 to 15 */

/* Get the CGA foreground palette */
CgaColorBurstEnable = (EgaPalette[3] & 0x80) >> 7;  /* 0 or 1 */
CgaPaletteValue     = (EgaPalette[3] & 0x40) >> 6;  /* 0 or 1 */
CgaIntensityValue   = (EgaPalette[3] & 0x20) >> 5;  /* 0 or 1 */
```

256-color VGA palette

When PCX was conceived, the EGA was the premium display adapter available from IBM for the PC. The EGA could display only 16 colors from a palette of 64, so PCX was originally designed with a color palette large enough to hold only 16 colors.

The 16-color EGA technology of 1984, however, gave way to the 256-color VGA technology of 1987. PCX now fell short of VGA standard images that could contain up to 256 colors from a palette of 262,144, and a new color palette needed to be added to the PCX file format for VGA images. Because there was not enough room in the header for it, the designers of the PCX format appended it to the end of the PCX file itself.

This unconventional, if not inconvenient, location for the VGA palette presents a problem; because the size of the image data varies, the location of the VGA palette is different for every file. The position of the palette must be determined by its offset from the end of the file rather than from the beginning.

To see if a VGA palette is attached to a file, seek backwards 769 bytes from the end of the file. If the byte at this location is set to value 0Ch, then the 768 bytes following this value constitute a VGA color palette. The PCX specification states that if the version number in the header (byte 1) is 5 (v3.0), then there might be a VGA color palette attached.

Normally, a PCX file must have a VGA color palette attached only if there are more than 16 colors in the image; otherwise the EGA palette can be used. However, many graphics programs create v3.0 PCX image files without a VGA color palette, while other programs always attach a VGA color palette, even for

Ex.1007 / Page 697 of 1158
APPLE INC.

2-color images. To confuse things even more, 24-bit PCX images are always marked as v3.0, yet never have an attached color palette.

A v3.0 PCX image might not have a color palette; the value 768 bytes from the end of the file might be 0Ch by coincidence. In this rare case, a PCX reader would interpret the last 768 bytes of the encoded image data as a VGA palette, so a truly bizarre displayed image would result. One solution to this problem would be to first read all the image data and note whether the file pointer stopped 769 bytes from the end of the file. If so, then a VGA color palette is present. Another method would be to check the three bytes following the 0Ch value. This is the first color of the color palette and is normally black, so the three bytes following the suspect VGA palette indicator value should all be zero.

When a VGA palette is present in the file, its information is always used to display the image data, rather than using any information that may be present in the EGA color palette. If the colors in an image do not display correctly, it may be necessary to disable the color palette so the display hardware may use its native color palette. Disabling the color palette is accomplished by changing the version number in the header (byte 1) from 5 to 3. The display software should recognize that this version of the format has no color palette and, therefore, should use its own default palette.

The VGA palette itself is an array of 768 bytes (256×3) containing the red, green, and blue values for each of the 256 possible colors in a VGA PCX image. Color values are organized into triples, as in the EGA palette. Bytes 0, 1, and 2 are the red, green, and blue values for the color 0; bytes 3, 4, and 5 are the red, green, and blues values for color 1; and so on. Each RGB value is in the range of 0 to 255.

In fact, the VGA palette is simply a much longer version of the EGA palette. VGA display devices, however, require that palette color values be in the range of 0 to 63, so all RGB values should be divided by four (shifted to the right twice). VGA images may have 2-, 4-, 8-, 16-, 32-, 64-, 128-, or 256-color entries in the palette.

Reading the PCX Header

The PCX specification does not specifically state that the PCX image file format must use the least significant byte–ordering scheme used on Intel 80*x*86 processors, but we may safely assume that this is so because the PCX format was developed for use on Intel-based machines. If code that reads PCX-format files

Ex.1007 / Page 698 of 1158
APPLE INC.

will only be executed on Intel machines, it is possible, although not portable, to use the *fread*() function to read the header on a little-endian machine:

```
PCXHEADER pcx;
if(fread(&pcx, sizeof(char), sizeof(PCXHEADER), fp) !=
        sizeof(PCXHEADER))
    fputs("Error reading PCX header.", stderr);
```

Compressing PCX Data

The data-encoding algorithm used in PCX files is a simple 1-byte/2-byte run-length encoding scheme. While this type of encoding is not the most effective in terms of reducing data size, it is very quick in its operation and quite easy to implement.

An image normally contains many series of pixel runs, that is, two or more contiguous pixels of the same value. Using the run-length data compression scheme, a run of pixels several bytes in length may be converted to a run code only two bytes in length.

The encoded data is read one byte at a time. If the two most significant bits (MSB) of the first byte read are set to 1, then this byte is the first byte of a 2-byte run code. The first byte in a 2-byte run code always contains the run count in its lower six bits, which is the length of the pixel run. Therefore, a pixel run may be 1 to 63 pixels in length.

Using the two most significant bits to indicate a 2-byte code rather than just one, MSB is a holdover from the early CGA days of Paintbrush. Use of only one MSB resulted in poor compression for CGA data, so two were used instead.

The second byte of a 2-byte run code is the value of the pixel run itself. This value may be in the range 0 to 255 and is written to the output a number of times equal to the run count.

If a run-count byte is read and the two MSBs are both 0, then this byte is a run-value byte, and the run count is considered to be 1. This 1-byte run code is used to prevent a 1-pixel run from encoding into a 2-byte run code.

The PCX RLE encoding scheme is not perfect, however. A 1-byte run code can contain a run value only in the range 0 to 63. If the pixel run value is in the range 64 to 255, a 2-byte run code must be used instead. If an image contains many single pixel runs of color values greater than 63, an increase of image data size can occur after PCX encoding. Such an increase in data size typically occurs only in very noisy or grainy images.

Ex.1007 / Page 699 of 1158
APPLE INC.

Decoding a PCX Format File

To decode a file in PCX format, you must read the header of the file and calculate the following data:

- Width of image in pixels
- Length of image in scan lines
- Number of bytes needed to hold a decoded scan line
- Number of padding bytes at the end of each scan line

Calculate the image width and height from the image dimension values as follows:

```
ImageWidth  = XEnd - XStart + 1; /* Width of image in pixels */
ImageHeight = YEnd - YStart + 1; /* Length of image in scan lines */
```

Calculating the number of bytes required to hold a decoded scan line is necessary if the decoded image data is to be stored in a buffer. It is also necessary to determine if the image data has been encoded across scan lines. The number of color planes multiplied by the number of bytes per line per plane yields this value:

```
ScanLineLength = NumBitPlanes * BytesPerLine;
```

The length of padding at the end of a scan line may be determined by calculating the number of pixels in an unencoded scan line and comparing this value with the pixel width of the displayed image:

```
LinePaddingSize = ((BytesPerLine * NumBitPlanes) *
                  (8 / BitsPerPixel)) - ((XEnd - XStart) + 1);
```

The decoding steps are the following:

1. Read a byte.
2. If the two MSBs are set to 1, then mask off the run count.
3. Read next byte.
4. Write the byte a number of times equal to the run count.
5. Else, if the two MSBs are set to 0, then mask off the run value.
6. Write the byte once.
7. Repeat steps 1 through 6 until the buffer is full.

Ex.1007 / Page 700 of 1158
APPLE INC.

The code used to decode a scan line of information is as shown as follows:

```
/*
** Decode a PCX scan line.
**
** In this example the size of Buffer[] and the value of BufferSize
** is equal to the scan line length. Data is read from the FILE
** stream fpIn and written to Buffer[].
*/
do
{
  byte = GetByte(fpIn);                    /* Get next byte */
  if ((byte & 0xC0) == 0xC0)               /* 2-byte code */

    runcount = byte & 0x3F;                /* Get run count */
    runvalue = GetByte(fpIn);              /* Get pixel value */
    }
  else                                     /* 1-byte code */
    {
    runcount = 1;                          /* Run count is one */
    runvalue = byte;                       /* Pixel value */
    }
  /* Write the pixel run to the buffer */
  for (total += runcount;                  /* Update total */
    runcount && index < BufferSize;        /* Don't read past buffer */
  runcount--, index++)                     /* Update counters */
  Buffer[index] = runvalue;                /* Assign value to buffer */
  } while (index < BufferSize);            /* Read to end of buffer */
```

The PCX specification states that a decoding break should occur at the end of each scan line. This means that when a run of data is being encoded, and the end of the scan line is reached, the run should stop and not continue across to the next scan line, if it is possible to stop it.

Decoding can be complicated by PCX files that have been encoded ignoring this rule. Encoding across scan lines gains a few extra bytes of compression, but the process of decoding a single scan line is made much more difficult.

Encoding PCX Image Data

The scheme for encoding a scan line is fairly straightforward with only a few exceptions. Raw data is read one byte at a time. The only information needed is the number of bytes in a scan line. The following is the procedure for encoding image data using the PCX compression algorithm:

Ex.1007 / Page 701 of 1158
APPLE INC.

1. Read a byte of pixel data, and store the value.
2. Set counter to 1.
3. Read the next byte, and check if it is the same as the stored value.
4. If it is the same, increment the counter.
5. If it is not the same and the count is greater than one, or the count is 63, or if the end of the scan line has been reached, then mask on the two MSBs, and output the count value.
6. Output the data value.
7. Repeat steps 1 though 6 until all scan lines have been read.

PCX Image Data Format

Once a scan line has been decoded, the format of the data it contains depends upon the BitsPerPixel and the NumBitPlanes values found in the header. Knowing the data format of a scan line is necessary so you can parse the pixel data from a scan line for display of the image or conversion of the image file from one format to another. All scan lines in a PCX file always have the same format.

Scan-line pixel data is stored in one of two ways—either pixel-oriented or plane-oriented. Pixel-oriented data is stored with all the pixel data (either real data or indexes into a color palette) in a contiguous line. Plane-oriented data unrolls the pixel data into its red, green, and blue components and groups them by color across the scan line.

Single-plane data is stored pixel by pixel in one long plane that runs the length of the scan line. The data in the scan line is not the actual image data itself but is instead a series of index values into either the EGA or VGA color palettes. The exception for single-plane data is the 1-bit monochrome image, where each bit in a scan line maps directly as a pixel value.

How much of the scan-line data a single pixel occupies is determined from the BitsPerPixel value. For example, with one bit per pixel, every byte of scan-line data contains eight pixel values. With eight bits per pixel, every byte of scan-line data contains one pixel value. Monochrome, CGA, and 256-color VGA images usually contain only a single plane per scan line.

Scan lines with three planes are uncommon, but they do exist. 24-bit PCX images are stored using three bytes per pixel spread over three planes. The

Ex.1007 / Page 702 of 1158
APPLE INC.

24-bit data values are the actual color values for the image, and no color palette is used. Paintbrush for Windows 2.0 uses a 3-plane/1-bit data format to store 8-color images, where each pixel value is an index into the EGA color palette.

Images with four planes are usually 16-color EGA images. In addition to the red, green, and blue planes, there is a fourth intensity color plane that is specific to the EGA display card. Scan-line data in 4-plane images contains index values into the EGA palette.

Related File Formats

Several other formats are direct spinoffs of the PCX file format. And in most cases they are just specialized versions of PCX.

PCC image file format

Earlier versions of PC Paintbrush had the capability of clipping and copying an area of a PCX image and saving it to a file using the Copy To... command. The resulting file was saved as a PCX format file with the extension .PCC, possibly to indicate that the image the file contained was a portion of another image. The current version of PC Paintbrush does not use the .PCC extension and uses the .PCX extension instead.

DCX image file format

The PCX file format is capable of storing only a single image per file. Applications that require two or more PCX image files to be identified as belonging to the same group often use a naming convention that will identify a collection of PCX files as being related to one another.

One such application is FAX software, where each facsimile page is stored as an individual image in a separate file. PCX became a popular format for PC-based FAX software, because facsimile pages saved in this format could be viewed using many popular paint and image display programs that supported PCX. However, storing each FAX page as a separate file can become quite cumbersome and also confusing, if each image has a cryptic filename.

In an effort to store PCX files in a manner more appropriate to facsimile applications, the DCX file format was created. The DCX format stores up to 1023 PCX images within a single DCX file. Each image in the DCX file is a complete PCX image file, including header and palette information. In applications, DCX files may contain all of the pages of a facsimile transmission, a series of

Ex.1007 / Page 703 of 1158
APPLE INC.

images of the same subject, or all of the illustrations within a document. The DCX header follows:

```
typedef struct _DcxHeader
{
  DWORD Id;                        /* DCX Id number */
  DWORD PageTable[1024];           /* Image offsets */
} DCXHEAD;
```

Id is a 4-byte word used to identify the file. The value of this word is 3ADE68B1h (987,654,321 decimal).

PageTable is a table of 1024 4-byte word values. The values in this table are the offsets of each PCX image contained within the DCX file. The offset of each PCX image is measured from the beginning of the file (byte 0). The last entry of the page list is the terminator value and is always set to zero.

Typically, a DCX file contains an entire 4096-byte page list (1023 4-byte offset values followed by a 4-byte terminator value), even if most of the values in the list are zero. Some DCX file writers may try to save space by writing only the values of the offsets, followed by a 0 terminating word, but not the remaining part of the list. It is, therefore, important never to expect the page list to be a full 4096 bytes in length. DCX file readers should always read one value at a time and stop when a word value of zero is read. If the first offset value in the page list is 1004h (4100 decimal), then an entire 4096-byte page list is contained within the DCX file.

The DCX format is quite convenient and very easy to use; however, this format suffers from one major drawback. When a series of PCX files is concatenated into a DCX file, all the information within the PCX files is preserved, but the actual names of the PCX files are lost. No provision in the DCX format (or in the PCX format for that matter) exists for storing the MS-DOS filename of the PCX image files. Therefore, if the original PCX filenames are important to your application, you will have to devise some sort of name list that is maintained outside of the DCX file. Future revisions of the DCX format might correct this oversight (perhaps by appending a name list onto the end of the DCX file itself).

For Further Information

For further information about the PCX format, see the format specification included on the CD-ROM that accompanies this book.

Ex.1007 / Page 704 of 1158
APPLE INC.

The PCX format was created and is maintained by ZSoft Corporation. For additional information, contact ZSoft at:

ZSoft Corporation
Attn: Shannon Donovan
450 Franklin Road, Suite 100
Marietta, GA 30067
Voice: 770-428-0008
FAX: 770-427-1150
BBS: 770-427-1045
CompuServe: 76702,1207

ZSoft publishes the following technical reference manual describing the PCX format:

ZSoft Corporation, *Technical Reference Manual*, Revision 5, Marietta, GA, 1990.

PCX is a very popular format that has been described in many books and magazine articles. The following manual and magazine articles also document the format and use of PCX files:

Ashdown, Ian, "PCX Graphics," *C Users Journal*, vol. 9, no. 8, August 1991, pp. 89–96.

Azer, S., "Working with PCX Files," *Microcornucopia*, no. 42, July–August 1988, p. 42.

Luze, Marv, "Printing PCX Files," *C Gazette*, vol. 5, no. 2, Winter 1990–91, pp. 11–22.

Quirk, K., "Translating PCX Files," *Dr. Dobb's Journal*, vol. 14, no. 8, August 1989, pp. 30–36, 105–108.

Ex.1007 / Page 705 of 1158
APPLE INC.

PDS

NAME:	PDS
ALSO KNOWN AS:	Planetary Data System Format
TYPE:	General data format
COLORS:	Unlimited
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	NA
ORIGINATOR:	NASA
PLATFORM:	All
SUPPORTING APPLICATIONS:	Many
SPECIFICATION ON CD:	No (specification is too lengthy)
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	FITS, VICAR2

USAGE: PDS is used by NASA and other institutions to store planetary data.

COMMENTS: PDS is an ODL-based format similar in form and use to FITS and VICAR2.

Overview

The PDS (Planetary Data System) file format is a standard format devised by the Planetary branch of the National Aeronautics and Space Administration (NASA) for storing solar, lunar, and planetary data collected on Earth and by interplanetary spacecraft.

PDS is actually a set of rules for the construction of labels to describe the structure of a variety of data files, including images. The basis for the labels is an Object Description Language (ODL), which describes each separate component of the data file as a distinct object. These labels have been designated for use by NASA's Planetary Data System.

A PDS image file could conceivably contain a number of different elements. The label might be a part of the data file, or it might be in a separate file. Each image row could have some leading and trailing information. There could be color palette or image histogram data before or after the image. There could

Ex.1007 / Page 706 of 1158
APPLE INC.

even be multiple images in a single file. All of these possibilities can be handled with the PDS syntax.

This syntax has been used on the CD-ROMs of spacecraft data distributed by the PDS, such as the 12-disk "Voyagers To The Planets" set. A number of software packages also support PDS labels.

File Organization

A PDS data file consists of a header (called the label) and a set of data objects. The label and data objects may reside in the same file or in separate files.

A PDS label is usually a collection of ASCII text records. The label records can be fixed length or variable length, although fixed length is more portable. Records are usually delimited by a carriage return and a linefeed character to ensure readability on the widest possible variety of computers.

The label uses ODL to give information about the data objects. The ODL object description has the general form:

```
OBJECT = object_name Information about the object
END_OBJECT
```

Statements within an object description all have the form:

```
name = value
```

where *name* is a keyword, the name of a particular attribute associated with the object, and *value* is the value of the attribute. The attribute name can be up to 32 characters in length. The first character must be alphabetic, but the remaining characters can be alphabetic, numeric or the underscore character. The attribute values can be numeric (integer, real, or real with units), literal or enumerated values, strings, times, or object names. The values can also be arranged into arrays.

Direct access to data objects is possible by utilizing pointers to the objects in the label. A pointer is expressed with the notation:

```
^object_name = location
```

The *location* may be numeric, in which case it represents a starting record number for the object, or it may be a string giving the name of an external file.

Ex.1007 / Page 707 of 1158
APPLE INC.

The first record of a PDS label may be a Standard Format Data Unit (SFDU) ID in the format:

```
nnnnnnnnnnnnnnnnnnnn = SFDU_LABEL
```

The ID is assigned by a central control authority (Consultative Committee for Space Data Systems), but it can be safely skipped by application software. Each PDS label must end with a statement of the form:

```
END
```

Comments may be embedded in the label, and a comment begins with the pair of characters /* (slash asterisk). The comment ends either at the end of the line or with the pair of characters */ (asterisk slash).

The PDS ODL is an evolving syntax, although downward compatibility is maintained. A number of new enhancements are supported under the second version of the syntax standard.

File Details

Here is a sample PDS version 1 label. The label is treated as a single byte stream of 203 bytes. Notice the (now obsolete) pointer syntax.

```
FILE_TYPE           = IMAGE
HEADER_RECORDS      =  1
HEADER_RECORD_BYTES = 203
IMAGE_LINES         = 512
LINE_SAMPLES        = 512
SAMPLE_BITS         =   8
IMAGE_POINTER       = OLDIMAGE.IMG
END
```

Here is sample PDS version 2 label. The image data starts at the tenth record in the file (that is, byte 2550).

```
NJPL1I00PDS100055825   = SFDU_LABEL
RECORD_TYPE    = FIXED_LENGTH
RECORD_BYTES   = 255
FILE_RECORDS   = 223 /* 210 image records + 13 label */
LABEL_RECORDS  = 13
/* This is a pointer to the file record where the image starts */
^IMAGE  = 10
OBJECT  = IMAGE
        LINES   = 210
        LINE_SAMPLES    = 255
        SAMPLE_BITS     = 8
END_OBJECT
END
```


Ex.1007 / Page 708 of 1158
APPLE INC.

Here is another sample PDS version 2 label for an image with a detached label file:

```
CCSD3ZF0000100000001NJPL3IF0PDS200000001 = SFDU_LABEL
/*          File Format and Length                  */
RECORD_TYPE                 = FIXED_LENGTH
RECORD_BYTES                = 1024
FILE_RECORDS                = 512
/*          Record Pointers to Major Objects        */
^IMAGE                      = 'SAMPLE.IMG'
/*          Descriptions of Objects in File         */
OBJECT = IMAGE
 LINES = 512
 LINE_SAMPLES  = 512
 SAMPLE_BITS   = 16
END_OBJECT     = IMAGE
END
```

For Further Information

Because the PDS specifications are so lengthy, and because they are freely available, we have decided not to include them on the CD-ROM that accompanies this book. For additional information about PDS, contact the JPL customer support facility:

National Aeronautics and Space Administration (NASA)
Planetary Branch
Jet Propulsion Laboratory
Mail Stop 525-3610
4800 Oak Grove Drive
Pasadena, CA 91109
Voice: 818-354-7587
Email: *PDS_Operator@jplpds.jpl.nasa.gov*
WWW: *http://stardust.jpl.nasa.gov/pds_home.html*

There is a set of several documents on PDS labels available from this facility:

Jet Propulsion Laboratory, *Standards for the Preparation and Interchange of Data Sets*, JPL Document D-4683, NASA, Pasadena, CA, 1988.

Jet Propulsion Laboratory, *Data Preparation Workbook*, JPL Document D-7669, NASA, Pasadena, CA, 1990.

Jet Propulsion Laboratory, *Planetary Data System Standards Reference*, JPL Document D-4683, NASA, Pasadena, CA, 1990.

Ex.1007 / Page 709 of 1158
APPLE INC.

Jet Propulsion Laboratory, *Specification for the Object Description Language*, NASA, Pasadena, CA, 1990.

Ex.1007 / Page 710 of 1158
APPLE INC.

NAME:	Pictor PC Paint
ALSO KNOWN AS:	PC Paint, PIC, .PIC, .CLP
TYPE:	Bitmap
COLORS:	24-bit maximum
COMPRESSION:	RLE, uncompressed
MAXIMUM IMAGE SIZE:	64K×64K pixels
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	Paul Mace Inc.
PLATFORM:	MS-DOS
SUPPORTING APPLICATIONS:	Too numerous to list
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	PCX

USAGE: Paint programs.

COMMENTS: A device-specific format designed for PC (MS-DOS) hardware.

Overview

The Pictor PC Paint format is device-dependent and is specifically designed around the needs of the IBM family of display adapters (CGA, EGA, VGA, and so on). Because of this, the PIC format resembles PCX, another popular paint file format designed specifically for IBM hardware. This format uses the .PIC extension.

File Organization

The header structure for the Pictor PC Paint format is 17 bytes long and consists of the following fields:

```
typedef struct _PicHeader
{
WORD  Id;                /* Magic number (always 1234h) */
WORD  Width;                   /* Width of image in pixels */
```


Ex.1007 / Page 711 of 1158
APPLE INC.

```
WORD  Height;          /* Height of image in pixels */
WORD  XOffset;         /* X of lower left corner of image */
WORD  YOffset;         /* Y of lower left corner of image */
BYTE  PlaneInfo;       /* BPP and number color planes */
BYTE  PaletteFlag;     /* Color palette/video flag */
BYTE  VideoMode;       /* Video mode of image */
WORD  PaletteType;     /* Type of color palette */
WORD  PaletteSize;     /* Size of color palette */

} PICHEAD;
```

File Details

Id is an identification value. This value is always 1234h.

Width and Height contain the size of the image in pixels.

XOffset and YOffset indicate the position of the image on the display screen. The default values of 0 and 0 indicate that the image starts at the origin point in the lower-left corner of the screen.

PlaneInfo contains two values. Bits 0 through 3 contain the number of bits per pixel per plane in the image. Bits 4 through 7 contain the number of additional color planes; there is always a minimum of one color plane. This value is 0 for one color plane, 2 for three color planes, and so on. These values may be used to determine the type of display hardware for which the image data is formatted. A value of 02h in this field indicates CGA data; a value of 31h indicates EGA data; and a value of 08h indicates VGA data.

The original version of the PC Paint format did not include any information on video modes or color palettes. Version 2.0 of the format adds the ability to store this additional information and increases the size of the header.

PaletteFlag contains the value FFh if the version of the PC Paint file is 2.0 or greater. In this case, data is present for the remaining three fields of the header. If the Marker field value is not FFh, then image data immediately follows the header.

VideoMode contains a single ASCII alphanumeric character indicating the screen mode used to create the image. This is useful only for setting the screen mode before displaying the image. The following mode values are used:

Ex.1007 / Page 712 of 1158
APPLE INC.

0	40 column text	F	EGA 640x350x4
1	80 column text	G	EGA 640x350x16
2	Monochrome text	H	EGA 720x348x2 (Hercules)
3	EGA 43-line text	I	EGA 320x200x16 (Plantronics)
4	VGA 50-line text	J	EGA 320x200x16
A	CGA 320x200x4	K	EGA 640x400x2 (AT&T or Toshiba 3100)
B	EGA 320x200x16	L	VGA 320x200x256
C	CGA 640x200x2	M	VGA 640x480x16
D	EGA 640x200x16	N	EGA 720x348x16 (Hercules InColor)
E	EGA 640x350x2	O	VGA 640x480x2

PaletteType indicates the type of color palette that is found after the header. A value of 0 indicates that no color palette is present (i.e., the image does not use a color palette, typical of monochrome image data). A value of 1 indicates a CGA color palette and border color. A value of 2 indicates a PC Jr. or non-ECD 16-color palette. A value of 3 indicates an EGA palette. A value of 4 indicates a VGA palette.

The CGA palette data is a single byte in size and the border data is also one byte in size. (See the PCX article for information on interpreting CGA palette data.) PC Jr. palette data, which may also be a generic 16-color palette, is stored as a 16-byte palette, one color value per byte. The VGA palette is stored as 256 3-byte triples, the same as in the PCX format.

PaletteSize stores the number of bytes of palette data that follow the header. For a CGA palette, this value is 2. For the PC Jr. and EGA palettes, this value is 16. For a VGA palette, this value is 768.

Image Data Encoding

The image data in a Pictor Paint image file may be stored in a compressed RLE format. Following the color palette data (if present) is a 16-bit data word that indicates how many run-length encoded blocks of data follow. If the image data in the Pictor file is not compressed, this value is 0.

Raw Pictor images never contain any color palette information, although the color palette information fields may be present in the header. The image data begins immediately after the header and is displayed starting at the bottom-left corner of the display screen. The image data always is stored by scan line and in consecutive color planes.

The RLE scheme used in the PC Paint format encodes runs of identical pixel values into blocks (also called packets). The number of data blocks in an image file is indicated by the 16-bit value found after the color palette information.

Ex.1007 / Page 713 of 1158
APPLE INC.

Each run-length encoded block begins with a 5-byte header. This header, which contains the information necessary to decode the image data stored in its data block, has the following format:

```
typedef struct _PicBlockHead
{
WORD  BlockSize;     /* Size of encoded block including header */
WORD  RunLength;     /* Size of decoded pixel data */
BYTE  RunMarker;     /* Start-of-run indicator */

} PICBLKHEAD;
```

BlockSize is the size of the entire block of encoded image data including the block header. This value is useful for reading the entire data block into memory before decoding it.

RunLength contains the total number of pixels encoded in this block.

RunMarker contains a unique character marker that identifies the start of an encoded run in this block. Data blocks may contain multiple runs of pixels, and this marker delineates the start of each encoded run in the block.

Each encoded data block may contain one or more runs of pixel data. The runs may be eight bits in length (1 to 255 pixels), or 16 bits in length (1 to 65,535 pixels). For buffering reasons, a run typically does not exceed 8192 pixels (or bytes) in length. It is also possible to store a literal run of pixels in a data block that is not encoded at all.

The first five bytes of an encoded data block make up the block header. Following the header is normally a RunMarker character designating the start of an encoded run. The byte following a RunMarker is the RunLength. This is an 8-bit value that stores the length of the pixel run. If this value is not zero, then the byte that follows it, the RunValue, is the actual pixel value that is to be repeated RunLength times:

```
WORD  BlockSize   Size of encoded block including header
WORD  RunLength   Size of decoded pixel data
BYTE  RunMarker   Start-of-run indicator
BYTE  RunMarker   Start-of-run indicator
BYTE  RunLength   Length of the pixel run (8-bit run length)
BYTE  RunValue    The value of the pixel run
```

If the RunLength value is 0 then a 16-bit word value, the RunCount, follows the RunLength field. The byte following the RunCount is the actual pixel run that is to be repeated RunCount times:

Ex.1007 / Page 714 of 1158
APPLE INC.

```
WORD  BlockSize   Size of encoded block including header
WORD  RunLength   Size of decoded pixel data
BYTE  RunMarker   Start-of-run indicator
BYTE  RunMarker   Start-of-run indicator
BYTE  RunLength   Length of the pixel run (8-bit run length)
WORD  RunCount    Length of the pixel run (16-bit run length)
BYTE  RunValue    The value of the pixel run
```

If the RunMarker is missing from a data block, the byte read is assumed to be a literal pixel value and is written directly to the output:

```
WORD  BlockSize   Size of encoded block including header
WORD  RunLength   Size of decoded pixel data
BYTE  RunMarker   Start-of-run indicator
BYTE  PixelValue  No RunMarker, literal pixel value
```

The RunMarker character is an arbitrary value chosen to delineate the start of each encoded run in a data block. The RunMarker value should not be the same as any RunValue or PixelValue in the data block. Each data block uses a RunMarker value appropriate to the data in the data block. The following example is of a data block that uses a RunMarker value that is the same as a pixel RunLength value. This arrangement could confuse a Pictor RLE decoder:

```
WORD  BlockSize   Size of encoded block including header
WORD  RunLength   Size of decoded pixel data
BYTE  RunMarker   Start-of-run indicator
BYTE  RunMarker   Start-of-run indicator
BYTE  RunLength   Length of the pixel run (8-bit run length)
BYTE  RunValue    The value of the pixel run
```

Because there is no "end of data block" marker, a PC Paint decoder must keep track of the number of pixels decoded in each data block, and must compare this value to the value of the RunLength field of the block header. When these values are equal, the block is finished and the next block, if any, should be read.

The decoded format of the image data varies depending upon the type of graphics display adapter that was used to create the image. Monochrome images are stored eight pixels per byte. EGA images are stored two pixels per byte in four planes of 4-bit index values each, in a BGRI order. VGA image data is stored one pixel per byte, each byte being an index value into the color palette. When there is more than one color plane, the image data is stored by plane first, then by pixel (plane 0, plane 1, plane 2, and so on).

The following pseudocode details the decoding process of the Pictor RLE image data:

Ex.1007 / Page 715 of 1158
APPLE INC.

```
StartOfDataBlock:
  Read BlockSize value from data block header
  Read RunLength value from data block header
  Read RunMarker value from data block header
StartOfRun:
  If the next byte is a RunMarker
    If the byte following the RunMarker is not 0
      Read the next byte as the RunLength
      Read the next byte as the RunValue
      Write the RunValue 'RunLength' times.
    else
      If the byte following the RunMarker is 0
        Read the next word as the RunCount
        Read the next byte as the RunValue
        Write the RunValue 'RunCount' times.
  else
    If the byte following the header is not a RunMarker
      Write the byte as a literal PixelValue
    If the number of pixels written so far equals the RunLength
      Goto StartOfDataBlock:
  else
  If the number of pixels written so far does not equal the RunLength
    Goto StartOfRun:
```

Below are several examples of Pictor PC Paint run-length encoded data blocks. These examples show how one or more different types of runs may be encoded in the same data block.

The following encoded data block is 10 bytes in size and contains a single run 800 pixels in length. The start-of-run indicator is the value FFh. The Run-Length field is 0, so the RunCount field contains the number of pixels in the run. The RunValue is the actual pixel value in the run.

```
WORD  BlockSize   Size of encoded block including header
WORD  RunLength   Size of decoded pixel data (800)
BYTE  RunMarker   Start-of-run indicator
BYTE  RunMarker   Start-of-run indicator (run 1)
BYTE  RunLength   Length of the literal run
WORD  RunCount    Length of the encoded run
BYTE  RunValue    The value of the pixel run
```

The following encoded data block is 13 bytes in size and contains two runs. A total of 8256 pixels are encoded in this data block and the RunMarker is the

Ex.1007 / Page 716 of 1158
APPLE INC.

value 80h. This first run is 64 pixels in length and has a value of 7. The second run is 8192 pixels in length and has a value of 1.

```
WORD  BlockSize   Size of data block including header
WORD  RunLength   Size of decoded pixel data (64 + 8192)
BYTE  RunMarker   Start-of-run indicator
BYTE  RunMarker   Start-of-run indicator (Run 1)
BYTE  RunLength   Length of the pixel run
BYTE  RunValue    Value of the pixel run
BYTE  RunMarker   Start-of-run indicator (Run 2)
BYTE  RunLength   Value is 0, get 16-bit RunCount
WORD  RunCount    Length of the pixel run
BYTE  RunValue    Value of the pixel run
```

The following encoded data block is 1039 bytes in size and contains two encoded runs and a literal run. The RunMarker in this block is the value 00h. The value FFh cannot be used because the block contains a run with this value. The first run is 1024 pixels in length, and each pixel has the value 01h. Following this run are three literal pixel values that are considered the second run in the block (literal pixel runs are not prefaced with a RunMarker). The third run contains 12 pixels each of the value FFh.

```
WORD  BlockSize   Size of data block including header
WORD  RunLength   Size of decoded pixel data (1024+1+1+1+12)
BYTE  RunMarker   Start-of-run indicator
BYTE  RunMarker   Start-of-run indicator (Run 1)
BYTE  RunLength   Value is 0, get 16-bit RunCount
WORD  RunCount    Length of the pixel run
BYTE  RunValue    Value of the pixel run
BYTE  PixelValue  No RunMarker, literal pixel value (Run 2)
BYTE  PixelValue  No RunMarker, literal pixel value
BYTE  PixelValue  No RunMarker, literal pixel value
BYTE  RunMarker   Start-of-run indicator (Run 3)
BYTE  RunLength   Length of the pixel run
BYTE  RunValue    Value of the pixel run
```

HiColor format

Four changes were made to Pictor PC Paint to support hicolor (65.536K colors) and truecolor (16,777,216 colors) video modes, and to correct past problems in compressing text-mode images.

First, the two 4-bit fields for the number of bit planes and number of bits per pixel (PlaneInfo) were combined into one field to support pixel depths greater than eight bits. The newly supported modes, and their hex values, are:

```
01      1 bit plane     1 bit per pixel 2 colors
02      1 bit plane     2 bits per pixel 4 colors
04      1 bit plane     4 bits per pixel 16 colors
```


Ex.1007 / Page 717 of 1158
APPLE INC.

```
11      2 bit planes    1 bit per pixel 4 colors
31      4 bit planes    1 bit per pixel 16 colors
08      1 bit plane     8 bits per pixel 256 colors
10      1 bit plane     16 bits per pixel 32,768 and 65,536 colors
18      1 bit plane     24 bits per pixel 16,777,216 colors
28      3 bit planes    8 bits per pixel 16,777,216 colors
```

Second, all text-mode images are now stored at 16 bits per pixel instead of eight bits per pixel, as in the past.

Third, the video mode may now be specified using two letters. The mark field, which always contained the value 0xFF in the past, is now the second letter. Valid values for the mark field are currently the ASCII characters 1, 2, and 3.

The hicolor extensions expanded the list of video modes supported by the .PIC format:

```
0       10  40x25 color text
1       10  80x25 color text
2       10  80x25 B&W text
3       10  EGA 80x43, VGA 80x50 color text
4       10  VESA 80x60 color text
5       10  VESA 132x25 color text
6       10  VESA 132x43 color text
7       10  VESA 132x50 color text
8       10  VESA 132x60 color text
A       02  CGA 4 color
B       04  PCjr/Tandy 16 color
C       01  CGA 640x200 2 color
D       31  EGA 640x200 16 color
E       01  EGA 640x350 2 color
F       11  EGA 640x350 4 color
G       31  EGA 640x350 16 color
H       01  Hercules 720x348 2 color
I       31  VGA 640x350 16 color
J       31  EGA 320x200 16 color
K       01  AT&T/Toshiba 640x400 2 color
L       08  VGA/MCGA 320x200 256 color
M       31  VGA 640x480 16 color
N       31  Hercules InColor 720x348 16 color
O       01  VGA/MCGA 640x480 2 color
P       01  EGA/VGA 800x600 2 color
Q       31  EGA/VGA 800x600 16 color
R       08  S-VGA 640x400 256 color
S       08  S-VGA 640x480 256 color
T       08  S-VGA 800x600 256 color
U       01  S-VGA 1024x768 2 color
V       31  S-VGA 1024x768 16 color
W       08  VGA 360x480 256 color
X       08  S-VGA 1024x768 256 color
Y       31  S-VGA 1280x1024 16 color
```


Ex.1007 / Page 718 of 1158
APPLE INC.

```
Z       08  S-VGA 1280x1024 256 color
L1      10  S-VGA 320x200 hicolor 15
s1      10  S-VGA 640x480 hicolor 15
t1      10  S-VGA 800x600 hicolor 15
x1      10  S-VGA 1024x768 hicolor 15
z1      10  S-VGA 1280x1024 hicolor 15
12      10  S-VGA 320x200 hicolor 16
s2      10  S-VGA 640x480 hicolor 16
t2      10  S-VGA 800x600 hicolor 16
x2      10  S-VGA 1024x768 hicolor 16
z2      10  S-VGA 1280x1024 hicolor 16
13      18  S-VGA 320x200 hicolor 24
S3      18  S-VGA 640x480 hicolor 24
T3      18  S-VGA 800x600 hicolor 24
X3      18  S-VGA 1024x768 hicolor 24
Z3      18  S-VGA 1280x1024 hicolor 24
```

The last change modified the compression algorithm to include 16- and 24-bit images. Each packed block has a similar format to the original byte-packed blocks of the previous compression method.

The first two bytes are 16-bit lengths of packed data, including the 4-byte header. The second two bytes are 16-bit lengths of unpacked data. What follows is a signed, 16-bit integer which, if negative, is a repeat count followed by a 16-bit repeat value (or 24-bit repeat value in 24-bit images). If the signed 16-bit integer is positive, then it is a run count of the number of 16-bit values (or 24-bit values) that follow. This repeats until the end of the packed block is reached.

BSAVE format

Pictor PC Paint 1.0 was developed for Mouse Systems in 1984 and supported only the BSAVE unpacked screen file format and stored only images using the 4-color CGA mode. PC Paint 1.5 supported a modified BSAVE format that allowed images larger than the screen to be stored and supported a rudimentary form of image compression. This revision 1.5 format was very short-lived, and very few image files of this format exist.

The header for the BSAVE format is as follows:

```
typedef struct _BsaveHeader
{
  BYTE Marker;              /* Marker value for packed data */
  WORD ScreenSegment;       /* PC screen memory segment */
  WORD ScreenOffset;        /* PC screen memory offset */
  WORD DataSize;            /* Size of screen data */

} BSAVEHEAD;
```


Ex.1007 / Page 719 of 1158
APPLE INC.

Marker is the byte value used to mark the start of an packed data run. This value is typically FDh or FEh if the image data is packed.

ScreenSegment is the segment address of the CGA video memory on the PC creating the BSAVE file. This value is typically B800h.

ScreenOffset is the offset address of the CGA video memory on the PC creating the BSAVE file. This value is typically 00h.

DataSize is the size of the screen image data stored in the file. This value is 16,384 for 4-color images, 32,768 for 16-color images, and 00h if the image data is packed. If DataSize is not 00h, then the image data immediately follows the header and is written literally to the PC's video memory. If the value is 00h, then two additional fields appear in the header:

```
WORD SizeOfData;        /* Total size of unpacked data in bytes */
WORD NumberOfBlocks;    /* Number of packed blocks */
```

SizeOfData is the total size of unpacked image data in bytes.

NumberOfBlocks is the number of packed data blocks stored in the file.

Following these fields is the image data in packed format.

At offset 8000 in each BSAVE file is the string "PCPAINT 1.0" or "PC Paint V1.5" indicating the format of the file. This ID string is followed by a byte indicating the current palette number and a second byte current border color number.

Clipping format

Early versions of Pictor PC Paint supported an image file format used to store image sections "clipped" from larger images. This clipping format uses the file extension .CLP and may store data in either a packed or unpacked form.

The header of the Pictor PC Paint clipping format is 11 or 13 bytes in length, depending upon how the image data is stored. The following 11 bytes appear in the header of every .CLP file:

```
typedef struct _ClpHeader
{
  WORD NumberOfBytes;       /* Size of the file, including header */
  WORD XSize;               /* Width of image in pixels */
  WORD YSize;               /* Length of image in pixels */
  WORD XOffset;             /* Left offset of image on display */
  WORD YOffset;             /* Top offset of image on display */
  BYTE BitsPerPixel;        /* Pixel depth */

} CLPHEAD;
```


Ex.1007 / Page 720 of 1158
APPLE INC.

NumberOfBytes is the total number of bytes in the clipped image.

XSize and YSize specify the size of the image in pixels.

XOffset and YOffset specify the location of the image on the display.

BitsPerPixel is the size of each pixel in bits. If this value is FFh, then the clipped image data is stored packed; otherwise, it is stored unpacked. If the BitsPerPixel value is not FFh, then the uncompressed image data follows the 11-byte header. If the BitsPerPixel value is FFh, then two additional fields appear in the header:

```
BYTE RealBits;              /* Number of bits per pixel */
BYTE Marker;                /* Marker byte value for packed data */
```

RealBits contains the number of bits per pixel (the value stored in BitsPerPixel if the data were not packed).

Marker is the value used to mark the beginning of a packed run.

The packed data then follows these fields. Packed data is stored as three bytes: the marker value, the run count, and the run value. A run value is repeated run count times. If a byte is read, and it does not contain the expected marker value, then the byte is written literally to the output.

Overlay format

Pictor PC Paint supports an image file format used to store collections of other images (usually .PIC and .CLP) in a single file. This Overlay format uses the file extension .OVR. It is also possible for .OVR files to contain other types of data besides image data. This is accomplished by appending a dummy 11-byte .PIC header to the data to fool PC Paint.

The header of an .OVR file is a list of each image file stored in the Overlay file. There is one entry in this list for each file stored in the .OVR file, plus an additional NULL entry to mark the end of the list. The format of this array is as follows:

```
typedef struct _PictureName
{
  WORD SizeOfList;          /* Size of the name list in bytes */
  struct _NameList          /* List of files in .OVR file */
  {
    LONG FileOffset;        /* Location of image in the .OVR file */
    CHAR Name[12];          /* Name of image file */
  } NameList[SizeOfList / sizeof(NameList)];

} PICTURENAME[];
```


Ex.1007 / Page 721 of 1158
APPLE INC.

SizeOfList is the total size of the name list in bytes, including the NULL entry at the end of the list.

NameList is an array of structures. There is one element per file stored, plus an additional NULL entry to mark the end of the list.

FileOffset is the location of this entry's file in the .OVR file. This offset is measured from the beginning of the .OVR file, and this value is 00h for the NULL list entry.

Name is the original filename of the file. This field is NULL padded for names shorter than 12 bytes and contains all NULLs for the NULL list entry.

The actual images stored in the .OVR file follows the NULL list entry.

As an example, let's say we have two files called IMAGE.PIC (2048 bytes in size) and IMAGE1.CLP (384 bytes in size) stored in an .OVR file. The internal format of the .OVR file would be as follows:

```
Field Name     Value
SizeOfList     48
FileOffset     50
Name           "IMAGE.PIC\0\0\0"
FileOffset     2097
Name           "IMAGE.CLP\0\0\0"
FileOffset     0
Name           "\0\0\0\0\0\0\0\0\0\0\0\0"
[ Image data for IMAGE.PIC starting at offset 50 ]
[ Image data for IMAGE.CLP starting at offset 2097 ]
```

For Further Information

For further information about the Pictor PC Paint format, see the specification included on the CD-ROM that accompanies this book.

The Pictor PC Paint image file format is supported by Paul Mace Software.

Paul Mace Software
Attn: Steven Belsky
400 Williamson Way
Ashland, OR 97520
Voice: 503-488-0224
FAX: 503-488-1549
BBS: 503-482-7435
WWW: *http://www.pmace.com/*

Ex.1007 / Page 722 of 1158
APPLE INC.

NAME:	Pixar RIB
ALSO KNOWN AS:	RenderMan Interface Bytestream, RIB
TYPE:	Scene description
COLORS:	Up to 24-bit
COMPRESSION:	NA
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Pixar
PLATFORM:	All
SUPPORTING APPLICATIONS:	RenderMan, modeling applications, others
SPECIFICATION ON CD:	Yes (summary description)
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	DKB, MTV, POV, QRT

USAGE: Storage of image scene descriptions for Pixar's RenderMan product.

COMMENTS: Although RIB is proprietary, RenderMan is a capable product available on a number of platforms, so an energetic aftermarket has developed. Enough applications besides RenderMan now read and write RIB files, which has pushed the format into prominence as a de facto standard for high-end rendering.

Overview

Pixar RIB files implement the RenderMan Interface Bytestream (RIB) Protocol, which was developed at Pixar to provide a "standard interface between photo-realistic modeling and rendering programs." Because in practice RIB files are supported by other applications mainly to provide output readable by Pixar's RenderMan application, this description would be considered disingenuous, were it not for the fact that RenderMan is so highly regarded.

RenderMan is available on a number of platforms and has a certain currency among sophisticated computer graphics artists and animators. As an application, it provides photorealistic rendering capability, through calls to a comprehensive library of functions. Thus the files resemble scripts, or a series of

Ex.1007 / Page 723 of 1158
APPLE INC.

function calls in a programming language. Each statement implements what Pixar calls a rendering primitive. A list of rendering primitives establishes a description of how a picture is to appear, without specifying how the rendering application should construct it.

File Organization

RIB files are written one byte at a time, and thus Pixar has avoided potential portability problems caused by byte sex differences. The RIB protocol implements a command language, and the data contained in the files can be either 7-bit ASCII or in a compressed binary form. The RIB protocol thus defines an abstract rendering model. In this sense, a RIB file takes the place of a rendering application (usually RenderMan). The user later applies a rendering application to the RIB file to produce an actual image. RIB files are streams of free-form data compatible with the abstract RIB rendering application.

Keeping this in mind, then, RIB files maintain a graphics state, which contains the information necessary to render a graphics primitive, such as color and the various coordinate mapping transformations. In some other rendering applications, "graphics state" refers to the set of attributes associated with any objects being rendered, but Pixar extends the terminology slightly.

RIB defines a number of 2D and 3D geometric primitives, some of them quite sophisticated.

File Details

The RIB rendering application is assumed to be an interpreter scanning a bytestream. To support this model, RIB files are contructed from a sequence of tokens. Tokens are delimited by a set of special characters (", #, [, and]), and the data stream may contain white space, defined as in the C language. Comments are strings preceded by a #.

Both signed real numbers and integers are supported, as are strings, and both also follow conventions similar to their counterparts in the C language.

Names, arrays, and parameter lists round out the data types defined in the specification. Names are usually text strings (or their binary counterparts) associated with RenderMan Interface requests, otherwise known as RenderMan commands. Alongside arrays and parameter lists, they allow the full specification of function calls to the rendering application.

Ex.1007 / Page 724 of 1158
APPLE INC.

A section of a RIB file might appear as follows:

```
Projection "perspective" "fov" [30.0]
Translate 0 1 0
Rotate 90 0 1 0
WorldBegin
Surface "wood" "roughness" [.3] "Kd" 1
Color [.2 .3 .9]
Polygon "P" [010 011 001 000]
WorldEnd
```

This is not an excerpt from a file, only an example of the kind of commands to be found in one.

For Further Information

For further information about the Pixar RIB format, see the article on the CD-ROM that accompanies this book; this article was prepared by Pixar specifically for this book. For additional information, contact:

Pixar
Attn: Ray Davis
1001 West Cutting
Richmond, CA 94804
Voice: 510-236-4000
FAX: 510-236-0388
Email: *rdavis@pixar.com*
WWW: *http://www.pixar.com/*

See the following site for information on RenderMan:

http://pete.cs.caltech.edu/RMR/index.html

See these references for additional information about Pixar RIB:

Pixar, *The RenderMan Interface Bytestream Protocol File Format*, Pixar, June 1990.

Pixar, *The RenderMan Interface*, Version 3.1, Pixar, September 1989.

Upstill, Steve, *The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics*, Addison-Wesley, Reading, MA, 1989.

The latter two documents are needed for a full understanding of the RIB format. The binary version of RIB is discussed in the following:

Pixar, *The RenderMan Interface*, Version 3.1, Appendix C, Pixar.

Ex.1007 / Page 725 of 1158
APPLE INC.

Plot-10

NAME:	Plot-10
ALSO KNOWN AS:	Tek Plot-10
TYPE:	Vector
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	NA
ORIGINATOR:	Tektronix
PLATFORM:	All
SUPPORTING APPLICATIONS:	NA
SPECIFICATION ON CD:	No
CODE ON CD:	Yes (one variant in pbmplus package)
IMAGES ON CD:	No
SEE ALSO:	None

USAGE: Tektronix terminal control, occasionally written to files instead of to the terminal.

COMMENTS: If you need this information, you really need it badly, and good luck. Plot-10 files are basically dumps of terminal commands, written on-the-fly to a file.

Overview

Plot-10 is associated with a series of graphics terminals manufactured by Tektronix, which were widely used for a certain period of time, primarily at scientific and laboratory sites, before cheap, high-resolution raster display terminals became widely available in the 1980s. Tektronix no longer supports these terminals except under contract. Many owners of Tektronix terminals, however, saw fit through the years to code applications that saved Tektronix terminal commands in local files. A typical application might acquire data from an experiment, for instance, and provide a real-time display of it on a Tektronix terminal. If this data was interesting in any way, it was saved. Unfortunately, because Tektronix provided no guidelines for this, the application developer was forced to make up his or her own format on the spot.

Ex.1007 / Page 726 of 1158
APPLE INC.

The result of this situation is that there are a lot of "Plot-10" formats in existence. Because most scientific programming has been in Fortran, you might consider that any data you find, on an old tape, for example, may be organized in accordance with the Fortran formatting conventions.

Tektronix technical support has access to the documents describing several terminal communication protocols from that era. These include IDL, STI, and TCS. Unfortunately, they appear to be in offsite backup storage, and we were unable to obtain them from Tektronix, because the company wanted to charge a hefty fee even to look for them. We hope that these documents might be made available to us in a future edition of this book.

For Further Information

For further information about Plot-10, you might try contacting:

Tektronix
Attn: Technical Support
26600 Southwest Parkway
Wilsonville, OR 97070-1000
Voice: 503-685-2418
WWW: *http://www.tek.com/*

Ask for information that will allow you to decode Plot-10 terminal dump files. Mentioning the IDL, STI, and TCS documentation may be helpful. Be sure to ask if there is a fee.

The pbmplus utilities included on the CD-ROM include an application to convert one of the Plot-10 versions.

Ex.1007 / Page 727 of 1158
APPLE INC.

PNG

NAME:	PNG
ALSO KNOWN AS:	Portable Network Graphic Format
TYPE:	Bitmap
COLORS:	1-bit to 48-bit
COMPRESSION:	LZ77 variant
MAXIMUM IMAGE SIZE:	2G×2G pixels
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	Thomas Boutell, Tom Lane, and many others
PLATFORM:	Any
SUPPORTING APPLICATIONS:	Many shareware and commercial packages
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes (in various packages)
IMAGES ON CD:	Yes
SEE ALSO:	GIF

USAGE: PNG is capable of losslessly storing bi-level to 48-bit truecolor image data. It is designed specifically for network image data transmission and storage.

COMMENTS: PNG is a well-designed and well-developed file format that is intended to replace CompuServe's GIF file format.

Overview

PNG (pronounced "ping") is a bitmap file format used to transmit and store bitmapped images. PNG supports the capability of storing up to 16 bits (grayscale) or 48 bits (truecolor) per pixel, and up to 16 bits of alpha data. It handles the progressive display of image data and the storage of gamma, transparency and textual information, and it uses an efficient and lossless form of data compression.

PNG is a very new format created with the intention of offering the graphics and imaging communities an alternative to CompuServe's Graphics Interchange Format (GIF) and the legalities associated with the "pay-to-implement" aspects of that format. (See the section called "LZW Legal Issues" in Chapter 9,

Ex.1007 / Page 728 of 1158
APPLE INC.

Data Compression.) The unofficial recursive derivation of the name "PNG" is "PNG's Not GIF."

PNG was designed with the goals that it be a simple format, one that is easy to implement and completely portable, and one that meets or exceeds all of the functional capabilities of the GIF format. It is also necessary that PNG be freely available and unencumbered by licensing fees and patent disputes.

PNG and GIF89a share the following features:

- Format organized as a data stream
- Lossless image data compression
- Storage of index-mapped images containing up to 256 colors
- Progressive display of interlaced image data
- Transparent key color supported
- Ability to store public and private user-defined data
- Independent from hardware and operating system

The following GIF features have been improved upon in PNG:

- Legally unencumbered method of data compression
- Faster progressive display interlacing scheme
- Greater extensibility for storing user-defined data

The following PNG features are not found in GIF:

- Storage of truecolor images of up to 48 bits per pixel
- Storage of gray-scale images of up to 16 bits per pixel
- Full alpha channel
- Gamma indicator
- CRC method of data stream corruption detection
- Standard toolkit for implementing PNG readers and writers
- Standard set of benchmark images for testing PNG readers

The following GIF features are not found in PNG v1.0:

- Capability of storing multiple images
- Support of storage of animation sequences

Ex.1007 / Page 729 of 1158
APPLE INC.

- Payment of a licensing fee required to sell software that reads or writes the GIF file format

Unlike most file formats, which are created by one or two programmers without much thought for the future expansion of the format, PNG was authored by a committee of interested implementors and GIF detractors (revision 1.0 of the PNG specification lists 23 authors) headed by Thomas Boutell.

PNG also holds the distinction of being one of the better designed file formats, allowing additional features to be added to the format without compromising existing functionality, and without forcing modifications to existing PNG-using software.

We are happy to report that the PNG specification is one of the most complete, well-thought-out, and well-written file format specifications yet examined by the authors of this book.

File Organization

A PNG format file (or data stream) consists of an 8-byte identification signature followed by three or more chunks of data. A chunk is an independent block of data conforming to a specifically defined structure. Chunks carry their own identification as to their internal format and are read sequentially from the beginning to the end of the file or data stream.

Several other file formats also use the concept of blocks or chunks of data. Most notably among these formats are GIF, IFF, and RIFF. Data in these formats is read serially from the beginning to the end of the file. This design makes it unnecessary to seek to different parts of the file using offset values; it also makes these types of formats ideal for use with networking and data transmission protocols. While each of these formats is usually thought of as a file format, it is more accurate to think of them as a data stream that is captured and stored to a file.

PNG defines four standard chunks, called *critical chunks*, that must be supported by every PNG file reader and writer. These chunks are the following:

The header chunk (IHDR)
: The header chunk contains basic information about the image data and must appear as the first chunk, and there must only be one header chunk in a PNG data stream.

Ex.1007 / Page 730 of 1158
APPLE INC.

The palette chunk (PLTE)

The palette chunk stores the colormap data associated with the image data. This chunk is present only if the image data uses a color palette and must appear before the image data chunk.

The image data chunk (IDAT)

The image data chunk stores the actual image data, and multiple image data chunks may occur in a data stream and must be stored in contiguous order.

The image trailer chunk (IEND)

The image trailer chunk must be the final chunk and marks the end of the PNG file or data stream.

Of these chunks, IHDR, IDAT, and IEND must appear in every PNG data stream.

Consider the following two basic types of PNG files, one with a color palette and one without:

Signature
IHDR Chunk
IDAT Chunk
IEND Chunk

Signature
IHDR Chunk
PLTE Chunk
IDAT Chunk
IEND Chunk

As you can see, the only difference in these two basic PNG formats is the presence of the palette chunk.

Optional chunks, called *ancillary chunks*, may be ignored by PNG file readers and need not be written by PNG file writers. However, failing to support ancillary chunks may leave a PNG reader unable to properly render many PNG

Ex.1007 / Page 731 of 1158
APPLE INC.

images. The images may appear too dark or too light, or the images may display in some other way not intended by the image's creator. It is therefore recommended that PNG-using software support the interpretation of most standard ancillary chunks (in particular, the Image Gamma chunk).

Together, the critical and ancillary chunks defined in the PNG specification proper are termed *standard chunks.* The people who maintain the PNG specification are also keeping a list of additional chunks, termed *special-purpose public chunks.* These chunks are expected to be less widely implemented than the standard chunks but may be of use for some applications. The list of special-purpose public chunks is expected to be extended from time to time. Applications may also define *private chunks* for their own purposes, if they wish to store data that need not be interpreted by other applications.

Here is a summary of all of the standard and special-purpose chunks defined by revision 1.0 of the PNG specification and associated documentation. The chunks in this list are arranged by the relative order (but not the only order) that they could appear in a PNG data stream.

TABLE PNG-1: *PNG Chunks*

Chunk Type	Multiple	Optional	Position
IHDR	No	No	First chunk
cHRM	No	Yes	Before PLTE and IDAT
gAMA	No	Yes	Before PLTE and IDAT
sBIT	No	Yes	Before PLTE and IDAT
PLTE	No	Yes	Before IDAT
bKGD	No	Yes	After PLTE and before IDAT
hIST	No	Yes	After PLTE and before IDAT
tRNS	No	Yes	After PLTE and before IDAT
oFFs	No	Yes	Before IDAT
pHYs	No	Yes	Before IDAT
sCAL	No	Yes	Before IDAT
IDAT	Yes	No	Contiguous with other IDATs
tIME	No	Yes	Any
tEXt	Yes	Yes	Any
zTXt	Yes	Yes	Any
fRAc	Yes	Yes	Any
gIFg	Yes	Yes	Any
gIFt	Yes	Yes	Any
gIFx	Yes	Yes	Any

Ex.1007 / Page 732 of 1158
APPLE INC.

Chunk Type	Multiple	Optional	Position
IEND	No	No	Last chunk

File Details

The PNG signature is eight bytes in length and contains information used to identify a file or data stream as conforming to the PNG specification.

```
typedef struct _PngSignature
{
    BYTE Signature[8];  /* Identifier (always 89504E470D0A1A0Ah) */
} PNGSIGNATURE;
```

Signature is eight bytes in length and contains the values 89h 50h 4Eh 47h 0Dh 0Ah 1Ah 0Ah ("\211PNG\r\n\032\n"). This seemingly random sequence of values has quite a few practical uses. The first byte value, 89h, is an 8-bit value used to identify the file as containing binary data. If the 8th bit were stripped from the file (courtesy of a 7-bit data channel), this value would then be changed to 09h and would provide an indication of how the file became corrupt.

The bytes that follow do the following:

- Allow the data stream to be visually identified ("PNG")
- Provide detection of a file transfer that alters the newline sequences ("\r\n" would become "\r", "\n" or "\n\r")
- Stops the listing of the PNG data stream on the MS-DOS operating system (Control-Z ["\032"])
- Detects file transfer CR/LF translation problems (the final newline)

Following the signature are three or more PNG data chunks. All PNG chunks have the same basic format and may contain a variable length payload of data.

```
typedef struct _PngChunk
{
    DWORD DataLength;   /* Size of Data field in bytes */
    DWORD Type;         /* Code identifying the type of chunk */
    BYTE  Data[];       /* The actual data stored by the chunk */
    DWORD Crc;          /* CRC-32 value of the Type and Data fields */
} PNGCHUNK;
```

DataLength is the number of bytes stored in the Data field. This value may be in the range 0 to $(2^{31})-1$.

Type is a 4-byte code identifying the type of data stored in the chunk. Each byte of this field may contain an uppercase or lowercase ASCII letter value (A–Z, a–z). For example, the chunk type IHDR would be identified by the value 69484452h in the Type field. PNG readers should treat Type codes as 32-bit literal values and not character strings. The fact that type codes are readable ASCII is primarily a convenience to humans.

Data is the actual data stored in the chunk. This field may be zero-length if a chunk has no associated data.

Crc is the CRC-32 value calculated for the Type and Data fields. This value is used to determine whether the data in the chunk has been corrupted. PNG uses the CRC algorithm defined by ISO 3309 and ITU-T V.42.

Chunks range in size from 12 bytes (no data) to $((2^{31})-1)+12$ bytes. Chunks are always aligned on byte boundaries and therefore never require any alignment padding.

Critical Chunks

This section describes the standard chunks that must be supported by every PNG file reader and writer.

Header chunk

The header chunk contains information on the image data stored in the PNG file. This chunk must be the first chunk in a PNG data stream and immediately follows the PNG signature. The header chunk data area is 13 bytes in length and has the following format:

```
typedef struct _IHDRChunk
{
    DWORD Width;          /* Width of image in pixels */
    DWORD Height;         /* Height of image in pixels */
    BYTE BitDepth;        /* Bits per pixel or per sample */
    BYTE ColorType;       /* Color interpretation indicator */
    BYTE Compression;     /* Compression type indicator */
    BYTE Filter;          /* Filter type indicator */
    BYTE Interlace;       /* Type of interlacing scheme used */
} IHDRCHUNK;
```

Width and Height are the width and height of the bitmap in pixels. These fields must each contain a value in the range 1 to $(2^{31})-1$.

Ex.1007 / Page 734 of 1158
APPLE INC.

BitDepth is the number of bits per pixel for indexed color images, and the number of bits per sample for gray-scale and truecolor images. Indexed color images may have a BitDepth value of 1, 2, 4, or 8. Gray-scale images may have BitDepth values of 1, 2, 4, 8, and 16. Only BitDepth values of 8 and 16 are supported for truecolor, truecolor with alpha data, and gray-scale with alpha data images.

ColorType indicates how the image data is to be interpreted. Valid values are 0 (gray-scale), 2 (truecolor), 3 (indexed color), 4 (gray-scale with alpha data), and 6 (truecolor with alpha data).

Compression indicates the type of compression used on the image data. Currently, the only valid value is 0, indicating that the Deflate compression method is used. Other compression methods may be defined in future extensions of PNG.

Filter specifies the type of filtering performed on the image data before compression. Currently, the only valid value is 0, indicating the adaptive filtering methods described in the PNG specification. Other filtering methods may be defined in future extensions of PNG. The filter value does not indicate whether the image data has been filtered; only the filter type byte at the start of each scan line can indicate that image data was filtered. Note that it is not a requirement that image data must be filtered before it is compressed.

Interlace indicates the interlacing algorithm used to store the image data—or more precisely, the transmission order of the pixel data. The values defined for this field are 0 (no interlacing) and 1 (Adam7 interlacing).

Palette chunk

The palette (PLTE) chunk is always found in PNG data streams that contain indexed-color image data; this is indicated when the Color field of the header chunk contains a value of 3. Truecolor PNG data streams (Color values 2 and 6) may also contain a palette chunk that non-truecolor display programs may use as a palette to quantize the image data. There will never be more than one palette chunk per PNG data stream.

The palette chunk may be from 3 to 768 bytes in length and has the following format:

```
typedef struct _PLTEChunkEntry
{
    BYTE Red;           /* Red component (0 = black, 255 = maximum) */
    BYTE Green;         /* Green component (0 = black, 255 = maximum) */
    BYTE Blue;          /* Blue component (0 = black, 255 = maximum) */
```


Ex.1007 / Page 735 of 1158
APPLE INC.

```
} PLTECHUNKENTRY;

PLTECHUNKENTRY PLTEChunk[];
```

PLTEChunk is an array containing 1 to 256 PLTECHUNKENTRY elements. Each PLTECHUNKENTRY contains three fields, Red, Green, and Blue, which store the red, green, and blue color values for that palette entry respectively.

Image Data chunk
The Image Data (IDAT) chunk stores the actual image data. The image data is always compressed, as required by the PNG specification. The image data may be stored in multiple contiguous IDAT chunks to make it easier for a PNG writer to buffer the compressed image data. There are no boundaries in the compressed data stream, so IDAT chunks may range in size from 0 to (2^{31})-1 bytes in length.

Image Trailer chunk
The final chunk in a PNG data stream is the Image Trailer (IEND) chunk. This chunk does not contain any associated data.

Ancillary Chunks

PNG v1.0 defines 10 ancillary chunks that may appear in a PNG data stream. Several of these chunks provide support for storing information that may be required for proper interpretation of the image data (such as the Image Gamma chunk). A brief description of the format of the Data field of each of these chunks is given below. Refer to the PNG specification on the CD-ROM for more detailed information on these chunks.

Background Color chunk
The Background Color chunk specifies the background color of the image. Note, however, that PNG readers may disregard this chunk and use any background color value they choose.

The data format of the background color chunk varies depending upon the format of the image data, as indicated by the ColorType field in the IHDR chunk. For an indexed-color image (ColorType value 3), the data is a single byte containing the index of the palette color to use as the background:

```
typedef struct _bKGDChunkEntry
{
    BYTE Index;     /* Index of background color in palette */
} BKGDCHUNKENTRY;
```


Ex.1007 / Page 736 of 1158
APPLE INC.

For gray-scale data, with or without alpha channel data (ColorType values 0 and 4), this chunk stores a 2-byte value specifying the gray level to be used as the background value:

```
typedef struct _bKGDChunkEntry
{
    WORD Value;    /* Background level value */
} BKGDCHUNKENTRY;
```

For truecolor images, with or without alpha channel data (ColorType values 2 and 6), the background chunk stores three 2-byte values specifying the RGB color used for the background:

```
typedef struct _bKGDChunkEntry
{
    WORD Red;    /* Red background sample value */
    WORD Green; /* Green background sample value */
    WORD Blue;  /* Blue background sample value */
} BKGDCHUNKENTRY;
```

Primary Chromaticities and White Point chunk

The Primary Chromaticities and White Point chunk stores information on RGB values based on the 1931 CIE XYZ colorspace. Only the x and y chromaticities are specified, and they are represented by values multiplied by 100,000.

```
typedef struct _cHRMChunkEntry
{
    DWORD WhitePointX;    /* White Point x value */
    DWORD WhitePointY;    /* White Point y value  */
    DWORD RedX;           /* Red x value */
    DWORD RedY;           /* Red y value */
    DWORD GreenX;         /* Green x value */
    DWORD GreenY;         /* Green y value */
    DWORD BlueX;          /* Blue x value */
    DWORD BlueY;          /* Blue y value */
} CHRMCHUNKENTRY;
```

Image Gamma chunk

The Image Gamma chunk stores the original gamma value of the image with respect to the original scene. The stored value is the gamma multiplied by 100,000. Note that it is "strongly" recommended by the PNG authors that decoders implement the gamma chunk.

```
typedef struct _gAMAChunkEntry
{
    DWORD Gamma;     /* Gamma value */
} GAMACHUNKENTRY;
```


Ex.1007 / Page 737 of 1158
APPLE INC.

Image Histogram chunk

The Image Histogram chunk stores data on the approximate usage frequency of each color in a palette. This chunk contains an array of 2-byte elements, one element per entry in the color palette.

```
typedef struct _hISTChunkEntry
{
    WORD Histogram[];     /* Histogram data */
} HISTCHUNKENTRY;
```

Physical Pixel Dimension chunk

The Physical Pixel Dimension chunk specifies the intended resolution for display of the image.

```
typedef struct _pHYsChunkEntry
{
   DWORD PixelsPerUnitX;     /* Pixels per unit, X axis */
   DWORD PixelsPerUnitY;     /* Pixels per unit, Y axis */
   BYTE  UnitSpecifier;      /* 0 = unknown, 1 = meter */
} PHYSCHUNKENTRY;
```

Significant Bits chunk

The Significant Bits chunk indicates the bit depth of the original image data. If a PNG writer needs to store image data of an unsupported bit depth, the data must be padded to the next greater supported bit depth to be stored. For example, to store RGB data with a resolution of five bits per sample (RGB555) using PNG, the image data would first need to be scaled up to an 8-bit sample depth (RGB888). The Significant Bits chunk would then store the bit depths of the original image data components.

There are four possible formats of the data in this chunk; the one used depends upon the format of the image data (as indicated by the ColorType field in the IHDR chunk):

```
/* Gray-scale (ColorType 0) image data */
typedef struct _sBITChunkEntry
{
    BYTE GrayscaleBits;     /* Gray-scale (ColorType 0) significant bits */
} SBITCHUNKENTRY;

/* Truecolor or indexed-color (ColorType 2 or 3) image data */
typedef struct _sBITChunkEntry
{
    BYTE RedBits;      /* Red significant bits */
    BYTE GreenBits;    /* Green significant bits */
    BYTE BlueBits;     /* Blue significant bits  */
} SBITCHUNKENTRY;
```


Ex.1007 / Page 738 of 1158
APPLE INC.

```
/* Gray-scale with alpha channel (ColorType 4) image data */
typedef struct _sBITChunkEntry
{
    BYTE GrayscaleBits; /* Gray-scale significant bits */
    BYTE AlphaBits;     /* Alpha channel significant bits */
} SBITCHUNKENTRY;

/* Truecolor with alpha channel (ColorType 6) image data */
typedef struct _sBITChunkEntry
{
    BYTE RedBits;     /* Red significant bits */
    BYTE GreenBits;   /* Green significant bits */
    BYTE BlueBits;    /* Blue significant bits  */
    BYTE AlphaBits;   /* Alpha channel significant bits */
} SBITCHUNKENTRY;
```

Textual Data chunk

The Textual Data chunk is typically used to store human-readable information, such as the name of the author of the image and the copyright notice, within a PNG file. The data of this chunk has the following structure:

```
typedef struct _tEXtChunkEntry
{
    char Keyword[];      /* Type of information stored in Text */
    BYTE NullSeparator;  /* NULL character used a delimiter */
    char Text[];         /* Textual data */
} TEXTCHUNKENTRY;
```

Keyword is a field of character data with a length of 1 to 79 bytes. This field may contain any printable Latin-1 character except NULL. Spaces are also allowed.

NullSeparator is a single byte initialized to 0. This field acts as a delimiter to separate the Keyword and Text fields.

Text is a field of character data that is the actual textual data stored in the chunk. The length of this field is determined from the value of the Data-Length field in the chunk header.

The value of Keyword indicates intellectual content information associated with the textual data stored in the Text field. The following keywords are defined by PNG v1.0:

Title
Author

Ex.1007 / Page 739 of 1158
APPLE INC.

Description
Copyright
Creation Time
Software
Disclaimer
Warning
Source
Comment

Additional keywords can be defined though public registration or can be invented by individual applications.

Image Last-Modification Time chunk

The Image Last-Modification Time chunk stores the time the image was last modified (rather than the time the image was first created). The format of this chunk's data is as follows:

```
typedef struct tIMEChunkEntry
{
    WORD Year;      /* Year value (such as 1996) */
    BYTE Month;     /* Month value (1-12) */
    BYTE Day;       /* Day value (1-31) */
    BYTE Hour;      /* Hour value (0-23) */
    BYTE Minute;    /* Minute value (0-59) */
    BYTE Second;    /* Second value (0-60) */
} TIMECHUNKENTRY;
```

Transparency chunk

The Transparency chunk stores a transparency value (key color) for a PNG image that does not contain associated alpha-channel data. Truecolor and gray-scale pixel values that match the transparency color are to be considered transparent (alpha value of 0), and all other pixels are regarded as opaque.

Indexed color images store an array of alpha values, up to one per element in the palette. These transparency values are treated as full alpha values. Any palette entries that do not have a corresponding transparency value are considered to have a default value of 255 (fully opaque).

There are three possible formats of the data in this chunk, depending on the format of the image data, as indicated by the ColorType field in the IHDR chunk:

Ex.1007 / Page 740 of 1158
APPLE INC.

```
/* Gray-scale (ColorType 0) image data */
typedef struct _tRNSChunkEntry
{
    WORD TransparencyValue;     /* Transparent color */
} TRNSCHUNKENTRY;

/* Truecolor (ColorType 2) image data */
typedef struct _tRNSChunkEntry
{
    WORD RedTransValue;         /* Red sample of transparent color */
    WORD GreenTransValue;       /* Green sample of transparent color */
    WORD BlueTransValue;        /* Blue sample of transparent color */
} TRNSCHUNKENTRY;

/* Indexed-color (ColorType 3) image data */
typedef struct _tRNSChunkEntry
{
    BYTE TransparencyValues[];  /* Transparent colors */
} TRNSCHUNKENTRY;
```

Compressed Textual Data chunk

The Compressed Textual Data chunk is used to store a large block of textual data in a compressed format. This chunk has the same format as the Textual Data chunk, but the Text field contains data compressed using the Deflate compression method used by PNG for compressing image data.

Image Data

PNG image data is laid out as a bitmap with scan lines running from left to right and from top to bottom. Pixels are always packed into scan lines and do not use any filler bits to maintain byte boundary alignment between pixels. Pixels less than eight bits in size are packed into bytes with the leftmost pixel occupying the most significant bits of the byte.

Scan lines always begin on byte boundaries and must always be padded to end on a byte boundary if necessary. Scan lines are also prepended with an extra "filter type" byte used during image compression and decompression. This extra byte indicates the type of filtering algorithm used to process the scan line. This byte is always present, even if filtering is not used, and it is not considered to be part of the actual image data.

Image data up to eight bits in depth may have its values mapped to a color palette or may be stored directly in the bitmap data as gray-scale values. Truecolor pixels are always stored as three separate color samples, one each for red, green, and blue. A fourth sample for alpha-channel data may also be included with each truecolor pixel.

Ex.1007 / Page 741 of 1158
APPLE INC.

Gray-scale and indexed color bitmaps contain one sample per pixel and are referred to as single-sample pixels. Every sample in an image is always the same size. This size is called bit depth and is the number of bits in the sample. A single component may range from 1 to 16 bits in depth. For indexed color data the bit depth indicates the maximum number of colors in the palette. PNG does not specifically define, nor preclude, the use of bi-level bitmaps.

Multi-sample pixels contain two or more samples per pixel. Samples in multi-sample pixels may either be 8 or 16 bits in depth, and all of the samples in a pixel must be the same size. Multi-sample pixels may range from 16 to 64 bits in depth.

For example, a typical gray-scale pixel contains a single 8-bit sample. A typical 24-bit RGB pixel contains three 8-bit samples, while a not-so-typical 64-bit RGBA pixel would contain four, 16-bit samples. Note that both single and multi-sample pixels that have samples of bit depths other than 8 or 16 are required to use a sample of the next greater size. For example, to store a 10-bit component, you would use a 16-bit sample. The unused bits in the sample are filled either by setting to zero (not recommended for bit depths less than 8 bits/sample, but for higher bit depths, zero-fill can significantly increase compression) or by linearly scaling the sample up to fill the range of possible values (recommended). The PNG authors recommend a quick method of scaling up by replication of the leftmost significant bits of the sample.

Alpha Channel

Gray-scale and truecolor images ranging from 8 to 16 bits in depth may also contain unassociated alpha-channel data called an *alpha mask*. If alpha mask data is used, each truecolor or gray-scale pixel will have an additional sample that stores the alpha-channel value for that pixel. Indexed color images may store alpha-channel data using the Transparency chunk.

An alpha value indicates the transparency level of that pixel. The minimum value of the bit depth (always 0) indicates complete transparency, and the maximum value for the bit depth indicates full opacity. If no alpha mask is stored, the pixel is assumed to be fully opaque.

Interlacing

PNG image data is typically stored as a series of scan lines starting with the first line at the top of the image and progressing sequentially to the last line at the

Ex.1007 / Page 742 of 1158
APPLE INC.

bottom of the image. PNG image data may also be stored in a specific interlace pattern to allow a progressive display of the image data from a low resolution to a full resolution display.

Progressive display is most useful when receiving a PNG file over a slow transmission link (like the one that connects your Web browser to the Internet). The gradual "fade in" effect typically allows a user to discern the content of the image before it has displayed in its entirety. This feature is very useful if the image is a menu on a Web page or a picture that you don't wish to waste the time downloading.

It is also a requirement that all PNG readers be able to interpret interlaced image data, although PNG viewers need not support the ability to perform a progressive display.

A typical interlace scheme, such as that used by GIF, simply rearranges the order in which the scan lines are stored. For example, rather than storing lines sequentially as 0, 1, 2, 3, 4, 5, 6,..., an interlace scheme might store scan lines as 0, 8, 4, 9, 2, 10, 5,... in the file. GIF uses this type of interlacing scheme and stores (or transmits) image data in four passes of 1/8, 1/8, 1/4, and 1/2.

PNG takes a somewhat different approach by interlacing images using a 7-pass scheme known as Adam7, after its inventor Adam M. Costello. Adam7 uses the first six passes to build up all even-numbered scan lines (0, 2, 4, 6,...) and the final (seventh) pass to fill in the remaining odd-numbered scan lines (1, 3, 5, 7...) in the image.

Rather than containing the pixels for entire scan lines, the initial six passes contain specific pixels of only every other scan line. The first two passes each contain 1/64th of the pixels in the image. The third pass contains 1/32nd, the fourth pass 1/16th, the fifth pass 1/8th, the sixth pass 1/4th, and the seventh (final) pass 1/2 of the image data.

The image itself is built up on the display, first as 8×8 squares, then 4×8 rectangles, then 4×4 squares, then 2×4 rectangles, then 2×2 squares, and then 1×2 rectangles. The final pass fills in the pixels of the odd-numbered scan lines.

Adam7 interlacing allows the progressive buildup of pixels to appear much more quickly on the display than it would if entire scan lines were displayed. The pixels in the image are also displayed in a more dispersed pattern, allowing the human eye to discern the typical interlaced PNG image after only 20 to 30 percent of the image data has been received, compared with the 50 percent or more needed from the GIF interlacing scheme.

Ex.1007 / Page 743 of 1158
APPLE INC.

Note, however, that PNG's interlacing method does trade off a bit of size for speed. The GIF interlacing scheme simply rearranges the storage order of scan lines, and does not have much impact on the storage space per scan line. In the PNG scheme, each pass except the last carries non-adjacent pixels; for example, pass 1 contains every 8th pixel from every 8th line.

On the average, there is less correlation between such pixels than there is between adjacent pixels. This means that compression is less effective on the interlaced data than it is on sequentially presented data, so the resulting file is bigger. Typically, an interlaced file will be up to 10 percent larger than an equivalent non-interlaced file. For most applications where interlacing is useful, this price is well worth paying in exchange for faster buildup of a useful image.

Adam7 interlacing is performed using the filter pattern below. Uncompressed PNG image data is interlaced by first reproducing this 8×8 map over the entire bitmap. The image data is then scanned seven times, and the pixel values indicated by the map are read to determine what pixel values are stored or transmitted during each pass.

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

Data compression

PNG image data is always stored in a compressed format. Image data is compressed using a prediction of pixel values with differences compressed by a variation of the Deflate compression method. Deflate was created by Phil Katz and is used in the *pkzip* file archiving utility. This lossless compression method is fast, well-documented, and freely available, and it is supported by a large number of operating platforms.

Ex.1007 / Page 744 of 1158
APPLE INC.

Deflate is a variation of the LZ77 compression algorithm originally patented (4,464,650) by Lempel, Ziv, Cohen, and Eastman in 1981. Deflate uses a variably sized sliding window and sorted hash tables to identify data patterns and compresses them using Huffman encoding. PNG uses a variation of Deflate that does not use sorted hash tables, and is therefore not subject to any patent claims or licensing agreements.

Image data may be optionally filtered before it is compressed. Filtering normalizes the byte values in a scan line, allowing the Deflate compression algorithm to be more effective and producing smaller compressed data.

All filtering algorithms are applied to the bytes in a scan line rather than to the pixels. Any alpha channel data present in the scan-line data is also filtered. And because a single filtering algorithm may not be effective when applied to an entire image, each scan line is filtered separately, and any or no filter may be applied to any line.

Several types of predictive filters are defined for use on PNG image data. Filtering is applied to the data before it is compressed, and the reverse of the filter is applied after the image data is decompressed, restoring the data to its original values. All of the PNG filters are therefore completely reversible and lossless.

The Sub filter stores the difference between a byte value of the current pixel and the value of the same byte in the previous pixel (the predictor). This method allows the same samples across multi-sample pixels to always be differenced. This is the same predictor algorithm used by the TIFF image file format.

The Up filter stores the difference between the byte in the current pixel and the related byte in the same pixel of the previous scan line. The Average filter stores the differences between the current pixel from the average of the pixels just above and to the left.

The Paeth filter uses a linear function to compute a value. The closest matching left, up, or upper left byte value is used as the predictor.

For Further Information

The complete PNG specification, special-purpose public chunks documentation, PNG implementation toolkit, and sample PNG images are available on the CD-ROM.

Ex.1007 / Page 745 of 1158
APPLE INC.

PNG (cont'd)

The current PNG specification can be found at the following Web page:

http://sunsite.unc.edu/boutell/png.html

and the following FTP sites:

ftp://swrinde.nde.swri.edu/pub/png/documents/
ftp://ftp.uu.net:/graphics/png/documents/

Your best Web source for PNG information and resources resides on Greg Roelofs' PNG group's homepage:

http://quest.jpl.nasa.gov/PNG/

Questions about PNG may be asked on the *comp.graphics.misc* newsgroup, or via email to:

png-info@uunet.uu.net

or directed to the principal author of the PNG specification:

Thomas Boutell
Email: *boutell@boutell.com*

PNG developers may join the PNG mailing list. Send email to *png-info@uunet.uu.net.*

Other PNG mailing lists include:

png-list@dworkin.wustl.edu	General PNG discussion
png-announce@dworkin.wustl.edu	Announcements related to PNG
png-implement@dworkin.wustl.edu	Implementation discussion

These lists contain a general discussion of PNG, announcements related to PNG, and discussions regarding PNG implementation. To find out more about the mailing list server, send email to *majordomo@dworkin.wustl.edu* with the word "help" (and nothing else) in the message body.

The official PNG FTP archive is:

ftp://ftp.uu.net/graphics/png/

A reference implementation in portable C of a PNG reader and writer is available at:

ftp://ftp.uu.net/graphics/png/src/

Ex.1007 / Page 746 of 1158
APPLE INC.

Test PNG images for your benchmarking pleasure are available from:

ftp://ftp.uu.net/graphics/png/images/

PNG materials, including a mirror of everything in *ftp://ftp.uu.net/graphics/png/* can also be found at:

ftp://swrinde.nde.swri.edu/pub/png/

All programs on this site are in beta test and should be used carefully. In the case of questionable implementation, the specification is to be considered correct and the code in error.

Group 42 is the author of the LIBPNG support library for developers using the PNG file format. Their Web page contains a developer's section that includes the LIBPNG library, PNG format specification, Compression Library, and Image Test Suite. A freeware version of this library is currently available. Group 42 may be reached at:

Group 42, Inc.
Voice: 800-520-0042
Voice: 513-831-3400
Email: *info@group42.com*
WWW: *http://www.group42.com/*

A good overview of PNG can be found in:

Crocker, Lee Daniel, "PNG: The Portable Network Graphic Format," *Dr. Dobb's Journal*, vol. 20, no. 232, July 1995, pp. 36–44.

The code for the above article is available at:

ftp://ftp.mv.com/pub/ddj/1995/1195.07/ptot.zip

A rather CompuServe-biased official press release is at:

http://www.compuserve.com/new/news_rel/png2.html

Ex.1007 / Page 747 of 1158
APPLE INC.

POV

NAME:	POV
ALSO KNOWN AS:	POV-Ray, Persistence of Vision
TYPE:	Vector
COLORS:	Unlimited
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	ASCII
ORIGINATOR:	POV-Team
PLATFORM:	MS-DOS, Macintosh, Amiga, UNIX
SUPPORTING APPLICATIONS:	POV-Ray
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	TGA, GIF

USAGE: POV is a scene-description language used to mathematically represent image data that is rendered by the POV-Ray ray tracing engine.

COMMENTS: One of the few formats described in this book that is created entirely by the human hand.

Overview

The POV (Persistence of Vision) format is used to store the scene description language used by the POV-Ray (Persistence of Vision Raytracer) software package. This format is very similar to other vector-based animation and ray tracing formats.

POV-Ray is capable of creating photorealistic, 3D images using a graphical rendering technique called ray tracing. Simple shapes, textures, lights, and properties are available to render images. POV-Ray also supports many advanced ray tracing features, such as Bezier patches, blobs, height-fields, and bump and material mapping.

Images are created by a POV-Ray user writing mathematical code in an editor and rendering the image using the ray-tracing engine. POV-Ray then writes out

Ex.1007 / Page 748 of 1158
APPLE INC.

the rendered image to a file using either the TGA (24-bit) or GIF (8-bit) raster file formats.

Because the process of creating a POV-Ray image is similar to the way in which a programmer writes and compiles code (rather than the way in which an artist uses a paint program), POV scene-description files are more akin to source code files than to typical graphics format files.

The information stored in a POV language file is a set of descriptions of the scenes in the rendered image. The POV scene description language may therefore be thought of as a PostScript-like page description language for ray-traced images.

File Organization

POV scene files contain three types of elements: camera, object, and light source. A *camera* is the angle of the view into the image. Different perspectives of the scene may be rendered by changing the angle of the camera view and the position of the camera within the image. An *object* is a visible shape that can be seen in the rendering. A *light source* is an invisible object that illuminates the visible objects in the scene. Each scene may have multiple objects and lights, but only one camera.

Let's look at a minimal POV scene file and examine the elements:

```
//
// The canonical red ball on a green floor
//
camera {
        location <0 1 -2>
        look_at <0 1 2>
}
object {
        sphere { <0 1 2> 1 }
        texture { color red 1 phong 1 }
}
object {
        plane { <0 1 0> 0 }
        texture { color green 1 }
}
object {
        light_source { <3 3 -3> color red 1 green 1 blue 1 }
}
```

In this example we see a camera and three objects: a sphere, a plane, and a light source. The camera object contains two statements. The first, *location*, indicates the position of the camera within the rendering. It is followed by a

Ex.1007 / Page 749 of 1158
APPLE INC.

parameter list containing the values of the X, Y, and Z coordinates of the physical camera location with respect to the origin point at location 0 0 0. In the example, the values 0, 1, -2 indicate that the camera is centered horizontally (X = 0), one unit up (Y = 1), and two units back (Z = -2) from the origin.

The second statement, *look_at*, specifies the direction the camera is pointing and the point of its focus. These points are also described using 3-dimensional coordinate values. In the example, the values 0, 1, 2 indicate that the camera is looking forward (X = 0), one unit up (Y = 1), and focused at a point two units in front of it (Z = 2).

The first object in the scene is the sphere. Being a visible object, this object contains a shape description and a material description. The shape description is a *sphere* statement, indicating the position of the sphere in the scene and the size of the radius. In the example, the sphere is located at the coordinates 0, 1, 2 and has a radius of 1 unit. The material description is a *texture* statement specifying that the sphere is red in color and has a phong highlighting intensity value of 1.

The next object is the plane. The *plane* statement defines a plane with its surface normal along the Y axis and offset 0 units from the Y axis. The *texture* statement indicates that the plane is green in color.

The last object is the light source object. The parameters of the *light_source* statement indicate the position of the (omnidirectional) light source and its color (white).

As you can see, objects and cameras are written using code similar to that used in many computer languages. Each statement begins with a keyword and is followed by a function body with zero or more values or statements enclosed in braces. For example:

```
sphere { <0 1 0> -4 }
box { <-2 -4 -3.5> <2.5 5.0 2.5> }
color_map { [0.0 0.2 color red 1] [0.2 0.4 color red .5] }
camera { location <0,0,0> look_at <0 1 2> }
```

In this example, the sphere object has a body containing two values, a vector coordinate parameter list and a floating-point value. The body of the box object contains two parameter lists. The color map body contains a two-element array. The camera object contains two statements, each statement in turn containing a parameter list.

There are many objects supported by the POV scene-description language and they are fully detailed in the POV documentation.

Ex.1007 / Page 750 of 1158
APPLE INC.

File Details

POV files are normal ASCII text and do not contain a header or any binary information. Text information in a POV file is case-sensitive. Lowercase words are reserved language keywords. Uppercase words are used for naming data constructs. A single character in double quotes is a literal character.

Comments in POV files use with the Standard C comment tokens /* */ or the C++ comment token //. The Standard C tokens may be nested. Data may be inserted into a POV file using the #include declaration, as follows:

```
#include "filename.inc."
```

POV include files normally have the extension ".inc" and contain information that is shared between multiple POV renderings.

One nice feature of POV is the ability to pre-define a set of data that is used repeatedly, similar to the type definition (*typedef*) feature found in the C language.

Predefinition is accomplished using the *#declare* keyword. In this example, we declare a texture with color parameters values for white:

```
texture { color red 1 green 1 blue 1 }
```

We can predefine the color white for visual clarity in meaning and for later reuse in the file:

```
#declare WHITE = color red 1 green 1 blue 1
texture { color WHITE }
```

For Further Information

For further information about the POV format, see the specification included on the CD-ROM that accompanies this book. Information about the POV-Ray description language can be found in the POV-Ray package itself. Versions of POV-Ray for MS-DOS, UNIX, Apple Macintosh, Commodore Amiga, and other computers are available from CompuServe, America Online, the Internet, and many BBSs.

See these POV-Ray sites:

ftp://alfred.ccs.carleton.ca/
http://www.povray.org/
ftp://ftp.povray.org/

Ex.1007 / Page 751 of 1158
APPLE INC.

POV (cont'd)

The Waite Group has published an excellent book on ray tracing on the PC using POV-Ray. This book comes with the POV-Ray tracing software for the PC with many scenes and objects and also contains a nice introduction to the art and concepts of ray tracing. See the following:

Wells, Drew and Young, Chris, *Ray Tracing Creations: Generate 3D Photo-Realistic Images on the PC*, Waite Group Press, Corte Madera, CA. 1993.

If you have questions about POV-Ray, you can contact:

Chris Young
POV-Ray Team Coordinator
Email: *76702.1655@compuserve.com*

Ex.1007 / Page 752 of 1158
APPLE INC.

Presentation Manager Metafile

NAME:	Presentation Manager Metafile
ALSO KNOWN AS:	MET
TYPE:	Metafile
COLORS:	Unlimited
COMPRESSION:	RLE
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	Microsoft Corp., IBM
PLATFORM:	OS/2
SUPPORTING APPLICATIONS:	Various under OS/2 Presentation Manager
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	Microsoft Windows Metafile, OS/2 Bitmap

USAGE: Storage and transport of graphics information associated with the OS/2 Presentation Manager GUI. Seldom found ouside the OS/2 environment.

COMMENTS: A complex format mainly consisting of aliased calls to Presentation Manager supporting libraries. Difficult to support outside of that environment.

Overview

Presentation Manager Metafile (MET) files are used to store vector- and bitmap-format image data in memory or in disk files, for later playback to an output device. Although the Presentation Manager Metafile format is specific to IBM's Presentation Manager for OS/2, many third-party applications support this format as a method for interchanging data between applications under OS/2. Because of the confusion in the market engendered by the IBM-Microsoft split, and the subsequent increase in the installed base of Microsoft Windows, Presentation Manager Metafile has found little support in the larger market, even though the OS/2 installed base is substantial.

Ex.1007 / Page 753 of 1158
APPLE INC.

File Organization

Presentation Manager Metafiles consist of a sequence of what IBM calls structured fields, which are followed by the actual data. Structured fields and the associated data are organized into one or more functional components, which are large blocks of data—documents, for instance, or complex graphics objects. These functional components are delimited by "begin-component" and "end-component" structured fields.

File Details

Structured fields start with the following header:

```
typedef struct _MetHeader
{
  CHAR Length[2];
  CHAR ID[3];
  BYTE Flags;
  CHAR SegSeqNum[3];

} MetHeader;
```

Length is the length of the field, in bytes.

ID is a field identifier.

Flags contains Boolean information related to the disposition of the field by the rendering application. Currently this is always 0.

SegSeqNum contains what IBM calls a segment sequence number. Again, this is currently always 0.

Following the header structure, which is common to all structured fields, is information that IBM calls positional information. This information extends the header; its exact nature depends on the actual structured field. Following these positional fields are what IBM calls triplets, which consist of a short header of the following form:

```
typedef struct _Triplet
{
  BYTE Length;
  BYTE ID;

} Triplet;
```

Length is the length in bytes of the triplet header and the following data.

Ex.1007 / Page 754 of 1158
APPLE INC.

ID contains a value identifying the triplet.

Following this header is the actual data associated with the triplet. The bulk of the data found in the metafile is located here.

Structured fields defined in the documentation are listed below.

Field Name	Field ID	Parameters	Triplets
Begin Document	D3A8A8	10 bytes	3
Begin Resource Group	D3A8C6	08 bytes	0
Begin Color Attribute	D3A877	08 bytes	0
Color Attribute Table	D3B077	03 bytes	(varies)
End Color Attribute Table	D3A977	08 bytes	0
Begin Image Object	D3A8FB	08 bytes	0
Begin Resource Group	D3A8C6	08 bytes	0
End Resource Group	D3A9C6	08 bytes	0
Begin Object Environment Group	D3A8C7	08 bytes	0
Map Color Attribute Table	D3AB77	02 bytes	2
Image Data Descriptor	D3A6FB	09 bytes	0
Image Picture Data	D3EEFB	(varies)	(varies)
End Image Object	D3A9FB	08 bytes	0
Begin Graphics Object	D3A8BB	08 bytes	0
Map Coded Font	D3AB8A	02 bytes	3
Map Data Resource	D3ABC3	02 bytes	2
End Object Environment Group	D3A9C7	08 bytes	0
End Graphics Object	D3A9BB	08 bytes	0
End Resource Group	D3A9C6	08 bytes	0
End Document	D3A9A8	08 bytes	0

Perhaps the most common structured field is the Graphics Data Descriptor, which contains the actual graphics data:

Graphics Data Descriptor	D3A6BB

Parameter information associated with the Graphics Data Descriptor provides an elaborate description of the following data, which takes the place of what might be encoded in another format as a long, complex header. Its length varies, but it may be several hundred bytes long. Please see the specification included on the CD-ROM for details and further information. (Note that the document appears to contain an incomplete list of the structured fields likely to be found in a Presentation Manager Metafile.)

Ex.1007 / Page 755 of 1158
APPLE INC.

For Further Information

For further information about the Presentation Manager Metafile format, see the specification included on the CD-ROM that accompanies this book.

Presentation Manager Metafile is also documented in the following IBM publication:

IBM Corporation, *OS/2 2.0 Technical Library Presentation Manager Programming Reference Volume III, Part Number 10G627.*

Relevant information is contained in Appendixes D, G, and F of the IBM publication. This document is available for purchase through your local IBM dealer or salesperson.

Support responsibility for OS/2 is now solely in the hands of IBM. For information, contact:

IBM Corporation
Attn: Independent Vendor League
150 Kettletown Road
Southbury, CT 06488
Voice: 203-266-2000
WWW: *http://www.ibm.com/*

OS/2 and the Presentation Manager Metafile format originated at Microsoft, and some documentation is still available there. You may be able to get some information by contacting:

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
Voice: 206-882-8080
Voice: 800-426-9400
FAX: 206-883-8101
WWW: *http://www.microsoft.com/*

Also see the following resources:

http://axion.physics.ubc.ca/os2/os2.html
OS/2 Resource listing

ftp://ftp-os2.cdrom.com/
OS/2 Archive at Walnut Creek CD-ROM

Ex.1007 / Page 756 of 1158
APPLE INC.

You might also be able to find some helpful information at the OS/2 shareware BBS:

Voice: 703-385-0201
BBS: 703-385-4325
WWW: *http://www.os2bbs.com/*
Telnet: *bbs.os2bbs.com*

Ex.1007 / Page 757 of 1158
APPLE INC.

PRT

NAME:	PRT
ALSO KNOWN AS:	Parallel Ray Trace
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Kory Hamzeh
PLATFORM:	All
SUPPORTING APPLICATIONS:	PRT ray trace application, others
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	NFF, POV, QRT, Radiance

USAGE: Description of 3D scenes for ray-tracing or other rendering applications.

COMMENTS: A simple ray tracing format, notable because it is one of the few formats designed to support parallel processing.

Overview

PRT (Parallel Ray Trace) is the format associated with the PRT ray-tracing application created by Kory Hamzeh. It is apparently based loosely on Eric Haines' Neutral File Format (NFF). Its main distinguishing characteristic is that the PRT application was designed to support parallel rendering, that is, rendering by a number of machines at once, over a network.

File Organization

Other than the fact that PRT files consist of a number of ASCII lines, there is little mandatory structure. Lines consist of keywords and parameters.

Ex.1007 / Page 758 of 1158
APPLE INC.

The following keywords may be found in a PRT file:

from
at
up
angle
resolution
light
background
surface
cone
sphere
hsphere
polygon
ring
quadric
instance
end_instance
instance_of

Each file must start with the following:

```
from %g %g %g
at %g %g %g
up %g %g %g
angle %g
resolution %d %d
```

The parameters are listed below:

`from`	Eye location in XYZ world coordinates
`at`	Center of the image, in XYZ world coordinates
`up`	Vector indicating which direction is up
`angle`	Angle of image in degrees
`resolution`	Resolution in pixels in both the x and y directions

File Details

The following information is extracted from the documentation supplied by Kory Hamzeh, the creator of PRT, and explains the keywords listed above:

Ex.1007 / Page 759 of 1158
APPLE INC.

Light Sources

A light source is defined as follows:

light X Y Z

Format:

```
light %g %g %g
```

This keyword defines the position of the light sources. All light sources must be defined before any objects are defined.

Background Color

A background color is defined as follows:

background R G B y

Format:

```
background %g %g %g y
```

The background color is in RGB. The last field is used for color cueing (not yet implemented) and must always be 'y'.

Surface Properties

A surface property is defined as follows:

surface Rr Rg Rb Ks Fr Fg Fb T Ar Ag Ab Dr Dg Db Sr Sg Sb P Ior

Format:

```
surface %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
```

Parameters are:

Rr Rg Rb — The reflective color triplet. This value should always be 1 1 1 (unless you want this surface to reflect a different percentage per color component).

Ks — The specular component. This value is the percentage of light that is reflected from this object. A value of 0 means no reflection, and a value of 1 means a perfect reflector (mirror).

Fr Fg Fb — The refractive color triplet. This value should always be 1 1 1 (unless you want this surface to refract a different percentage per color component).

Ex.1007 / Page 760 of 1158
APPLE INC.

T	Transparency value. The amount of light that can go through this object; a value of 0 means a totally opaque object. A value of 1 means a totally transparent object.
Ar Ag Ab	The ambient color for this object; this means the color of an object if it were fully shadowed. All objects are assigned this color before any shading algorithm is started.
Dr Dg Db	The diffuse color component
Sr Sg Sb	This value is the color of the specular highlights. Usually, it should be 1 1 1.
P	The Phong cosine power for highlights. The higher the number (for example 100), the smaller the highlight.
Ior	Index of refraction

Cylinder or Cone

A cylinder or cone is defined as follows:

```
cone
base.x base.y base.z base_radius
apex.x apex.y apex.z apex_radius
```

Format:

```
cone
%g %g %g %g
%g %g %g %g
```

Sphere

A sphere is defined as follows:

```
sphere center.x center.y center.z radius
```

Format:

```
sphere %g %g %g %g
```

Hollow Sphere

A hollow sphere is defined as follows:

```
sphere center.x center.y center.z radius thickness
```


Ex.1007 / Page 761 of 1158
APPLE INC.

Format:

```
sphere %g %g %g %g %g
```

Polygon

A polygon is defined as follows:

polygon total_vertices
vert1.x vert1.y vert1.z
[etc. for total_vertices vertices]

A polygon is defined by a set of vertices. With these databases, a polygon is defined to have all points coplanar. A polygon has only one side, with the order of the vertices being counterclockwise as you face the polygon (right-handed coordinate system). The first two edges must form a non-zero convex angle, so that the normal and side visibility can be determined.

Format:

```
polygon %d
[ %g %g %g ] ← for total_vertices vertices
```

Ring

A ring is defined as follows:

ring center.x center.y center.z p1.x p1.y p1.z p2.x p2.y p2.z or ir

A ring is a flat coplanar round-shaped object. For a ring object, you must specify the following: center, two points on the surface of the ring, the inner radius, and the outer radius. If the inner radius is non-zero, then the ring has a hole in the middle with the given radius.

Format:

```
ring %g %g %g %g %g %g %g %g %g %g %g
```

Quadratic

A quadratic is defined as follows:

quadric center.x center.y center.Z
min.x min.y min.z max.x max.y max.z
a b c d e
f g h i j

Ex.1007 / Page 762 of 1158
APPLE INC.

You can ray trace any quadratic object by specifying the center, minimum, maximum, and coefficients. This is a very powerful object type. It can do ellipsoids, hyperbolas, and any other quadratic surface.

In the model shown above, the fields "a" through "j" are the coefficients.

Format:

```
quadric %g %g %g
%g %g %g %g %g %g
%g %g %g %g %g
%g %g %g %g %g
```

Object Instances

You may define a group of objects (and surface properties) to an instance and assign a name to that instance. When the instance is used, all the objects in that instance are placed relative to the given origin. Note that instances by themselves do not create any objects; the objects are created when the instance is referenced. Instances cannot be nested.

An instance is defined as follows:

```
instance nameofthisinstance
  [ objects and surface properties ]
end_instance
```

where *nameofthisinstance* is a user-assigned name such as, for example, `tile_pattern`.

An instance is referenced as follows:

```
instance_of nameofinstance loc.x loc.y loc.z
```

where *nameofinstance* is the name assigned to a previously defined object instance, and *loc.x*, *loc.y*, *loc.z* represent the location of this object group.

For Further Information

For further information about the PRT format, see the specification included on the CD-ROM that accompanies this book. You can also contact:

Kory Hamzeh / Avatar
6217 Melba Avenue
Woodland Hills, CA 91367
Email: *kory@avatar.com*

Ex.1007 / Page 763 of 1158
APPLE INC.

QRT

NAME:	QRT
ALSO KNOWN AS:	Quick Ray Trace
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Steve Koren
PLATFORM:	All
SUPPORTING APPLICATIONS:	QRT ray-tracing application, others
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes
IMAGES ON CD:	No
SEE ALSO:	NFF, POV, PRT, Radiance

USAGE: Description of 3D scenes for ray tracing or other rendering applications.

COMMENTS: A solid scene description format similar to other ray-tracing formats described in this book. It is not used much today.

Overview

QRT (Quick Ray Trace) is associated with the QRT ray-tracing application created by Steve Koren. As such, it implements the QRT scene-description language. Each QRT file consists of a number of ASCII lines, which define objects in the QRT system, and operations which can be performed by QRT.

File Organization

QRT files consist of a number of ASCII lines consisting of keywords. Like most ray-trace formats, it was designed to be human-readable and to be composed and altered with standard text-editing tools. Keywords may appear in any order in the file. Parameters associated with each keyword may appear in any order, provided that there is no ambiguity.

Ex.1007 / Page 764 of 1158
APPLE INC.

Elements in the file may be floating-point vectors, color value triples, integers, floating-point numbers, and text strings.

QRT is a prototype of many current scene-format and 3D files. Please consult the documentation on the CD-ROM for more information about usage conventions.

File Details

The following is a list of QRT keywords. Each keyword may be followed by one or more parameters, and may or may not be required. Following the list of keywords is an explanation of the parameters.

SKY	FILE_NAME	SPHERE
QUADRATIC	POLYGON	END_INSTANCES
GROUND	OBSERVER	PARALLELOGRAM
PATTERN	BEGIN_BBOX	INSTANCE_OF
FOC_LENGTH	FIRST_SCAN	TRIANGLE
RECTANGLE	END_BBOX	DEFAULT
LAST_SCAN	LAMP	RING
CIRCLE	BEGIN_INSTANCES	

SKY

Type: optional

Parameters: zenith = (r, g, b)
horiz = (r, g, b)
dither = x

Example:
```
SKY ( horiz = (0, 0, .5), zenith = (.5, 0, 0),
dither = 3 )
```

QUADRATIC

Type: optional

Parameters: loc = (x,y,z) (required)
a = floating-point (required)
b = floating-point (required)
c = floating-point (required)
d = floating-point (required)
xmax = floating-point (required)
xmin = floating-point (required)

Ex.1007 / Page 765 of 1158
APPLE INC.

ymax = floating-point (required)
ymin = floating-point (required)
zmax = floating-point (required)
zmin = floating-point (required)
name = string
pattern = string
remove = string
amb = (r,g,b)
diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = integer
index = positive float
dither = integer
sreflect = positive float
reflect = positive float

Example:
```
QUADRATIC (loc = (10,10,10), a = 1, b = 0, c = 1,
d = 100))
```

Notes: Synonyms for loc are location, pos, and position. parameters a, b, c, and d are coefficients in a quadratic equation defining a surface. See the documentation on the CD-ROM for more information.

PATTERN

Type: optional

Parameters:
name = string (required)
x_size = positive float (required)
y_size = positive float (required)
CIRCLE = circle_def
RECTANGLE = rect_def
POLYGON = poly_def

Example:
```
PATTERN ( x_size = 12, y_size = 12, name = TEST)
```

Notes: CIRCLE, RECTANGLE, and POLYGON definitions are the same as with normal keyword syntax.

Ex.1007 / Page 766 of 1158
APPLE INC.

FOC_LENGTH

Type: optional

Parameters: focal_length (integer)

Example:
```
FOC_LENGTH = 60
```

RECTANGLE

Type: optional

Parameters: start_x = positive float (required)
start_y = positive float (required)
end_x = positive float (required)
end_y = positive float (required)
amb = (r,g,b)
diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = integer
index = positive float
dither = integer
sreflect = positive float
reflect = positive float

Example:
```
RECTANGLE ( start_x = 10, start_y = 12, end_x = 15,
end_y = 15)
```

Notes: RECTANGLE is used within a pattern definition only.

LAST_SCAN

Type: optional

Parameters: eliminated in QRT v1.5 and greater

Example: NA

Notes: Replaced by use of X_RES and V_RES.

CIRCLE

Type: optional

Parameters: radius = positive float
amb = (r,g,b)

Ex.1007 / Page 767 of 1158
APPLE INC.

diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = integer
index = positive float
dither = integer
sreflect = positive float
reflect = positive float

Example: `CIRCLE ( radius = 5 )`

Notes: CIRCLE is used within a pattern definition only.

FILE_NAME

Type: required

Parameters: file_name (ASCII)

Example: `FILE_NAME = IMAGE.RAW`

POLYGON

Type: optional

Parameters: point1 = (x,y) (required)
amb = (r,g,b)
diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = integer
index = positive float
dither = integer
sreflect = positive float
reflect = positive float

Example:
```
POLYGON ( point = (0,0), point = (8,0),
point = (0,8), point = (0,0))
```

Notes: POLYGON is used within a pattern definition only. There must be at least four point parameters specified in the parameter list, with the first and last points being the same.

Ex.1007 / Page 768 of 1158
APPLE INC.

OBSERVER

Type: required

Parameters: loc = (x,y,z) (required)
lookat = (x,y,z) (required)
up = (x,y,z)

Example:
```
OBSERVER (loc = (0,0,0), lookat = (90, 0, 0))
```

Notes: Observer cannot look up, lookat must be different from loc, and up must be nonzero. Synonyms for loc are pos, location, and position.

BEGIN_BBOX

Type: optional

Parameters: name = string

Example:
```
name = TEST_BOX
BEGIN_BBOX
   SPHERE      sphere definition here
   TRIANGLE    triangle definition here
END_BBOX
```

Notes: BEGIN_BBOX and END_BBOX always are used together to define a block.

FIRST_SCAN

Type: optional

Parameters: eliminated in QRT v1.5 and greater

Example: NA

Notes: Replaced by use of X_RES and V_RES.

END_BBOX

Type: optional

Parameters: name = TEST_BOX

Example: See BEGIN_BBOX

Notes: END_BBOX is always paired with BEGIN_BBOX.

Ex.1007 / Page 769 of 1158
APPLE INC.

LAMP

Type: optional

Parameters: loc = (x,y,z) (required)
dist = positive float (required)
amb = (r,g,b)
radius = positive float

Example:
```
LAMP ( loc = (10,20,30), dist = 123.5) )
```

BEGIN_INSTANCES

Type: optional

Parameters: NA

Example:
```
BEGIN_INSTANCES
   NAME = object_1
   BEGIN_BBOX
      list of primitives here
   END_BBOX

   NAME = object_2
   BEGIN_BBOX
      list of primitives here
   END_BBOX
END_INSTANCES
```

Notes: Always paired with END_INSTANCES to define a block.

SPHERE

Type: optional

Parameters: loc = (x,y,z) (required)
radius = x (floating-point) (required)
name = string
pattern = string
remove = string
amb = (r,g,b)
diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = a (integer)

Ex.1007 / Page 770 of 1158
APPLE INC.

index = p (floating-point)
dither = a (integer)
sreflect = p (floating-point)
reflect = p (floating-point)

Example:
```
SPHERE ( loc = (10,10,10), radius = 8.8,
diff = (1.0, .1, .1))
```

END_INSTANCES

Type: optional

Parameters: NA

Example: See BEGIN_INSTANCES

Notes: Always paired with a BEGIN_INSTANCES statement to define a block.

PARALLELOGRAM

Type: optional

Parameters: loc = (x,y,z) (required)
v1 = (x,y,z) (required)
v2 = (x,y,z) (required)
name = string
pattern = string
remove = string
amb = (r,g,b)
diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = integer
index = floating-point
dither = integer
sreflect = floating-point
reflect = floating-point

Example:
```
PARALLELOGRAM ( loc = (10,10,10), v1 = (10,0,0),
v2 = (0,0,20))
```

Notes: Synonyms for loc are pos, location, and position. For v1 and v2 you can substitute vect1 and vect2, respectively.

Ex.1007 / Page 771 of 1158
APPLE INC.

INSTANCE_OF

Type: optional

Parameters: name = string (required)
loc = (x,y,z) (required)
scale = (x,y,z)

Example:
```
INSTANCE_OF (name = object_1, loc = (10, 10, 20),
scale = .5,.4,.3))
```

Notes: See BEGIN_INSTANCES and END_INSTANCES keywords.

TRIANGLE

Type: optional

Parameters: loc = (x,y,z) (required)
v1 = (x,y,z) (required)
v2 = (x,y,z) (required)
name = string
pattern = string
remove = string
amb = (r,g,b)
diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = integer
index = floating-point
dither = integer
sreflect = floating-point
reflect = floating-point

Example:
```
TRIANGLE ( loc = (40,20,30), v1 = (8,4,2),
v2 = (0,0,10))
```

Notes: Synonyms for loc are pos, location, and position. For v1 and v2 you can substitute vect1 and vect2, respectively.

Ex.1007 / Page 772 of 1158
APPLE INC.

DEFAULT

Type: optional

Parameters: no_shadow
threshold = positive float
x_res = positive float
y_res = positive float
aspect = positive float
amb = (r,g,b)
diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = integer
index = floating-point
dither = integer
sreflect = floating-point
reflect = floating-point

Example:
```
DEFAULT (no_shadow, diff = (1,0,1), aspect = .8)
```

RING

Type: optional

Parameters: loc = (x,y,z) (required)
v1 = (x,y,z) (required)
v2 = (x,y,z) (required)
rad_1 = floating-point (required)
rad_2 = floating-point (required)
name = string
pattern = string
remove = string
amb = (r,g,b)
diff = (r,g,b)
trans = (r,g,b)
density = (r,g,b)
mirror = (r,g,b)
fuzz = integer
index = floating-point

Ex.1007 / Page 773 of 1158
APPLE INC.

dither = integer
sreflect = floating-point
reflect = floating-point

Example:
```
RING (loc = (10,10,10), v1 = (8,5,0), v2 = (0,0,10),
rad_1 = 10, rad_2 = 20)
```

Notes: Synonyms for loc are pos, location, and position. For v1 and v2 you can substitute vect1 and vect2, respectively. Note that rad_2 must be greater than rad_1, and both must be greater than zero.

For Further Information

Although the author, Steve Koren, says that QRT is obsolete and has been for some time, we find that the QRT distribution is still widely available via FTP from various sites. It is also downloaded with some regularity from the major PC BBSs.

For further information about the QRT file format, see the QRT specification included on the CD-ROM that accompanies this book, and the *QRT Language Reference* found in the QRT distribution. You can also contact:

Steve Koren
Email: *koren@hpfcogu.fc.hp.com*

Ex.1007 / Page 774 of 1158
APPLE INC.

NAME:	QuickTime
ALSO KNOWN AS:	QTM, QuickTime Movie Resource Format
TYPE:	Audio/video data storage
COLORS:	Up to 24 bits
COMPRESSION:	RLE, JPEG, others
MAXIMUM IMAGE SIZE:	64K×64K pixels
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	Apple Computer
PLATFORM:	Apple Macintosh, Microsoft Windows
SUPPORTING APPLICATIONS:	QuickTime, QuickTime for Windows, others
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	JPEG, MPEG, RIFF

USAGE: Storage and interchange of time-based information under the Macintosh and Microsoft Windows environments.

COMMENTS: Currently the most widely used audio-video format, although it competes with Microsoft's RIFF/AVI on Intel machines under Microsoft Windows.

Overview

QuickTime (sometimes called QTM) is the native method of storing audio and motion video information on the Apple Macintosh platform. It is used to record and play back multimedia information and store the data on magnetic or optical media. In this sense, it is similar to multimedia data formats. QuickTime, however, is not only a data-storage format. It is also a collection of tools (the Movie Toolbox) that allows QuickTime movies to be modified (edit, cut, copy, paste, and so on), just as a word processor is capable of modifying an ordinary text file.

A QuickTime movie may be stored as a disk file or may be encoded on a DAT or a CD-ROM. Playback of audio and video data is quick, and the audio and video output at least matches the quality of a VCR-taped program.

Ex.1007 / Page 775 of 1158
APPLE INC.

QuickTime (cont'd)

The QuickTime format allows the storage of multiple tracks of audio and video data. Multiple audio tracks may be used to store the narration for a movie in several different languages. Multiple video tracks may be used to change the video output based on the user responses to an interactive multimedia application. QuickTime movies may also contain a preview, which is a five-second sequence of audio and video data from the movie, and a poster, which is a single frame displayed from the movie data. Both previews and posters are used to quickly identify a movie and its contents.

QuickTime movies are normally structured for the Macintosh environment. However, it is possible to store QuickTime movies in an interchange format, which allows time-based information to be exchanged between the Macintosh and other platforms. This ability allows many multimedia applications that run under non-Macintosh environments, such as Microsoft Windows, the capability of recording and playing back QuickTime movies.

The Movie Toolbox defines six different compression methods that may be used in a QuickTime movie. All of the compression methods used, except for JPEG (Joint Photographic Experts Group, described in Chapter 9, *Data Compression*), are proprietary to Apple Computer and are mentioned only briefly below.

- The Photo Compressor uses the JPEG compression method to compress single-frame images. Continuous-tone images with a pixel depth of eight to 24 bits compressed are the optimal source images for the photo compressor.
- The Video Compressor is a lossy, motion-video compression method, which uses both spatial and temporal compression techniques and has a very fast decompression time. The video compressor is for use with 24-bit, continuous-tone video images.
- The Compact Video Compressor is a lossy, motion-video compression method which is for use with 16- and 24-bit continuous-tone video images. The Compact Video Compressor offers higher image quality, greater compression ratios, and a faster playback speed than is possible when using the Video Compressor, but it requires much more time to perform the initial compression of the video information.

Ex.1007 / Page 776 of 1158
APPLE INC.

- The Animation Compressor uses a motion-video compression method to compress computer-generated and animation sequences. This compressor uses a run-length algorithm which operates on images of any pixel depth and may be selected to perform lossy or lossless compression. The lossy option offers greater data compression ratios at the expense of image quality. This compressor produces high compression ratios at the expense of a slower decompression speed.
- The Graphics Compressor employs a compression algorithm that is used to encode 8-bit still images and image sequences. This compressor produces lower compression ratios, but is able to decompress the image data very quickly. This method is used to encode sequences that will be stored on slower devices, such as CD-ROMs.
- The Raw Compressor is simply a conversion program that increases (pads) or reduces (decimates) the number of bits in a pixel. A 32-bit image is reduced to a 24-bit image by stripping off the alpha channel bits. A 16-bit image is decimated to an 8-bit image by throwing away the eight least significant bits of each pixel. A 4-bit image is padded out to an 8-bit image by adding four bits to each pixel. The Raw Compressor is used most for preprocessing image data to an appropriate pixel depth before it is encoded by another compressor.

Audio data in QuickTime movie files is digitally encoded into 8-bit samples. A sample is an amplitude value represented by the signed integer range of -128 to 127, with 0 representing silence (two's-complement sound encoding), or an unsigned integer range of 0 to 255, with 128 representing silence (offset-binary sound encoding). Samples stored using the Audio Interchange File Format (AIFF) use the two's-complement encoding method, while samples stored directly in a movie's sound media resource are offset-binary encoded.

The following sections describe only the basic format of the QuickTime movie file. For a complete explanation of the QuickTime file architecture, refer to the *Inside Macintosh* series, specifically the sections which describe QuickTime and the Movie Toolbox, or the QuickTime Developer Kit reference manuals and CD-ROM.

File Organization

A QuickTime movie is called a movie resource. In the Macintosh environment it is not necessary to know the internal arrangement of a movie resource. All of the functions available in the Movie Toolbox handle the reading, writing, and

Ex.1007 / Page 777 of 1158
APPLE INC.

interpretation of the movie data for the programmer. In non-Macintosh environments that do not have an emulation of the Macintosh Movie Toolbox, functions must be written to read the movie resources directory, and the internal arrangement of the resources must therefore be known. This is necessary so non-Macintosh platforms may create and play back QuickTime movies.

In the Macintosh environment, QuickTime movies are normally stored in both the resource fork and the data fork of a file. The resource fork contains information about the QuickTime movie data. The data fork contains either the actual movie data or a reference to where the data is located.

A second type of QuickTime movie file, called the single-fork movie file, stores all of the movie data in the data fork, and the resource fork is left empty. This interchange format is used when the movie file will be transported to a non-Macintosh system. When most Macintosh files are moved to a non-Macintosh system, such as MS-DOS or UNIX, the useful information is mostly found in the data fork, and the resource fork information is discarded. When non-Macintosh files are transported to the Macintosh, a resource fork is either created for the file or the resource fork is simply left empty.

In the Macintosh environment movie files have the file type 'moov'. In non-Macintosh environments, movie files usually have the extension .QTM.

File Details

The basic data structure in a move file is called an *atom.* Each atom is a specific collection of data similar to a "chunk" found in the IFF and Microsoft RIFF file formats. The basic format of the atom is shown here:

```
typedef struct _Atom
{
  DWORD Size;          /* Size of the atom in bytes */
  DWORD Type;          /* Atom type identifier */
  ATOM  Atom;          /* One or more atom structures */
  DATA  Data;          /* One or more pieces of data contained with
                          this atom */

} ATOM;
```

Size indicates the size of the atom in bytes, including the Size and Type fields.

Type specifies the type and format of data that is stored in the atom.

Atom identifiers are always 4-character ASCII values.

Ex.1007 / Page 778 of 1158
APPLE INC.

An atom is the actual movie structure. Two varieties of atoms are defined for use in QuickTime movies: the container atom and the leaf atom. Container atoms may contain other atoms, including other container atoms. A leaf atom contains only data and no other atoms.

Two atom types are found in every data fork of a movie file. The first is the movie data atom which has the type identifier mdat. This atom contains the actual movie data. The second is the movie resource atom, which has the type identifier moov. This atom always follows the movie data atom and contains the description of the movie file. Other atoms may follow the movie resource atom, but only the mdat and moov atoms are required to occur in every movie file.

The movie resource atom is actually a directory of all of the information found in the movie and is the closest thing to a header that you will find in a QuickTime movie file. The resource atom contains the following:

- Movie header atom (mvhd)
- Clipping atom (clip)
- One or more track atoms (trak)
- User-defined data (udta)

The clipping and track atoms can, in turn, contain a number of different types of atoms.

Each data stream in a movie file is stored in a track atom. There is one track atom per data stream stored in the movie file. A movie which contains a single audio and video data stream therefore contains two track atoms. Each track atom contains a track header and a media atom, which describes the actual stream data.

Using a C syntax-like notation, you can see the nested structure of atoms within a QuickTime movie file:

```
struct _MovieDirectory
{
  struct _MovieHeaderAtom;
  struct _ClippingAtom
  {
    struct _ClippingRegionAtom;
  }
  struct _TrackDirectory
  {
    struct _TrackHeaderAtom;
    struct _ClippingAtom
```


Ex.1007 / Page 779 of 1158
APPLE INC.

```
    {
      struct _ClippingRegionAtom;
    }
    struct _EditsAtom
    {
      struct _EditListAtom;
    }
    struct _MediaDirectory
    {
      struct _MediaHeaderAtom;
      struct _MediaHandlerAtom;
      struct _MediaInfoAtom;
      {
        struct _VideoMediaInfoAtom
        {
        }
        struct _SoundMediaInfoAtom
         {
           struct _SoundMediaInfoHeaderAtom
           {
             struct _SoundMediaInfoHeaderAtom;
           }
           struct _HandlerAtom;
           struct _DataReferenceAtom;
           struct _SampleTableAtom;
         }
        }
      }
    struct _UserDataAtom;
  }
  struct _UserDataAtom
  {
    struct _MoviesUserData
    {
    }
  }
}
```

This schematic of a QuickTime movie file shows the atoms nested down only to five levels. There are several more levels and dozens of additional atoms not shown here. In this article, we discuss only the main atom types—the movie, track, and media atoms and their respective header atoms. For information on all other atoms, see the references listed in the section called "Movie Resource Atom."

As stated, the movie resource atom is really a directory containing all the information about the movie except for the movie data itself. The movie resource atom has the following structure:

Ex.1007 / Page 780 of 1158
APPLE INC.

```
typedef struct _MovieDirectory
{
  LONG              AtomSize;     /* Size of this atom in bytes */
  LONG              AtomType;     /* Type of atom ('moov') */
  MOVIEHEADERATOM   MovieHeader;  /* Movie header atom for this atom */
  CLIPPINGATOM      MovieClip;    /* Clipping atom for this atom */
  TRACKDIRECTORY    Track[];      /* One or more track atoms */
  USERDATAATOM      UserData;     /* User Definable Extensions */

} MOVIEDIRECTORY;
```

AtomSize is the size in bytes of the atom.

AtomType is the type of atom.

MovieHeader is an atom containing global information about the MovieDirectory atom, the data it contains, and how the data is to be played back.

MovieClip is an atom containing data pertaining to the visual appearance of the movie.

Track is an atom containing an array for each track contained in the movie. There is one track per data stream found in the movie file.

UserData is an atom containing information such as the movie's date of creation, the copyright notice, and the names of the movie's director, producer, writers, and so on.

MovieHeader Atom

The structure of the MovieHeader atom is the following:

```
typedef struct _MovieHeaderAtom
{
LONG  AtomSize;         /* Size of this atom in bytes */
LONG  AtomType;         /* Type of atom ('mvhd') */
LONG  Flags;            /* Atom version and flags */
LONG  CreationTime;     /* Time/date atom was created */
LONG  LastModifyTime;   /* Time/date atom was last modified */
LONG  TimeScale;        /* Time scale used for this movie */
LONG  Duration;         /* Duration of this movie */
DWORD DataRate;         /* Rate at which to play this movie */
SHORT Volume;           /* Movie loudness */
SHORT Reserved1;        /* Not used */
LONG  Reserved2;        /* Not used */
LONG  Reserved3;        /* Not used */
DWORD Matrix[3][3];     /* Transform matrix used by this movie */
LONG  PreviewTime;      /* Time in track the preview begins */
LONG  PreviewDuration;  /* Duration of the movie preview */
LONG  PosterTime;       /* Time in track the poster begins */
LONG  SelectionTime;    /* Time in track the current selection begins */
```


Ex.1007 / Page 781 of 1158
APPLE INC.

```
LONG  SelectionDuration; /* Duration of the current selection */
LONG  CurrentTime;      /* Movie time the current selection begins */
LONG  NextTrackID;      /* Next value to use for an track ID */

} MOVIEHEADERATOM;
```

AtomSize is the size of the atom in bytes.

AtomType is the type of atom.

The first byte of the Flags field indicates the version number of the movie header atom. The remaining three bytes of the Flags field are not used and are reserved for future use.

CreationTime holds the time and date stamp when the header atom was created. LastModifyTime indicates the time and date this atom was last modified. On the Macintosh, these fields are a number representing the number of seconds that have occurred since midnight January 1, 1904 and the actual time/date represented by these fields.

TimeScale contains the number of units per second in the time coordinate system used by this movie. A TimeScale value of 100 indicates that a single unit of time is 1/100th of a second in length.

Duration is the length of the movie in TimeScale units.

DataRate is the rate of data throughput necessary to properly play back the movie.

Volume indicates the volume level at which to play the movie.

Reserved1, Reserved2, and Reserved3 are not used and are set to 0.

Matrix is a 2D array of integers used to transform one visual coordinate system to another.

PreviewTime indicates where in the track the movie preview begins.

PreviewDuration indicates the length of the preview.

PosterTime indicates where the movie poster occurs in the track.

SelectionTime and SelectionDuration indicate the location and length of the currently selected segment of the movie.

CurrentTime indicates the time at which the current selection appears within the movie.

NextTrackID is the track ID of the next occurring track in the movie.

Ex.1007 / Page 782 of 1158
APPLE INC.

TrackDirectory Atom

Each type of data stream in the movie file is represented by a TrackDirectory atom. These 'trak' atoms are stored as an array in the moov atom and have the following structure:

```
typedef struct _TrackDirectory
{
  LONG              AtomSize;     /* Size of this atom in bytes */
  LONG              AtomType;     /* Type of atom ('trak') */
  TRACKHEADERATOM   TrackHeader;  /* Standard track information */
  CLIPPINGATOM      TrackClip;    /* Clipping atom for this track*/
  EDITSATOM         Edits;        /* Edit atom for this track*/
  MEDIADIRECTORY    Media;        /* Media atom for this track */
  USERDATAATOM      UserData;     /* Additional data about this track*/

} TRACKDIRECTORY;
```

AtomSize is the size of the atom in bytes.

AtomType is the type of atom.

TrackHeader contains information specific to this track atom only.

TrackClip is an atom containing data which specifies the spatial clipping region for the track.

Edits is an atom specifing how to map the media data stored in the track.

Media is an atom containing information describing the actual media data represented by this track.

UserData contains user-definable extension data.

TrackHeader Atom

The structure of the TrackHeader atom is the following:

```
struct _TrackHeaderAtom
{
  LONG  AtomSize;       /* Size of this atom in bytes */
  LONG  AtomType;       /* Type of atom ('tkhd') */
  LONG  Flags;          /* Atom version and flags */
  LONG  CreationTime;   /* Time/date atom was created */
  LONG  LastModifyTime; /* Time/date atom was last modified */
  LONG  TrackID;        /* Track ID number */
  LONG  Reserved1;      /* Not used */
  LONG  Duration;       /* Length of track */
  LONG  Reserved2;      /* Not used */
  LONG  Reserved3;      /* Not used */
  SHORT Layer;          /* Priority for this track in movie */
```


Ex.1007 / Page 783 of 1158
APPLE INC.

```
    SHORT AlternateGroup; /* Track group ID value */
    SHORT Volume;         /* Loudness of the track */
    SHORT Reserved4;      /* Not used */
    DWORD Matrix[3][3];   /* Transform matrix used by this track*/
    LONG  TrackWidth;     /* Track width */
    LONG  TrackHeight;    /* Track height */

} TRACKHEADERATOM;
```

AtomSize is the size of the atom in bytes.

AtomType is the type of atom.

The first byte of the Flags field indicates the version number of the track header atom. The remaining three bytes of the Flags field are not used and are reserved for future use.

CreationTime and LastModifyTime fields indicate when this atom was first created and last modified respectively.

TrackID contains a unique value used to identify the track within the movie.

Reserved1 is not used and is set to 0.

Duration indicates the playing time of the track data.

Reserved2 and Reserved3 are not used and are set to 0.

Layer contains the layer level of this track.

AlternateGroup is an identification value associating this track with a specific group of data found within the movie.

Volume is the loudness setting for the track media.

Reserved4 is not used and is set to 0.

Matrix is an array containing a set of data that defines how to map points from one coordinate space into a different coordinate space.

TrackWidth and TrackHeight are the width and height of the rectangle that encloses a visual media track.

Media Atom

The description of the actual media data for this track is contained within the Media atom (mdia). A media atom can contain other atoms, such as a media header (mdhd), a handler reference (hdlr), media information (minf), and user-defined data (udta). Only the media header atom is required.

Ex.1007 / Page 784 of 1158
APPLE INC.

The media atom has the following structure:

```
typedef struct _MediaDirectoryAtom
{
  LONG              AtomSize;        /* Size of this atom in bytes */
  LONG              AtomType;        /* Type of atom ('mdia') */
  MEDIAHEADERATOM   MediaHeader;     /* Media attributes */
  HANDLERATOM       MediaHandler;    /* Media handler atom */
  MEDIAINFO         MediaInfo;       /* Media information atom */

} MEDIADIRECTORYATOM;
```

AtomSize is the size of the atom in bytes.

AtomType is the type of atom.

MediaHeader is an atom specifying the attributes of the media data stream contained within this media atom.

MediaHandler is an atom specifying the type of software service (media handler) that is to interpret the media data.

MediaInfo is an atom that stores information that the media handler uses to interpret the actual media data. The format of this atom varies depending upon the type of media stored.

Media Header Atom

The structure of the MediaHeader atom is as follows:

```
struct _MediaHeaderAtom
{
  LONG  AtomSize;         /* Size of this atom in bytes */
  LONG  AtomType;         /* Type of atom ('mdhd')
  LONG  Flags;            /* Atom version and flags */
  LONG  CreationTime;     /* Time/date atom was created */
  LONG  LastModifyTime;   /* Time/date atom was last modified */
  LONG  TimeScale;        /* Time scale used for this media */
  LONG  Duration;         /* Length of this media */
  SHORT Language;         /* Language code for this media */
  SHORT Quality;          /* Quality rating for this media */

} MEDIAHEADERATOM;
```

AtomSize is the size of the atom in bytes.

AtomType is the type of atom.

Ex.1007 / Page 785 of 1158
APPLE INC.

The first byte of the Flags field indicates the version number of the media header atom. The remaining three bytes of the Flags field are not used and are reserved for future use.

CreationTime and LastModifyTime indicate when this atom was first created and last modified respectively.

TimeScale and Duration specify the type of time scale used and the duration of the media stream in TimeScale units.

Language indicates the language code of this atom.

Quality holds a quantitative value indicating the relative quality of the data stored in the media atom.

For Further Information

For further information about QuickTime, see the documentation included on the CD-ROM that accompanies this book.

The developer guide and kit for QuickTime are available from Apple. Information on Apple Computer programming products, development tools, and technical references also may be obtained directly from:

Apple Computer, Inc.
Attn: APDA
20525 Mariani Avenue
Mail Stop 33-G
Cupertino, CA 95014-6299
Voice: 800-282-2732 (United States)
Voice: 800-637-0029 (Canada)
Voice: 800-562-3910 (all other countries)
FAX: 408-562-3971
WWW: *http://www.quicktime.apple.com/*

Information about Apple developer support programs may be obtained from:

Apple Computer, Inc.
Attn: Macintosh Developer Technical Support
20525 Mariani Avenue
Mail Stop 75-3T
Cupertino, CA 95014-6299
Voice: 408-974-4897

Ex.1007 / Page 786 of 1158
APPLE INC.

Information about QuickTime can also be found in the following:

Apple Computer, *Inside Macintosh,* Imaging and QuickTime volumes, Addison-Wesley, Reading, MA.

Ex.1007 / Page 787 of 1158
APPLE INC.

Radiance

NAME:	Radiance
ALSO KNOWN AS:	None
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Greg Ward
PLATFORM:	All
SUPPORTING APPLICATIONS:	Radiance, others
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes
IMAGES ON CD:	No
SEE ALSO:	DKB, NFF, POV, PRT, QRT, Rayshade, RTrace

USAGE: 3D scene description for use by ray-tracing applications and other renderers.

COMMENTS: A well-documented, well-thought-out format.

Overview

Radiance is a rendering application created by Greg Ward. It implements some of the techniques later lumped under the term *radiosity*. It is well-regarded and has been widely distributed through the Internet graphics community, although perhaps not as widely distributed as some of the other ray-tracing applications. Radiance and the accompanying documentation have attracted a good deal of interest of late due to increased interest in ray-tracing, radiosity methods, and photorealistic rendering in general.

There are actually two formats associated with the Radiance application: the input (or scene description) format and the output format. The input format is well-documented, carefully conceived, flexible, and extensive. The output is a simple bitmap format.

Ex.1007 / Page 788 of 1158
APPLE INC.

File Organization

The input format consists of a series of ASCII lines implementing the scene description language used by the Radiance application. The output format consists of an ASCII information header terminated by an empty line and followed by the bitmap data.

File Details

The input file consists of a list of surfaces and materials. Surface types include spheres, polygons, cones, and cylinders. Materials can be plastic, metal, glass, and others. Light sources known to the system are distant disks as well as local spheres, disks, and polygons.

This section is adapted from the Radiance documentation.

A scene description file represents a 3D physical environment in Cartesian (rectilinear) world coordinates. It is stored as ASCII text, with the following basic format:

```
# comment
modifier PM identifier n S1 S2 S3 ... Sn
0
m R1 R2 R3 ... Rm
modifier (alias identifier reference)
! command
```

A comment line begins with a pound sign (#).

The scene description primitives all have the same general format and can be either surfaces or modifiers. Here are some definitions:

- A primitive has a modifier, a type, and an identifier.
- A modifier is either the identifier of a previously defined primitive or "void" (no modifier).
- An identifier can be any string (i.e., a sequence of non-blank characters).

The arguments associated with a primitive can be strings or real numbers. The first integer following the identifier is the number of string arguments, and it is followed by the arguments themselves (separated by white space). The next integer is the number of integer arguments, followed by the integer arguments themselves. (There are currently no primitives that use them, however.) The next integer is the real argument count and is followed by the real arguments.

Ex.1007 / Page 789 of 1158
APPLE INC.

An alias gets its type and arguments from a previously defined primitive. This is useful when the same material is used with a different modifier or as a convenient naming mechanism. Surfaces cannot be aliased.

A line beginning with an exclamation point (!) is interpreted as a command. It is executed by the shell, and its output is read as input to the program. The command must not try to read from its standard input, or confusion will result. A command may be continued over multiple lines using a backslash (\) to escape the newline.

Blank space is generally ignored, except as a separator. The exception is the newline character after a command or a comment.

Commands, comments, and primitives may appear in any combination, as long as they are not intermingled.

The following example defines a sphere by specifying its center and radius:

```
mod sphere id
0
0
4 xcent ycent zcent radius
```

For other examples, see the CD-ROM that accompanies this book.

For Further Information

For further information about the Radiance format, particularly the Radiance parameters and their primitives, see the Radiance specification and other documents included on the CD-ROM that accompanies this book. On the CD-ROM you will also find source code for reading and writing Radiance format files. You can also contact the author:

Lawrence Berkeley Laboratory
Attn: Gregory J. Ward
Lighting Systems Research Group
Energy & Environment Division
University of California
Building 90-3111
1 Cyclotron Road
Berkeley, CA 94720
Voice: 510-486-4757
FAX: 510-486-4089
Email: *GJWard@lbl.gov*

Ex.1007 / Page 790 of 1158
APPLE INC.

The source code for the Radiance application is maintained at:

ftp://hobbes.lbl.gov/

At that site there are several directories containing files associated with the application:

ftp://hobbes.lbl.gov/pub/models/
ftp://hobbes.lbl.gov/pub/objects/
ftp://hobbes.lbl.gov/pub/pics/

Information is also available via the World Wide Web at:

http://radsite.lbl.gov/radiance/HOME.html

Ex.1007 / Page 791 of 1158
APPLE INC.

Rayshade

NAME:	Rayshade
ALSO KNOWN AS:	None
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Craig Kolb
PLATFORM:	All
SUPPORTING APPLICATIONS:	Rayshade, others
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	DKB, MTV, NFF, POV, PRT, QRT

USAGE: Description of scenes meant to be rendered by programs such as Rayshade.

COMMENTS: A well-constructed format that has influenced writers of more recent ray-tracing programs.

Overview

Rayshade is a ray-tracing application created by Craig Kolb. It is well-respected and has been widely distributed, particularly on the Internet and throughout the PC MS-DOS world, but it has largely been superseded by more recent ray-tracing programs.

The format implements a scene description language, which could be (and has been) used as a model for later rendering applications. It would be a good model to study if you are in the process of writing yet another ray-tracing or 3D scene-rendering application.

Ex.1007 / Page 792 of 1158
APPLE INC.

File Organization

Like many ray-tracing formats, Rayshade files consist of a series of ASCII lines that implement a proprietary command language, this one associated with the Rayshade application.

File Details

The following summary information about the Rayshade format is extracted from the *Rayshade 4.0 Quick Reference* document by Craig Kolb, which is included on the CD-ROM that accompanies this book.

Reals and integers may be written in exponential notation, with or without a decimal point. Reals are truncated to integers when need be.

Numbers may also be written as expressions surrounded by a matched pair of parentheses. Sub-expressions may be parenthesized to control the order of evaluation.

Variables may be defined and used in parenthesized expressions.

Predefined variables include the following:

time (current time)
frame (current frame number, 0 - frames - 1)
pi
dtor (pi/180)
rotd (180/pi)

Available operators include the following:

+ (addition)
- (subtraction and negation)
* (multiplication)
/ (division)
% (remainder)
^ (exponentiation)

Functions include the following:

sin
cos
tan
asin
acos

Ex.1007 / Page 793 of 1158
APPLE INC.

atan
sqrt
hypot

Strings are written as non-quoted strings that may include uppercase and lowercase letters, non-leading digits, and the following special characters:

/ (slash)
- (dash)
_ (underscore)
. (period)

The following command-line options are supported. These override options are set in the input file:

–A frame	First frame to render
–a	Toggle alpha channel
–C cutoff	Adaptive tree cutoff
–c	Continued rendering
–D depth	Maximum ray tree depth
–E eye_sep	Eye separation
–e	Exponential RLE output
–F freq	Report frequency
–f	Flip triangle normals
–G gamma	Gamma exponent
–g	Use Gaussian filter
–h	Help
–j	Toggle jittered sampling
–l	Render left eye view
–m	Produce sample map
–N frames	Total frames to render
–n	No shadows
–O outfile	Output filename
–o	Toggle opaque shadows
–P cpp-args	Arguments for cpp
–p	Preview-quality
–q	Run quietly
–R xres yres	Resolution
–r	Right eye view
–S samples	Use Samples^2 samples
–s	Toggle shadow caching

Ex.1007 / Page 794 of 1158
APPLE INC.

–T r g b	Contrast threshold
–u	Toggle use of cpp
–V filename	Verbose file output
–v	Verbose output
–W lx hx ly hy	Render subwindow
–X l r b t	Crop window

Here are the author's specifications for the construction of the input file:

```
File: /* Input file consists of... */
  <Item> [<Item> ... ]

Item:
  <Viewing>
  <Light>
  <Atmosphere>
  <RenderOption>
  <ObjItem>
  <Definition>

ObjItem: /* Items used in object definition blocks */
  <SurfDef>
  <ApplySurf>
  <Instance>
  <ObjDef>

Viewing:
  eyep Xpos Ypos Zpos          /* Eye position (0 -10 0) */
  lookp Xpos Ypos Zpos         /* Look position (0 0 0) */
  up Xup Yup Zup               /* "up" vector (0 0 1) */
  fov Hfov [Vfov]              /* Field of view in degrees (horiontal=45) */
  aperture Width               /* Aperture width (0) */
  focaldist Distance           /* Focal distance (|eyep - lookp|) */
  shutter Speed                /* Shutter speed (0 --> no blur) */
  framelength Length           /* Length of a single frame (1) */
  screen Xsize Ysize           /* Screen size */
  window Xmin Xmax Ymin Ymax /* Window (0 xsize-1 0 ysize-1) */
  crop left right bot top      /* Crop window (0 1 0 1) */
  eyesep Separation            /* Eye separation (0) */

SurfDef: /* Give a name to a set of surface attributes */
    surface Name <SurfSpec> [<SurfSpec> ...]

Surface: /* Surface specification */
    <SurfSpec>                 /* Use gven attributes */
    Surfname [<SurfSpec> ...]
         /* Use named surface w/ optional mods */
    cursurf  [<SurfSpec> ...]
         /* Use cur. surface w/mods - see ApplySurf */
```


Ex.1007 / Page 795 of 1158
APPLE INC.

Rayshade (cont'd)

```
SurfSpec: /* Surface attribute specification */
  ambient R G B              /* Ambient contribution */
  diffuse R G B              /* Diffuse color */
  specular R G B             /* Specular color */
  specpow Exponent           /* Phong exponent */
  body R G B                 /* Body color */
  extinct Coef               /* Extinction coefficient */
  transp Ktr                 /* Transparency */
  reflect Kr                 /* Reflectivity */
  index N                    /* Index of refraction */
  translu Ktl R G B Stpow    /* Translucency, transmit diffuse, spec exp */
  noshadow                   /* No shadows cast on this surface */

Effect: /* Atmospheric Effects */
  mist   R G B Rtrans Gtrans Btrans Zero Scale
  fog    R G B Rtrans Gtrans Btrans

Atmosphere: /* Global atmosphere */
  atmosphere [Index] <Effect> [<Effect>...]
             /* Global index, effects */

ApplySurf:
  applysurf <Surface>
             /* Apply surf to all following objs w/o surface */

Instance:                    /* Instance of an object */
  <Object> [<Transforms>] [<Textures>]

Object:
  Primitive                   /* Primitive object */
  Aggregate                   /* Named aggregate */

ObjDef:                      /* Define a named object */
  name Objname <Instance>

Primitive:                   /* Primitive object */
  plane    [<Surface>] Xpos Ypos Zpos Xnorm Ynorm Znorm
  disc     [<Surface>] Radius Xpos Ypos Zpos Xnorm Ynorm Znorm
  sphere   [<Surface>] Radius Xpos Ypos Zpos
  triangle [<Surface>] Xv1 Yv1 Zv1
                       Xv2 Yv2 Zv2  Xv3 Yv3 Zv3
            /* flat-shaded triangle */
  triangle [<Surface>] Xv1 Yv1 Zv1 Xn1 Yn1 Zn1
                       Xv2 Yv2 Zv2 Xn2 Yn2 Zn2
                       Xv3 Yv3 Zv3 Xn3 Yn3 Zn3
            /* Phong-shaded triangle */
  polygon  [<Surface>] Xv1 Yv1 Zv1
                       Xv2 Yv2 Zv2  Xv3 Yv3 Zv3 [Xv3 Yv4 Zv4 ...]
  box      [<Surface>] Xlow Ylow Zlow
                       Xhi  Yhi  Zhi
  cylinder [<Surface>] Radius Xbase Ybase Zbase Xapex Yapex Zapex
```


Ex.1007 / Page 796 of 1158
APPLE INC.

```
  cone      [<Surface>] Rbase Xbase Ybase Zbase  Rapex Xapex Yapex
Zapex
  torus     [<Surface>] Rswept Rtube Xpos Ypos Zpos Xnorm Ynorm
Znorm
  blob      [<Surface>] Thresh Stren Rad Xpos Ypos Zpos
                  [Stren Rad X Y Z ...]
  heightfield [<Surface>] Filename

Aggregate:
  Grid
  List
  Csg

Grid:
  grid X Y Z <ObjItem> [<ObjItem> ...]  end

List:
  list <ObjItem> [<ObjItem> ...] end

Csg:
  union      <Object> <Object> [<Object> ...] end
  intersect  <Object> <Object> [<Object> ...] end
  difference <Object> <Object> [<Object> ...] end
  /* CSG only works properly when applied to closed objects, e.g.:
   * sphere, box, torus, blob, closed Aggregate, other Csg object
   */

Transforms: /* Transformations */
  translate   Xtrans Ytrans Ztrans
  scale       Xscale Yscale Zscale
  rotate      Xaxis Yaxis Zaxis Degrees
  transform  A   B   C
             D   E   F
             G   H   I
            [Xt  Yt  Zt]

Textures:
  texture   <TextType> [Transforms] [<Texture>
            [Transforms]...]

Texture:
  checker   <Surface>
  blotch    Scale <Surface>
  bump      Bumpscale
  marbl     [Colormapname]
  fbm       Offset Scale H Lambda Octaves Thresh [Colormapname]
  fbmbump   'Offset Scale H Lambda Octaves
  wood
  gloss     Glossiness
  cloud     Scale H Lambda Octaves Cthresh Lthresh Transcale
  sky       Scale H Lambda Octaves Cthresh Lthresh
```


Ex.1007 / Page 797 of 1158
APPLE INC.

```
  stripe    <Surface> Width Bumpscale [<Mapping>]
  windy     Scale Wscale Cscale Bscale Octaves Tscale Hscale Offset
  image     Imagefile [<ImageTextOption>
            [<ImageTextOption>...]]

ImageTextOption:
  component  <SufComp>
  range      Lo Hi
  smooth
  textsurf   <Surface>
  tile       U V
  <Mapping>

SurfComp:
  ambience
  diffuse
  reflect
  transp
  specular
  specpow

Mapping:
  map uv
  map cylindrical   [Xorigin Yorigin Zorigin Xup Yup Zup Xu Yu Zu]
  map planar        [Xorigin Yorigin Zorigin Xv  Yv  Zv  Xu Yu Zu]
  map spherical     [Xorigin Yorigin Zorigin Xup Yup Zup Xu Yu Zu]

Light:
  light R G B <LightType>     [noshadow]
  light Intensity <LightType> [noshadow]

LightType:
  ambient
  point         Xpos Ypos Zpos
  directional   Xdir Ydir Zdir
  extended      Radius Xpos Ypos Zpos
  spot          Xpos Ypos Zpos Xat Yat Zat Coef Thetain Thetaout
  area          Xorigin Yorigin Zorigin Xu Yu Zu
                Usamples Xv Yv Zv Vsamples

RenderOption:
  samples      Nsamp [jitter | nojitter]
  /* Use Nsamp^2 pixel samples (3^2 jittered) */
  background R G B    /* Background color (0 0 0) */
  outfile    Filename /* Output file name (written to stdout) */
  frames     Nframes  /* Number of frames to render (1) */
  starttime  Time     /* Time corresponding to start of frame 0 */
  contrast   R G B    /* Maximum contrast w/o supersampling */
  maxdepth   Depth    /* Maximum ray tree depth (5)  */
  cutoff     Factor   /*  inium spawned ray contribution (.001) */
```


Ex.1007 / Page 798 of 1158
APPLE INC.

```
  report [verbose] [quiet] [Freq] [Statfile]
                    /* Reporting mode (false false 10 stderr) */
  shadowtransp      /* Toggle object opacity affects shadows */

Definition:        /* Variable definition */
  define Name Expr  /* Assign value for Name */
```

For Further Information

For further information about the Rayshade format, see the *Rayshade 4.0 Quick Reference,* included on the CD-ROM that accompanies this book.

You can also contact the author:

Stanford University
Attn: Craig Kolb
372 Gates Building
Department of Computer Science
Stanford, CA 94305-4070
Email: cek@graphics.stanford.edu

The Rayshade application package is available from various archive sites on the Internet and from many PC MS-DOS BBSs. For further information, contact the author at the above address.

The author also maintains the following Rayshade sites:

ftp://graphics.stanford.edu/pub/rayshade/
http://www-graphics.stanford.edu/~cek/rayshade/rayshade.html

Ex.1007 / Page 799 of 1158
APPLE INC.

RIX

NAME:	RIX
ALSO KNOWN AS:	RIX Image File, ColoRIX VGA Paint
TYPE:	Bitmap
COLORS:	Up to 32 bits per pixel
COMPRESSION:	Undocumented
MAXIMUM IMAGE SIZE:	64K×64K
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	RIX SoftWorks
PLATFORM:	MS-DOS
SUPPORTING APPLICATIONS:	ColoRIX VGA Paint
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	None

USAGE: Storage of bitmap files with few colors under MS-DOS.

COMMENTS: RIX's programs have been bundled with several video cards for the PC running MS-DOS.

Overview

In most respects, the RIX format appears to be a nice format to support. Unfortunately, although the rest of the format, which we have included on the CD-ROM that accompanies this book, is reasonably well-documented, the compression algorithm used in the files is not. RIX SoftWorks says that the algorithm is not published because it is "extremely complicated." The ColoRIX VGA Paint document goes on to explain that:

> Although some compression schemes are more efficient for some pictures, the RIX compression scheme performs extremely well with a broad range of picture types.

Expert opinion is mixed between skepticism and outright dismissal, so it is a shame that there is no way to verify this claim. Certainly, an advance in compression technology would bring RIX more than a modest portion of riches

Ex.1007 / Page 800 of 1158
APPLE INC.

and fame. In any case, until RIX decides to publish its full format specification, you'll just have to wing it with the information provided here.

File Organization

The RIX format is a simple bitmap format, consisting of a fixed header, a palette, and bitmap data.

File Details

The RIX header is structured as follows:

```
typedef struct _RIX_HEAD
{
  CHAR ID[3];           /* Three-character ID field, "RIX" */
  WORD Width;           /* Image width in pixels */
  WORD Height;          /* Image height in lines */
  CHAR PaletteType;     /* Palette type code */
  CHAR StorageType;     /* Format of bitmap data */

} RIX_HEAD;
```

Width and Height represent the size of the image.

PaletteType identifies the type of display device and can have any of the values listed below. These are calculated using a scheme discussed in the specification document.

Value	Type of Display Device
CB	EGA
AB	Extended EGA
AF	VGA
E7	Targa 16
9F	IBM PGA
10	Targa 16
18	Targa 24
20	Targa 32

StorageType can have any of the values listed below. Refer to the specification document for a discussion of the scheme used to calculate these values.

Ex.1007 / Page 801 of 1158
APPLE INC.

Value	Type of Data
80	Compressed
40	Extension block
20	Encrypted
00	Linear, one byte per pixel
01	Planar (0213), similar to EGA
02	Planar (0123), similar to EGA
03	Text

If the storage type value indicates an extension block value, it is followed by a byte containing an extension format value. Some typical format extension types are illustrated below.

Value	Type of Extension
00	ASCII text
01	Original image origin
02	Original image screen resolution
03	Encryptor's ID
04	Bitmap palette in use; length is either 2 or 32 bytes

After the extension format value is a byte containing the total number of bytes in the extension block. The actual extension block data follows immediately afterward.

RIX suggests that developers with special extension needs request an extension storage type value less than 128.

Following the header is a palette, which is either 48 or 768 bytes long. Palette entries are stored as RGB triples, one for each color. Following the palette is the image data. If the image data is not encrypted or compressed, the data format can be deduced from the storage type value.

For Further Information

For further information about the RIX format, see the specification included on the CD-ROM that accompanies this book. You can contact:

RIX SoftWorks, Inc.
Attn: Richard Brownback
18023 Sky Park Circle, Suite J
Irvine, CA 92714
Voice: 714-476-8266

Ex.1007 / Page 802 of 1158
APPLE INC.

You can also contact:

Paul Harker
Voice: 714-476-8486

The RIX file format is also documented in the *ColoRIX VGA Paint* manual, available by purchasing a copy of the program from RIX SoftWorks.

Ex.1007 / Page 803 of 1158
APPLE INC.

RTrace

NAME:	RTrace
ALSO KNOWN AS:	SFF, SCN
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	António Costa
PLATFORM:	All
SUPPORTING APPLICATIONS:	RTrace
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	DKB, NFF, POV, PRT, QRT, Rayshade

USAGE: 3D scene description used for ray-trace and other rendering applications.

COMMENTS: RTrace is a simple but well-designed format that could find general use for describing 3D scenes.

Overview

The RTrace format (called SFF by its author) was created by António Costa and is associated with his ray-trace application, RTrace. Work on both the application and the format date from 1988. The format itself was designed to support the description of 3D scenes. Although it was originally intended for ray-trace applications, it could just as well be used for other renderers. RTrace would be a good format to study if you are in the process of writing yet another ray trace or rendering application.

In his program documentation, António Costa mentions, as a motivation for producing a new format, limits embodied in Eric Haines' NFF format.

Ex.1007 / Page 804 of 1158
APPLE INC.

File Organization

Like many ray trace formats, RTrace implements the scene description language in its own application. The RTrace format consists of a series of ASCII text lines and is designed to be human-readable and easily edited.

File Details

The following information is based on António Costa's documentation, *The SFF Ray-Tracing Format, Version 8.*

An RTrace (SFF) file is divided into five sections (sometimes six, for compatibility reasons). In each, there are definitions, which may be of several types:

- Viewing
- Ambient/Background
- Lights
- Surfaces
- Objects, Textures, and Transformations

Viewing Section

The Viewing section is the first to appear. It has five lines consisting of:

Comments
Eye point
Look point
Up vector
Horizontal and vertical view angles

Each of these items must be on a separate line. Comments can follow up to the end of the line.

Example:

```
viewr
8 0 0   - Eye <EOL>
0 0 0   - Look <EOL>
0 1 0   - Up <EOL>
20 20   - View angles (horizontal and vertical) <EOL>
```


Ex.1007 / Page 805 of 1158
APPLE INC.

Ambient and Background Section

The Ambient and Background section follows the Viewing section. It contains three lines:

- Comments
- Background color
- Ambient color

Both background color and ambient color are defined in RGB format.

After each item there may be comments up to the end of the line.

Example:

```
colors<EOL>
0.1 0.5 0.7  - Sky blue(red=0.1 green=0.5 blue=0.7) <EOL>
0.2 0.2 0.2  - Dark gray(red=0.2 green=0.2 blue=0.2) <EOL>
```

Lights Section

The Lights section contains a series of lines:

- Comments
- One line for each light definition
- Empty line

There are three types of light definitions:

Point light

To define a point light, specify the point code (1), a position, and RGB brightness.

Direction light

To define a direction light, specify the directional code (2), a position, RGB brightness, direction, angle, and attenuation factor. This kind of light radiates from a point in the specified direction inside the solid angle, and the transition may be sharp (factor ~= 1) or soft (factor >> 1). A truly directional light may be simulated by positioning it far away from the objects and defining its brightness to be negative. Normally, illumination decreases with distance; to make illumination distance-independent, make at least one component of the brightness negative (at least one component).

Ex.1007 / Page 806 of 1158
APPLE INC.

Extended light

To define an extended light, specify the extended code (3), a position, RGB brightness, radius and samples. This kind of light is simulated by a sphere of specified radius, which is sampled to calculate the actual illumination (a low value for frequency of samples produces undesirable effects).

Example:

```
lights<EOL>
1  4 5 1  0.9 0.9 0.9  - White point light <EOL>
2  0 10 0  0 0 1  0 -1 0  15 5  - Blue spot light <EOL>
3  8 1 -3  0 1 0  0.3 8  - Green extended light <EOL>
1  1000 1000 1000  -1 -1 -1  - Directional light <EOL>
<EOL>
```

Surfaces Section

The Surfaces section defines all of the surfaces. It consists of a series of lines:

- Comments
- One line for each surface definition
- Empty line

There are two types of surface definitions:

Code 1 definition.

A code 1 surface definition defines a surface by body RGB color, diffuse RGB factor, specular RGB factor, specular exponent factor, metalness factor, and transmission RGB factor. The RGB's colors and factors must be in the [0,1] range. The diffuse RGB factor defines the quantity of light coming from all directions. The specular RGB factor defines the quantity of light coming from the ideal reflection direction. The exponent factor controls the shininess of the surface (the surface is very shiny with a factor bigger than 10; if the factor is near zero, the surface appears dull). The metalness factor makes the reflected light appear white if it is small (as in plastic) or metallic if it is near 1. The transmission RGB factor defines the transparency. The sum of diffuse, specular, and transparency RGB factors should be equal or approximately 1.

Ex.1007 / Page 807 of 1158
APPLE INC.

Code 2 definition.

A code 2 surface definition has a body RGB color, a smoothness RGB factor, a metalness RGB factor, and a transmission RGB factor. This method is an alternative for defining surfaces, but it is more intuitive. The smoothness RGB factor controls the shininess of the surface.

Example:

```
surfaces<EOL>
1  1 0 0  1 1 1  0 0 0  0 0  0 0 0  - Matte <EOL>
1  0 1 0  0.5 0.5 0.5  0.5 0.5 0.5  10 0.5   0 0 0 <EOL>
1  0 0 1  0.7 0.8 0.9  0.3 0.2 0.1  100 1    0 0 0  - Metallic <EOL>
1  1 1 1  0.1 0.1 0.1  0.1 0.1 0.1  200 0.8  0.8 0.8 0.8 <EOL>
2  1 1 0  0 0 0  0 0 0  0 0 0  - Matte <EOL>
2  1 0 1  1 1 1  1 1 1  0 0 0  - Mirror <EOL>
<EOL>
```

Objects, Textures, and Transformations Section

The Objects, Textures, and Transformations section defines the 3D objects, and, optionally, the textures and transformations. It consists of a series of lines, plus one line for each object definition.

All objects are defined by a code, a surface index (which specifies one of the previously defined surfaces, starting in 1), a refraction index, and then the data itself.

Example 1:

The object with code 1 is a sphere and has a center point and radius.

```
1  3  1.0  4 3 2  0.5  - Sphere centered at (4,3,2) radius=0.5 <EOL>
*
```

Example 2:

The object with code 2 is a parallelipiped aligned with the XYZ axis. It requires a center point and three dimensions, for the X, Y, and Z directions.

```
2  2  1.0  1 0 0  10 1 3  - Box at (1,0,0) with sizes (+-10,+-1,+-3) <EOL>
```

Example 3:

The object with code 3 is a bicubic patch or a group of bicubic patches. It is followed by a translation vector, three scale factors for X, Y, and Z, and a file-

Ex.1007 / Page 808 of 1158
APPLE INC.

name or -. If there is a filename, then the patch's geometry is read from that file; otherwise, it is read from the following lines in the SFF file, ending with an empty line:

```
3  1  1.0  0 0 0  1 1 1  example.pat  - Read from file example.pat <EOL>
3  2  1.0  0 0 0  2 1 1  - Read from the next lines <EOL>
```

A bicubic patch is defined by 12 points.

```
1 2 4 5 6 7 9 10 11 12 14 15  - Patch 1<EOL>
2 3 5 6 7 8 10 11 12 13 15 16  - Patch 2<EOL>
<EOL>
0 -0.5 -1  - Vertex 1 <EOL>
0 0 -2  - Vertex 2 <EOL>
0 -0.5 -3 <EOL>
1 0 0 <EOL>
1 0 -1 <EOL>
1 0.5 -2 <EOL>
1 0 -3 <EOL>
1 0 -4 <EOL>
2 0 0 <EOL>
2 0 -1 <EOL>
2 0.5 -2 <EOL>
2 0 -3 <EOL>
2 0 -4 <EOL>
0 -0.25 -1 <EOL>
0 0 -2 <EOL>
0 0 -3 <EOL>
<EOL>
```

Example 4:

The code 4 object is a cone or cylinder. It has an apex center point and radius, followed by a base center point and radius. The apex radius must be less than (in the case of a cone) or equal to (for a cylinder) the base radius. The cone/cylinder are opened objects (i.e., they do not have any circular surfaces in the apex or base).

```
4  1  1.0  0 1 0  0  0 0 0  1  - Cone <EOL>
4  1  1.0  0 1 0  1  0 0 0  1  - Cylinder <EOL>
```

Example 5:

The code 5 object is a polygon or a group of polygons, similar to patches. It is followed by a translation vector, three scale factors for X, Y, and Z, and by a filename or -. If there is a filename, then the polygon's geometry is read from

Ex.1007 / Page 809 of 1158
APPLE INC.

that file; otherwise, it is read from the following lines in the RTrace file, ending with an empty line.

```
5  1  1.0  0 0 0  1 1 1  example.pol  - Read from file example.pol <EOL>
5  2  1.0  0 0 0  1 1 1  - Read from the next lines <EOL>
```

A polygon is defined by its vertices in counterclockwise order. A file with polygons is composed of two parts. In the first part are the number of polygons and the polygon's definitions, using indices into the vertex list. In the second part (after an empty line) is the vertex list, which is terminated by another empty line.

```
5  1 2 3 4 5  - polygon 1 <EOL>
3  1 6 2  - polygon 2 <EOL>
<EOL>
0 0 -2  - vertex 1 <EOL>
1 0 0  - vertex 2 <EOL>
2 0 -1  - vertex 3 <EOL>
2 0 -3  - vertex 4 <EOL>
1 0 -4  - vertex 5 <EOL>
0.5 2 -1  - vertex 6 <EOL>
<EOL>
```

Example 6:

The code 6 object is a triangle or a group of triangles, similar to a polygon but also specifying each vertex normal vector. (These triangles are also known as Phong triangles.) It is followed by a translation vector, three scale factors for X, Y, and Z, and a filename or -. If there is a filename, then the triangle's geometry is read from that file; otherwise, it is read from the following lines in the RTrace file, ending with an empty line.

```
6  1  1.0  0 0 0  1 1 1  example.tri  - Read from file example.tri <EOL>
6  2  1.0  0 0 0  1 1 1  - Read from the next lines <EOL>
⋮
```

A triangle is defined by its vertices (data and normal) in counterclockwise order. A file with triangles is composed of the triangle's definitions: first data and normal vertices, followed by the second data and normal vertices, and finally the third data and normal vertices. It is terminated by another empty line.

```
0 0 0 0 1 0  1 0 0 0 1 0  1 0 -1 0 1 0  - Triangle 1 <EOL>
0 0 0 1 1 0  0 1 0 1 0 0  0 1 -1 1 0 1  - Triangle 2 <EOL>
```

Example 7:

The code 7 object is an extruded primitive derived from closed segments composed of lines and splines. This object is very well-suited to trace high-quality

Ex.1007 / Page 810 of 1158
APPLE INC.

text, although it may be used for many other purposes. It is followed by a filename or -. If there is a filename, then the character's geometry is read from that file; otherwise, it is read from the following lines in the RTrace file, ending with an empty line.

```
spacing     0.1<EOL>
orientation 0 0 -1  0 1 0  1 0 0 <EOL>
encoding    ascii.ppe <EOL>
font        roman.ppf <EOL>
scale       0.4 0.4 0.2 <EOL>
at          1 1 2  "RTrace /copyright/ António Costa 1993"<EOL>
font        times.ppf <EOL>
scale       0.5 0.6 0.3 <EOL>
at          0 3 0  "Etc" <EOL>
:
```

The spacing keyword defines the separation between characters. Most of the supplied fonts have their characters enclosed in a one-unit square.

The orientation keyword defines how the text appears in the 3D space; the first vector defines the text direction, the second the up direction, and the third the extrusion direction. As these are independent, it is possible to slant the text or create more complex effects.

The encoding keyword specifies a file that contains translations from character names to character codes, which are used to access the character data.

The font keyword specifies a file that contains the character's data (number of closed segments—lines and splines—and other data for each character).

The scale keyword defines the scaling for the characters, using the directions specified by the orientation keyword.

The at keyword specifies the starting baseline position and which characters to trace. With the supplied font files, it is possible to use PostScript names for the characters in almost all languages; in this case, the character name must be enclosed in / /.

The next codes are not used to define primitive objects, but rather to associate, transform, or texture objects. The code 64 defines a texture to be applied to an object, usually the previous one. It is followed by a type, an object ID, a transformation matrix, and local data. The supported types are:

Ex.1007 / Page 811 of 1158
APPLE INC.

RTrace (cont'd)

Type	Name	Parameters
0	Null	
1	Checker	surface
2	Blotch	scale surface [file(colormap)]
3	Bump	scale
4	Marble	[file(colormap)]
5	FBM	offset scale omega lambda threshold octaves
6	FBM Bump	offset scale lambda octaves
7	Wood	color(red) color(green) color(blue)
8	Round	scale
9	Bozo	turbulence [file(colormap)]
10	Ripples	frequency phase scale
11	Waves	frequency phase scale
12	Spotted	[file(colormap)]
13	Dents	scale
14	Agate	[file(colormap)]
15	Wrinkles	scale
16	Granite	[file(colormap)]
17	Gradient	turbulence direction(x) direction(y) direction(z) [file(colormap)]
18	Imagemap	turbulence mode axis(horizontal) axis(vertical) file(image)
19	Gloss	scale
20	Bump	3D scale size

For Further Information

For further information about the RTrace format, see the specifications included on the CD-ROM that accompanies this book.

The RTrace application is available for downloading at many Internet archive sites and on PC/MS-DOS BBSs. For further information, contact:

ISEP/INESC
Attn: António Costa
Computer Graphics & CAD Group
Largo Mompilher 22
4000 Porto
Portugal
Email: *acosta@porto.inesc.pt*

Ex.1007 / Page 812 of 1158
APPLE INC.

You can obtain the latest RTrace codes, scenes, images, and documents from the following Web site:

http://diana.inesc.pt/acc.html

or FTP site:

ftp://asterix.inesc.pt/pub/RTrace/

You can get the SCN specifications in PostScript at:

ftp://asterix.inesc.pt/pub/Rtrace/docs/

or in HTML at:

http://www.cica.indiana.edu/graphics/object_specs/scn/SCN.format.html

RTrace images are available at the following:

ftp://asterix.inesc.pt/pub/Rtrace/images/
ftp://asterix.inesc.pt/pub/Rtrace/images-med/
ftp://asterix.inesc.pt/pub/Rtrace/images-mol/
ftp://asterix.inesc.pt/pub/Rtrace/images-3D/

A Macintosh version (MacRTrace 1.8.4.2) developed by Greg Ferrar (*ferrar@uxa.cso.uiuc.edu*) is available at:

ftp://asterix.inesc.pt/pub/Rtrace/macintosh/

Also see the ray-tracing homepage at:

http://arachnid.cs.cf.ac.uk/Ray.Tracing/

Ex.1007 / Page 813 of 1158
APPLE INC.

SAF

NAME:	SAF
ALSO KNOWN AS:	Standard Archive Format, AMSC Archive Format, POD
TYPE:	Bitmap and other (see text)
COLORS:	unlimited
COMPRESSION:	GZIP (see article)
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Any
ORIGINATOR:	Advanced Missile Signature Center
PLATFORM:	All
SUPPORTING APPLICATIONS:	Unknown
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	None

USAGE: Archival storage of multiple data types in one standard format.

COMMENTS: A flexible format designed for the storage of bitmap and other types of data.

Overview

The Advanced Missile Signature Center (AMSC) has created a file format known as the Standard Archive Format (SAF) designed for flexible and extensible use in data archiving. The data may be in ASCII or one of a number of binary formats. The file header is ASCII, however, and can be browsed visually to quickly determine content. As an example of file format technology, it incorporates some unique and interesting features, perhaps reflecting its recent vintage and the experience of its creators.

A file suffix of .SAF usually indicates that the file contains image data. The .POD suffix is usually reserved for files containing POD (Parameter Oriented

Ex.1007 / Page 814 of 1158
APPLE INC.

Data) information. This article discusses the forms of the file that are appropriate for storing image data. Please refer to the document on the CD-ROM for the complete specification and further information.

File Organization

The file consists of an ASCII header followed by binary or ASCII data.

File Details

This section describes the details of the SAF file header, tags, and image data.

Header

The header, always at the start of the file, begins with the ASCII string "HdSize", followed by the length of the header in bytes. HdSize functions as a designated magic number intended to enhance automated file type recognition. HdSize may be followed by the string "auto", in which case the header must be terminated by the string "data".

The rest of the header consists of a list of tags, or *field identifiers*, followed by data fields. The number of tags is not fixed, and tags can be in any order. Tags in SAF files are simply names in ASCII text followed by a space and are case-insensitive. Only tags necessary to interpret or render the data are included, which keeps down fixed file overhead. This technique—using variable numbers of tags in the header—was apparently designed to enhance automated file handling and data extraction while maintaining maximum flexibility. The header supports editing revisions after initial file creation and can be extended at will.

The specification allows both standard tags and user-defined tags. Standard tags are 6 characters or less, and user-defined tags may be up to 29 characters in length. ASCII lines may be terminated by either linefeed or carriage return/linefeed pairs.

Tags

The following tables, adapted from the SAF specification included on the CD-ROM that accompanies this book, list standard tags in SAF files.

Ex.1007 / Page 815 of 1158
APPLE INC.

TABLE SAF-1: *Generic SAF Tags*

Tag	Description	Data Type
HdSize	Header Size (in bytes, or it may be AUTO)	Integer
AqMode	Acquisition Mode	Text
AspAng	Aspect Angle (degrees)	Float
BgFile	Background File	Text
BgType	Background Type	Text
BgValu	Background Value (Average or Fixed)	Float
BPFile	Bad Pixel File	Text
BytOrd	Byte Order	Text
CaFile	Calibration File	Text
ChTemp	High Temp of Cal Source (degrees C)	Float
Class	Classification	Text
CIDay	Collection IRIG Day	Integer
CIHour	Collection IRIG Hour	Integer
CIMin	Collection IRIG Minute	Integer
CISec	Collection IRIG Seconds	Float
ClTemp	Low Temp of Cal Source (degrees C)	Float
COMENT	Comment Line (Repeated as Required)	Text
CSFile	Calibration Source File	Text
Data	End of header	
DaType	Data Type	Text
DaUnit	Data Units	Text
DDOff	Data Distribution Office	Text
DiaFOV	Circular FOV (degrees)	Float
DiStat	Distribution Statement	Text
ElAng	Elevation Angle (deg)	Float
EURAW	Processed Level of the Data EU/RAW/Flat-fielded (FF)	Text
ExpID	AMSC Experiment ID#	Text
Filter	Filter Name	Text
Filtno	Filter Number	Integer
FOVAxl	Chamber Test FOV Axial Location (meters)	Float
FOVRdl	Chamber Test FOV Radial Location (meters)	Float
HdVers	Header Version (= 2.0)	Text
HorFOV	Horizontal FOV (degrees)	Float
IHFOV	Instantaneous Horizontal FOV (microradians)	Float
Itime	Integration Time (seconds)	Float
IVFOV	Instantaneous Vertical FOV (microradians)	Float
LinLog	Linear/Log Indicator (LIN/LOG/ASG)	Text
LODAng	Observer Line-of-Sight Angle with respect to True North (CW is +)	Float

Ex.1007 / Page 816 of 1158
APPLE INC.

Tag	Description	Data Type
LogASl	Log Amp Slope	Float
LogOff	Log Offset	Float
Mach	Mach Number	Float
Mdate	Mission Date (MM/DD/YY)	Text
MeasUn	Measurement Uncertainty-Nominal (%)	Float
Miss	Mission	Text
NCoads	Number of Coads	Integer
NEQ	Noise Equivalent Quantity	Float
NodeNo	Node Number	Text
Note01	Numbered Notes, valid from Note01 through Note99	Text
OffCor	Offset Correction	Float
PPCNam	Platform AMSC Component Name	Text
RDFile	Raw Data File	Text
RolAng	Roll Angle (degrees)	Float
SBPLo	System Bandpass, Lower Wavelength (microns)	Float
SBPUp	System Bandpass, Upper Wavelength (microns)	Float
SclFac	Calibration or Scale Factor	Float
SDLevl	SDIO Data Level	Text
SecCol	Number of Seconds Collected	Integer
SLFile	Spectral Lamp File	Text
SltRng	Slant Range (meters)	Float
SnsAlt	Sensor Altitude (meters)	Float
SPCNam	Sensor AMSC Component Name	Text
Stage	Launch Vehicle Stage Number	Float
StdUnt	Standard Data Units Index	Integer
SUncLo	Independent Parameter Uncertainty Lower Limit	Float
SUncUp	Independent Parameter Uncertainty Upper Limit	Float
TALO	TALO (seconds)	Float
TAOA	Target Angle of Attack (degrees)	Float
Target	Target Name	Text
TestNo	Test Number	Text
TPCNam	Target PC Component Name	Text
TPFact	Transmission Path Factor	Float
TrgAlt	Target Altitude (meters)	Float
TrgHdg	Target Heading with respect to True North (CW is +)	Float
TrgTyp	Target Type (Liquid/Solid)	Text
TrgVel	Target Velocity (meters/second)	Float
TrlNum	Trial Number	Text
TZDay	T-Zero Day	Integer
TZHour	T-Zero Hour	Integer

Ex.1007 / Page 817 of 1158
APPLE INC.

SAF (cont'd)

Tag	Description	Data Type
TZMin	T-Zero Minute	Integer
TZSec	T-Zero Seconds	Float
USRCON	User-Specified Parameters Header Filename	Text
VrtFOV	Vertical FOV (degrees)	Float
Warn01	Warning to the user, valid from 01 thru 99	Text
XUncUn	Independent Parameter Uncertainty Units	Float
YUncLo	Dependent Parameter Uncertainty Lower Limit	Float
YUncLo	Dependent Parameter Uncertainty Upper Limit	Float
YUncUn	Dependent Parameter Uncertainty Units	Float

TABLE SAF-2: ***Additional Tags for SAF Image Data***

Tag	Description	Data Type
ACFile	Area Correction Factor File	Text
ADJFAC	Non-Linear Adjustment Factor	Float
ApSize	Aperture Size	Float
BGBLLX	Lower Left X for Background Box	Integer
BGBLLY	Lower Left Y for Background Box	Integer
BGBLRX	Lower Right X for Background Box	Integer
BGBLRY	Lower Right Y for Background Box	Integer
BGBULX	Upper Left X for Background Box	Integer
BGBULY	Upper Left Y for Background Box	Integer
BGBURX	Upper Right X for Background Box	Integer
BGBURY	Upper Right Y for Background Box	Integer
Bnd01	Boundary Point for PAV files, valid from 01 thru 99	Integer
Bx1Int	Box 1 Integral Value	Float
Bx1LLX	Box 1 Lower Left X	Integer
Bx1LLY	Box 1 Lower Left Y	Integer
Bx1LRX	Box 1 Lower Right X	Integer
Bx1LRY	Box 1 Lower Right Y	Integer
Bx1ULX	Box 1 Upper Left X	Integer
Bx1ULY	Box 1 Upper Left Y	Integer
Bx1URX	Box 1 Upper Right X	Integer
Bx1URY	Box 1 Upper Right Y	Integer
Bx2Int	Box 2 Integral Value	Float
Bx2LLY	Box 2 Lower Left X	Integer
Bx2LLY	Box 2 Lower Left Y	Integer
Bx2LRX	Box 2 Lower Right X	Integer
Bx2ULX	Box 2 Upper Left X	Integer

Ex.1007 / Page 818 of 1158
APPLE INC.

Tag	Description	Data Type
Bx2ULY	Box 2 Upper Left Y	Integer
Bx2URX	Box 2 Upper Right X	Integer
Bx2URY	Box 2 Upper Right Y	Integer
CentMX	Center Cursor X Position	Integer
CentMY	Center Cursor Y Position	Integer
CGain	Camera Gain	Float
Colr1	Color 1 Value	Text
Colr2	Color 2 Value	Text
Colr3	Color 3 Value	Text
Colr4	Color 4 Value	Text
Colr5	Color 5 Value	Text
Colr6	Color 6 Value	Text
Colr7	Color 7 Value	Text
Colr8	Color 8 Value	Text
Colr9	Color 9 Value	Text
Colr10	Color 10 Value	Text
Colr11	Color 11 Value	Text
Colr12	Color 12 Value	Text
Colr13	Color 13 Value	Text
Colr14	Color 14 Value	Text
Colr15	Color 15 Value	Text
Colr16	Color 16 Value	Text
ComPrs	Compression Type	Text
DGFld	Digitizer Gamma setting (non-linear adj.)	Float
DiType	Digitizer Type	Text
DPtNum	Data Point Number	Integer
DSGain	Digitizer System Gain	Float
DSOff	Digitizer System Offset	Float
FldFrm	Fields per Frame	Integer
FlorFr	Field or Frame Data in this File	Text
FRate	Frame Rate (frames/second)	Float
FrstCl	First Color	Integer
EMFile	Emissivity File	Text
IDFile	Image Display Control File	Text
ImDisp	Pre-digitized Image Display Format	Text
ImQual	Image Quality	Text
ImSig	Pre-digitized Image Signal Type	Text
ImSize	Image Size (in bytes after any compression)	Integer
Intrlc	Frame Data Are Interlaced in the File	Text
NClrs	Number of Colors Used	Integer

Ex.1007 / Page 819 of 1158
APPLE INC.

SAF (cont'd)

Tag	Description	Data Type
PLeftX	Profile Left X	Integer
PLeftY	Profile Left Y	Integer
PRghtX	Profile Right X	Integer
PRghtY	Profile Right Y	Integer
ProCX	Profile Cursor X	Integer
ProCY	Profile Cursor Y	Integer
SpecFn	Special Function to Apply to the Image	Text
SRFile	Slant Range Correction Factor File	Text
VrtAtt	Vertical Attitude (degrees)	Float
XMag	X Compress or Enlarge Factor	Float
XPixls	Digitized Data Image Width (pixels)	Integer
XPxWid	X (horizontal) Pixel Width (meters)	Float
YMag	Y Compress or Enlarge Factor	Float
YMax	Maximum y value	Float
YMin	Minimum y value	Float
YPixls	Digitized Data Image Height (pixels)	Integer
YPxWid	Y (vertical) Pixel Width (meters)	Float

TABLE SAF-3: *Tags for Ordered Pair Data*

Tag	Description	Data Type
Calc01	Calculated Parameter, valid from 01 thru 99	Text
FreRsp	System Electrical Frequency Response (Hz)	Float
Keywrd	XY File Keyword	Text
NParam	Number of Parameters in File	Integer
NumDPs	Number of Data Points in File	Integer
PcSize	Parameter Classification Size (bytes)	Integer
PltSub	XY Plot Subtitle	Text
PltTtl	XY Plot Title	Text
PnSize	Parameter Name Size (bytes)	Integer
PodOrd	Data Ordering in POD Files	Text
PuSize	Parameter Units Size (bytes)	Integer
SampRa	Sample Rate (Hz)	Float
XCFile	X-Axis Calibration File	Text
XDaUnt	X-Axis Data Units	Text
XFName	X-Axis Filename (a 1-parameter POD file)	Text
XParam	X-Axis Parameter name	Text
XScFac	X-Axis Scale Factor (applied to XFName values)	Float
XYFNum	XY File Number	Integer

Ex.1007 / Page 820 of 1158
APPLE INC.

Tag	Description	Data Type
XYFrst	XY First Value	Float
XYLast	XY Last Value	Float
YParam	Y-Axis Parameter name	Text

The following are the possible data types that can be stored in the DaType tag:

ASCII	ASCII strings
Int8	1-byte integer (0–255 unsigned)
Int16	2-byte integer
Int32	4-byte integer
Int64	8-byte integer
Flt32	Single-precision floating-point
Flt64	Double-precision floating-point
RGB24	24-bit color image (3 bytes per pixel, RGB)

KeyWrd tags can contain the following values:

Data Type	Tag Value	
Image	IMG	2D array of values (the default tag value)
	CMAP	2D array of colormap indices
	PAV	Position and value triplets
Parameter Oriented Data	POD	Multiple dependent parameters with the same independent parameter
xy pairs	XYPT	(x,y) pairs, y vs. points
	XYFN	(x,y) pairs, y vs. file number
	XYTM	(x,y) pairs, vs. time
	XYDI	(x,y) pairs, y vs. distance (in meters)
y values only	YPT	y vs. points
	YFN	y vs. file number
	YTM	y vs. time
	YDI	y vs. distance (in meters)
	YWL	y vs. wavelength
	YWN	y vs. wavenumber

Ex.1007 / Page 821 of 1158
APPLE INC.

Image Files

Files containing image data contain the tag KeyWrd, which is followed by one of the three types: IMG, CMAP, and PAV:

- If the type is IMG, the data consists of intensity values.
- If the type is CMAP, the data consists of a colormap and indexes into the colormap.
- If the type is PAV (Position and Value), the data consists of ordered triplets in the form (x,y,value).

In IMG files, the header is followed by the image data, XPixls columns by Ypixls rows, and stored in row major order (row one, followed by row two, followed by row three, etc.). The file may contain more than one field; in this case, the image is assumed to be interlaced. A footer may be present following the image data and may contain information about background information (to be subtracted from the image during rendering or analysis) in the form of an array of single-precision floating-point numbers. Units are specified by values for StdUnt or DaUnit tags.

Following the header in CMAP files is a 768-byte palette, consisting of 256 RGB values. This is following by the image data stored as palette indices indexed from the beginning of the palette.

PAV files contain data in the form of ordered triplets, consisting of an ordered pair denoting position and another value related to amplitude. This is designed to enable a rendering application to generate a rectangular array of a size denoted by the values in the XPixls and YPixls tags. Origin is in the upper left corner of the image.

XY data files

Ordered pair data consists of a list of data pairs in (x y) format. The second element of the pair is separated from the first by one or more spaces. Parameter names are stored in tags XParam and YParam, and units for x and y elements of the pair are stored in XDaUnt and DaUnit, respectively.

Data in (x y) format can also be stored as a list of singlet y data, with x values calculated from data stored in tags XYFIRST, XYLAST, and NumDPs.

Other types of files supported directly with predefined tags are Active Source Files and Parameter Oriented Data (POD) files. Please consult the documentation on the CD-ROM for more information on these types of data.

Ex.1007 / Page 822 of 1158
APPLE INC.

CMAP file example

The following fragment should give you some idea of what a SAF file looks like. This is a uncompressed CMAP (palette) bitmap image file six pixels wide by six pixels high. The DaType of Int8 denotes byte size data. Note that at the time of this writing SAF supports only GZIP compression of image data (or none).

```
HdSize Auto
Class  Unclassified
DaType Int8
Keywrd CMAP
XPixls 6
YPixls 6
ComPrs NONE
Data
ff ff ff ab 0f 0d...[rest of 768-byte palette data]...00 00 00
00 00 01 fe 0d 0c
bb 02 04 05 09 27
cc 0d 0a 23 21 22
41 de d2 c3 b9 02
aa ab ad 0a 09 32
cd 44 01 09 03 09
```

For More Information

Information about the SAF format was kindly provided by Dave Holt, and you can reach him at:

Dale Holt
Sverdrup Technology
Arnold AFB, TN 37355
Email: *holt@hap.arnold.af.mil*

Consult the specification document on the SAF format included on the CD-ROM that accompanies this book.

Ex.1007 / Page 823 of 1158
APPLE INC.

Sense8 NFF

NAME:	Sense8 NFF
ALSO KNOWN AS:	WorkdToolkit Neutral File Format
TYPE:	Scene description
COLORS:	16M
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	ASCII, big-endian
ORIGINATOR:	Sense8
PLATFORM:	All
SUPPORTING APPLICATIONS:	Sense8 applications, conversion programs
SPECIFICATION ON CD:	No
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	Discussion of VRML files in Chapter 1

USAGE: Used to describe scenes in Sense8 virtual reality hardware/software systems.

COMMENTS: An early attempt at describing 3D scenes for virtual world construction. It is unfortunate that Sense8 has chosen to name their format NFF, given the prior existence and wide distribution of Eric Haines' NFF format in the 3D community.

Overview

Sense8 is a manufacturer of hardware/software systems that are designed to allow users to experience immersion in a 3D multisensual virtual world. The file format stores the scene description information associated with the visual component of this system.

There are two versions of the format, one in ASCII and one in binary. The binary version follows the conventions of the ASCII version with the exception that only 24-bit color is supported. Unfortunately, information on the binary codes associated with the ASCII tokens was not available at the time of this writing. Version 2.1 of the ASCII version is described here.

Ex.1007 / Page 824 of 1158
APPLE INC.

In the ASCII version, lines may be terminated with either linefeed or linefeed/carriage return pairs. Excess white space between tokens in the file is ignored. Comments start with "//" and are treated like comments in C++ —that is, characters to the right of the comment token are ignored, and a comment is terminated by a linefeed or by a linefeed/carriage return pair.

File Organization

NFF files have a short header followed by one or more object descriptions. ASCII files are terminated according to operating system conventions on the host platform. A reader/parser application is expected to check whether objects are complete prior to rendering.

```
header
object #1
object #2
 ⋮
object #n
```

File Details

The header consists of the file ID "nff", followed by the version number and two optional lines defining the viewpoint. The syntax is illustrated in the following example:

```
nff
version 2.10
viewpos 0.000 0.000 0.000
viewdir 1.000 1.000 1.000
```

Following the header is a list of objects. Objects in NFF files are sets of polygons, and each polygon is defined by a list of vertices. An object definition starts out with a name, followed by the number of vertices, and a list of the actual vertices. Vertices are the x, y, and z coordinates of a point expressed as ASCII real numbers, separated by one or more white spaces. Following the vertex information is a list of polygons constructed from the vertex information.

Vertices are indexed according to the order in which they are defined in the file; the first vertex is number 0.

```
Object Name
number of vertices
vertex #0
vertex #1
 ⋮
vertex #n
```


Ex.1007 / Page 825 of 1158
APPLE INC.

```
polygon #1
polygon #2
  ⋮
polygon #m
```

Vertex information lines may contain an optional normal definition, a color definition, and texture-mapped information. The normal definition is a vector similar in form to the actual vertex specification. Color definitions are expressed as hexadecimal numbers between 000000 and ffffff in the form 0xrrggbb. The presence of texture-mapped information is signified by the token "uv" and is followed by a pair of map coordinates expressed as real numbers. A fully specified vertex line has the following form:

```
x y z norm nx ny nz 0xrrggbb uv U V
```

The following is an example of a vertex at 0,0,0 with normal 0,0,1, colored red, and with u and v values of .5:

```
0.000 0.000 0.000 norm 0.000 0.000 1.000 0xff0000 uv 0.5 0.5
```

In many 3D programs, objects can be either colored or texture-mapped but not both. Color information can be useful, however, when a texture is not available to the rendering application or when texture-mapping is turned off for performance reasons. Colors and uv information (related to texture-mapping) may be included on the same vertex definition line, but the color information will be ignored if a texture file is present and textures are being applied to the object.

Polygon definitions start with an integer indicating the number of vertices making up the polygon. Following that is a list of vertex indices. After the vertex indices is a polygon color, specified in a manner similar to vertex colors. These are normally 24-bit colors. They can, however, be stored in 0xrgb form as 12-bit colors.

Following the color definition is an optional token "both", indicating that both sides of the polygon are visible. If the both token is not present, only the front of the polygon is visible. Polygon orientation is conveyed by noting the order in which the vertices are defined. Vertices are arranged in counter-clockwise order from the point of view of an observer facing the front of a polygon.

Optional texture names and attributes may be specified on the polygon definition line. Texture information tokens are constructed by concatenating texture attribute strings and the texture filename. The following texture attributes are recognized:

Ex.1007 / Page 826 of 1158
APPLE INC.

`_v_` "vanilla" texture (no shading)
`_s_` shaded
`_t_` transparent
`_u_` shaded and transparent

Thus, a token signifying a shaded texture created from bitmap texture file *grass* would have the form `_s_grass`. Transparency works on black pixels in the texture bitmap. All black pixels are treated as transparent if transparency is turned on.

Textures applied to polygons may also be modified through the use of the tokens rot (rotate), scale, trans (translate), and mirror. Tokens rot, scale, and trans are followed by real numbers. Rot is expressed in radians, mirror is a toggle, and all are values in u-v texture coordinate space. Texture modifications are applied in the following order: mirror, rot, scale, and trans.

Polygons may be labeled with a token of the form "id=*n*", where *n* is a unique polygon number. A "portal name" may also be assigned to a polygon. This assignment is meant to allow the loading of an extension to the virtual world being rendered when the polygon, acting as a "portal," is crossed (presumably by the traveling point of view in an interactive situation).

The complete polygon syntax has the following form:

```
v1 v2...vn 0xrrggbb both _t_name rot 0.000 scale 0.000 trans 0.000
    mirror id=m -nextworld
```

An example of this would be a six-sided polygon, with shaded grass texture mapped to the front, with the texture rotated, scaled, and translated. An `id` value of 2 has been assigned to it, and portal "nextworld" is to be loaded when the polygon is traversed by the point of view or camera during an interactive rendering session. Note that the red value is ignored due to the presence of a texture specification.

```
5 0 1 2 3 4 0xff0000 _s_grass rot 1.0 scale 0.5 trans 1.0
    1.0 id=2 -nextworld
```

Note that vertex color specifications may override polygon color specifications, so for full compatibility with Sense8 products, you must examine color definitions for each polygon and compare them with those in the vertex definitions. If the vertex definitions are all one color, this color is assigned to the polygon, and the color information associated with the polygon is ignored.

In the same way, vertex texture values may override polygon texture designators if the vertex values making up the polygon all include u-v values.

Ex.1007 / Page 827 of 1158
APPLE INC.

The following is an example of a WTK ASCII NFF file.

```
nff
version 2.1
viewpos 0.0 0.0 0.0              // optional viewpoint...
viewdir 0.0 0.0 1.0              // ...and viewing direction
FirstObject                      // object name
8                                // total vertices
3.0 3.0 -3.0                     // vertex list
3.0 -3.0 -3.0
-3.0 -3.0 -3.0
-3.0 3.0 -3.0
3.0 3.0 3.0
3.0 -3.0 3.0
-3.0 -3.0 3.0
-3.0 3.0 3.0
6                                   // total polygons.
4 0 1 2 3 0xff0000                  // red
4 7 6 5 4 0x00ff00                  // green
4 0 4 5 1 0x0000ff                  // blue
4 1 5 6 2 0xffff00  _s_grass        // shaded texture "grass"
4 2 6 7 3 0xffffff  _t_sky rot 1.0  // transparent, rotated texture "sky"
4 3 7 4 0 0x000000  _v_world -nextworld
                                    // vanilla texture "world", and
                                    // polygon is a "portal" to nextworld
```

For Further Information

For additional information about the Sense8 NFF file format, contact:

Sense8 Corporation
100 Shoreline Highway
Suite 282
Mill Valley, CA 94941
Voice: 415-331-6318
WWW: *http://www.sense8.com/*

You might also want to take a look at the SIG-WTK FTP site, an archive of related 3D objects and image textures, as well as user-contributed software:

ftp://artemis.arc.nasa.gov/~ftp/sig.wtk

Ex.1007 / Page 828 of 1158
APPLE INC.

NAME:	SGI Image File Format
ALSO KNOWN AS:	SGI, RLE, Haeberli
TYPE:	Bitmap
COLORS:	16 million
COMPRESSION:	RLE
MAXIMUM IMAGE SIZE:	64K×64K pixels
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	Silicon Graphics
PLATFORM:	UNIX
SUPPORTING APPLICATIONS:	SGI graphics software
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	None

USAGE: The SGI image file format is a generic bitmap format used for storing black-and-white, gray-scale, and color images.

COMMENTS: SGI is one of the few formats to make use of a scan-line offset table to indicate the beginning of each scan line within compressed image data. SGI files may also contain only color maps or alpha channel data, and they may have a very large number of bit planes.

Overview

The SGI image file format is actually part of the SGI image library found on all Silicon Graphics machines. SGI image files may store black-and-white (.BW extension), color RGB (.RGB extension), or color RGB with alpha channel data (.RGBA extension) images. SGI image files may also have the generic extension .SGI as well.

The SGI image file format was developed by Paul Haeberli at Silicon Graphics.

Ex.1007 / Page 829 of 1158
APPLE INC.

File Organization

The SGI image file format header is 512 bytes in size and has the following structure:

```
typedef struct _SGIHeader
{
  SHORT Magic;          /* Identification number (474) */
  CHAR Storage;         /* Compression flag */
  CHAR Bpc;             /* Bytes per pixel */
  WORD Dimension;       /* Number of image dimensions */
  WORD XSize;           /* Width of image in pixels */
  WORD YSize;           /* Height of image in pixels */
  WORD ZSize;           /* Number of bit planes */
  LONG PixMin;          /* Smallest pixel value */
  LONG PixMax;          /* Largest pixel value */
  CHAR Dummy1[4];       /* Not used */
  CHAR ImageName[80];   /* Name of image */
  LONG ColorMap;        /* Format of pixel data */
  CHAR Dummy2[404];     /* Not used */

} SGIHEAD;
```

Magic is the SGI file identification value; it is always decimal 474.

Storage indicates whether the image data is compressed using an RLE algorithm (value of 01h) or is stored uncompressed (value of 00h).

Bpc is the number of bytes per pixel. This value may be 01h or 02h; most SGI images have a value of 01h, or one byte per pixel.

Dimension indicates how the image data is stored. A value of 01h indicates that a single-channel image is stored as one long scan line. A value of 02h indicates a single-channel bitmap whose dimensions are indicated by the XSize and YSize header field values. A value of 03h indicates a multi-channel bitmap with the number of channels shown by the value of the ZSize header field.

XSize and YSize are the width and height of a bitmap image in pixels.

ZSize is the number of channels in a bitmap image. Black-and-white images typically have a ZSize of 01h, RGB images a ZSize of 03h. RGB images with an alpha channel have a ZSize of 04h.

PixMin specifies the minimum pixel value in the image. This value is typically 00h.

PixMax specifies the maximum pixel value in the image. This value is typically FFh.

Ex.1007 / Page 830 of 1158
APPLE INC.

Dummy1 is a 4-byte NULL character field that is not used.

ImageName is an 80-byte character field used to store the name of the image. The name string may be up to 79 characters in length and must be terminated with a NULL.

ColorMap specifies how the pixel values are to be regarded. Values are shown below:

00h	Normal pixel values. Black-and-white images have one channel, color images have three channels, and color images with an alpha value have four channels.
01h	Dithered image with only one channel of data. Each dithered pixel value is one byte in size, with the red channel value stored in bits 0, 1, and 2; the green value in bits 3, 4, and 5; and the blue value in bits 6 and 7.
02h	Single-channel image. The image contains pixels that are index values into a color map stored in another SGI file.
03h	Stored image data is a color map to be used for other images and should not be displayed.

SGI files with ColorMap values of 01h and 02h are considered obsolete.

Dummy2 is a 404-byte NULL character field used to pad the header out to an even 512 bytes in size.

In SGI files containing uncompressed image data, the image data appears immediately after the header. In SGI files with compressed image data, a scan-line offset table follows the header, and the compressed image data follows the table.

File Details

The majority of SGI files store single-channel, 8-bits-per-pixel, black-and-white images. Such images typically have a Bpc of 01h, a Dimension of 01h, and a ColorMap of 01h. Color RGB images typically have a Bpc of 01h, a Dimension of 02h (or 03h if an alpha channel is present), a ZSize of 03h (or 04h if an alpha channel is present), and a ColorMap of 01h.

The origin (0,0) for all SGI images is the lower-left corner of the display with the first scan line starting at the bottom of the image.

Ex.1007 / Page 831 of 1158
APPLE INC.

SGI files are found in two basic flavors: run-length encoded and verbatim (uncompressed). Verbatim image data is written out by plane as scan lines. For example, a 3-channel image has all of the data for its first plane written first, followed by the data for the second plane, and finally by the data for the third. If Bpc is set to 01h, then each scan line contains an XSize number of BYTEs; if Bpc is set to 02h, then each scan line contains an XSize number of SHORTs.

In RLE image data, a scan-line offset table is used to keep track of the offset where each scan line begins within the compressed image data. The offset table appears immediately after the header and before the compressed image data. The table contains one entry per scan line, determined by multiplying the YSize and ZSize values together. Each entry is a LONG (4-byte) value.

The offset table is actually two tables written consecutively to the SGI file. The first table contains the starting offset values of each scan line; each offset is calculated from the beginning of the file. If the image data is stored as two or more bit planes (ZSize > 1), then all of the offset values for the first plane are stored first, followed by all of the offsets for the second plane, and so on. The second table stores the compressed length of each encoded scan line in BYTEs. And, once again, if the data is stored in more than one bit plane, the offset values are stored by plane.

Note that the scan-line table cannot be ignored during the reading of compressed data, even if you are decoding the image completely from beginning to end. There are several reasons for this:

1. The SGI specification dictates that scan lines need not be stored in consecutive order; only planes are required to be stored consecutively. It is therefore possible that a scan line might be stored in an interleaved fashion (0, 4, 8, 12, ... rather than 0, 1, 2, 3, ...).
2. Multiple entries in the scan-line table might point to the same scan line. An image with many identical scan lines (containing all white pixels, for example) might encode only one such scan line and have all identical entries in the offset table pointing to the same line. It is also possible that a gray scale image stored as three planes (RGB) would have each plane pointing towards the same scan line.

The RLE algorithm used to compress the SGI image data varies in format depending upon the value of the Bpc field in the header. If the Bpc is 01h, this indicates one byte per pixel. A simple 2-byte RLE encoding scheme is used, in

Ex.1007 / Page 832 of 1158
APPLE INC.

which the lowest seven bits of the first byte is the run count. The high bit in this byte is the run-count flag. If this bit is set to 0, then the next byte (the run value) is repeated a number of times equal to the run count. If the run count flag is 1, then the run count indicates the number of BYTEs to copy literally from the input stream to the output stream.

If the Bpc value is 02h, then each pixel is stored in a 2-byte SHORT value. The RLE algorithm is basically the same, with each RLE packet being three BYTEs long, rather than two. Bits 0 through 6 of the low byte are the run count, and bit 7 is the run-count flag. The run value is the SHORT value following the run-count byte. If bit 7 is set to 0, this indicates a repeat run count of SHORT pixel values. If bit 7 is set to 1, this indicates a literal run count of SHORT pixel values. Using either pixel size, each decompressed scan line should be XSize pixels in length.

For Further Information

For further information about the SGI image format, see the specification included on the CD-ROM that accompanies this book. Information on all Silicon Graphics file formats may be obtained directly from SGI:

Silicon Graphics Inc.
Attn: Visual Magic Marketing
2011 North Shoreline Blvd.
Mountain View, CA 94039-7311
Voice: 800-800-4SGI
FTP: *ftp://sgi.com/*
WWW: *http://www.sgi.com/*

If you are using a Silicon Graphics workstation, you may refer to the documentation on the *–limage* library by using the man command *man 4 rgb.*

Ex.1007 / Page 833 of 1158
APPLE INC.

SGI Inventor

NAME:	SGI Inventor
ALSO KNOWN AS:	IRIS, 3D Interchange File Format
TYPE:	3D scene description
COLORS:	Unlimited
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	NA
ORIGINATOR:	Silicon Graphics
PLATFORM:	UNIX
SUPPORTING APPLICATIONS:	Many
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes
IMAGES ON CD:	No
SEE ALSO:	SGO

USAGE: Known primarily through SGI's IRIS Inventor system.

COMMENTS: SGI Inventor was designed for the exchange of 3D modeling information between software applications.

Overview

The SGI Inventor file format was first released in July 1992 by Silicon Graphics, and was specifically designed for the exchange of 3D modeling information between software applications. It has been used by CAD, chemistry, financial data, scientific visualization, art history, earth sciences, creative, presentation, architecture, animation, and other applications.

The SGI Inventor file format was created as part of the IRIS Inventor 3D Toolkit. The toolkit is an object-oriented 3D class library for the C and C++ languages that enables programmers to write interactive 3D graphics programs. IRIS Inventor's file format is used by a large assortment of 3D applications, such as SGI's Showcase, SGI's Explorer Scientific Visualization System, Industrial Light and Magic's animation system, SDRC, Parametric Technology, and many more.

Ex.1007 / Page 834 of 1158
APPLE INC.

The toolkit is based on an object-oriented database (OODB) to represent a 3D hierarchical scene. The scene database contains many types of objects, such as groups, transformations, labels, materials, drawing styles, cameras, lights, 18 different geometry types, and so on. And because SGI Inventor is object-oriented, the file format can be extended to support custom objects. The SGI Inventor file format therefore is an ASCII version of a scene database.

File Organization

The SGI Inventor file format may write either binary or ASCII data, depending on how the file is to be used. The data in both file types are machine-independent. The internal format of the Inventor binary format is proprietary and cannot be discussed in depth in this article.

SGI Inventor ASCII files contain only 7-bit, ASCII information and are parsed just as any other text file would be when read. Each line of information in an Inventor ASCII file is terminated by a linefeed (ASCII 0Ah) character. Lines beginning with the # are comments and are generally ignored.

Inventor ASCII files contain the following header signature:

```
#Inventor V1.0 ascii
```

while Inventor binary files contain the following header signature:

```
#Inventor V1.0 binary
```

Although these signatures may vary in design in future versions of the Inventor file formats, they will never be longer than 80 characters and will always begin with the # comment character.

File Details

Following the header signature is a series of data nodes that contain the actual rendering information. All information in an Inventor file is conceptualized as objects called nodes. Nodes may contain other nodes (called child nodes) and may also be grouped into collections of nodes called graphs. The syntax for a node is as shown below:

```
nodename {
    fieldname1 value1 value2
    fieldname2 value1 value2 value3
    fieldname3
    fieldnameN value
}
```


Ex.1007 / Page 835 of 1158
APPLE INC.

A node contains a node name followed by a series of data field names and data field values. The fields may appear in any order within a node and are not written out if their values are the default values for the field. The fields of a node are always enclosed in braces.

Nodes may also contain other nodes called child nodes:

```
nodename {
    fieldname1 value
    fieldname2 value
    fieldnameN value
    childnode1 {
        fieldname1 value
        fieldname2 value
        fieldnameN value
    }
    childnode2 {
        fieldname1 value
        fieldname2 value
        fieldnameN value
    }
}
```

The values in each field are defined for each node. A field may contain one or more values. Each value is separated by a white space character. A field containing multiple groups of values surrounds the entire grouping with brackets, and commas separate each group; for example:

```
nodename {
    fieldname1 value1 value2 value3
    fieldname2 [ value1a value1b value1c, value2a value2b value2c ]
}
```

The format used to write field values depends upon their data type, as shown below:

100	Integers
100.0	Floating-point
1.0e2	Floating-point
"name"	Strings
mnemonic	Enumerations and bit fields
~	Ignore this field

Ex.1007 / Page 836 of 1158
APPLE INC.

A field name which contains a tilde (~) for a value is ignored by the SGI Inventor file reader. The following is a fragment of an SGI Inventor ASCII file:

```
Separator {
    Normal {
        vector  0 1 0
    }
    Material {
        ambientColor    0.2 0.2 0.2 ~
        diffuseColor    0.720949 0.714641 0.492981
        specularColor   0.2 0.2 0.2 ~
        emissiveColor   0 0 0 ~
        shininess       0 ~
        transparency    0 ~
    }
    Coordinate3 {
        point [  0.0    0.0  0.0,    # 0
                 1.0    0.0  0.0,    # 1
                 1.0    1.0  0.0,    # 2
                 2.0    0.0  0.0,    # 3
                 2.0    1.0  0.0,    # 4
                 2.0    1.0  0.0,    # 5
                 3.0    0.0  0.0,    # 6 ]
    }
    IndexedTriangleMesh {
        coordIndex     [ 0, 1, 2, 3, 4, 5, -1 ]
    }
}
```

Each node stored in an Inventor file represents a 3D shape, property, or grouping. All SGI Inventor nodes are divided into five classes:

- Shape
- Property
- Group
- Light
- Camera

Shapes include geometric objects. Properties are the qualitative aspects of the objects. Group indicates the type of organization applied to one or more nodes. Light and Camera information affect the appearance of the rendered image.

Ex.1007 / Page 837 of 1158
APPLE INC.

The following list contains the Inventor nodes grouped by class:

Shape:

Cone	IndexedNurbsCurve	PointSet
Cube	IndexedNurbsSurface	QuadMesh
Cylinder	IndexedTriangleMesh	Sphere
FaceSet	LineSet	Text2
IndexedFaceSet	NurbsCurve	Text3
IndexedLineSet	NurbsSurface	TriangleStripSet

Property:

BaseColor	MatrixTransform	Texture2
Complexity	Normal	Texture2Transform
Coordinate3	NormalBinding	TextureCoordinate2
Coordinate4	NurbsProfile	TextureCoordinateBinding
DrawStyle	PackedColor	TextureCoordinateCube
Environment	PickStyle	TextureCoordinateCylinder
Font	ProfileCoordinate2	TextureCoordinateEnvironment
Info	ProfileCoordinate3	TextureCoordinateFunction
Label	ResetTransform	TextureCoordinatePlane
LightModel	Rotation	TextureCoordinateSphere
LinearProfile	RotationXYZ	Transform
Material	Scale	Translation
MaterialBinding	ShapeHints	Units

Group:

Array	LayerGroup	Separator
CustomNode	MultipleCopy	Switch
File	PathSwitch	
Group	Selection	

Light:

DirectionalLight PointLight SpotLight

Camera:

OrthographicCamera PerspectiveCamera

Ex.1007 / Page 838 of 1158
APPLE INC.

For Further Information

For further information about the SGI Inventor format, see the following documents included on the CD-ROM that accompanies this book.

Silicon Graphics, Inc., *How to Write an IRIS Inventor File Translator*, Release 1.0.

Silicon Graphics, Inc., *How to Write an Open Inventor File Translator*, Release 2.0.

Silicon Graphics, Inc., *IRIS Inventor Nodes Quick Reference Guide*, Release 1.0.

Silicon Graphics, Inc., *IRIS Inventor Nodes Quick Reference Guide*, Release 2.0.

Information on the SGI Inventor file format may also be found in the following document, available directly from Silicon Graphics:

SGI Inventor Programming Guide, Volume I: Using the Toolkit.

The following book contains a chapter describing the SGI Inventor file format for Release 2.0:

Wernecke, Josie, *The Inventor Mentor*, Addision-Wesley, Reading, MA, 1994.

For additional information, contact:

Silicon Graphics Inc
Attn: Visual Magic Marketing
2011 North Shoreline Blvd.
Mountain View, CA 94039-7311
Voice: 800-800-4SGI
FTP: *ftp://sgi.com/sgi/inventor/*
WWW: *http://www.sgi.com/Technology/Inventor/index.html*

You can also contact:

Gavin Bell
Silicon Graphics Inc.
FAX: 415-390-6056
Email: *gavin@sgi.com*

Ex.1007 / Page 839 of 1158
APPLE INC.

SGI YAODL

NAME:	SGI YAODL
ALSO KNOWN AS:	Yet Another Object Description Language, Power-Flip Format
TYPE:	3D scene description
COLORS:	Millions
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	ASCII, big-endian
ORIGINATOR:	Silicon Graphics
PLATFORM:	SGI
SUPPORTING APPLICATIONS:	Powerflip
SPECIFICATION ON CD:	No
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	SGI Inventor

USAGE: A simple format used for storing 3D vector data.

COMMENTS: YAODL is not widely used at Silicon Graphics and is constantly in danger of being rendered obsolete by more robust 3D imaging formats, such as SGI Inventor.

Overview

YAODL (Yet Another Object Description Language) is a description language used for storing 3D object data to disk files. YAODL is a rather obscure format that was created specifically for the Silicon Graphics Powerflip demo program, and it is not widely supported by SGI. However, it is a simple example of a basic vector-object description language.

YAODL supports a small collection of object types, including NURBS, polygons, and quad-meshes. Object properties such as normals (facet or vertex), colors (object, facet, and vertex), and texture coordinates are also supported.

Ex.1007 / Page 840 of 1158
APPLE INC.

Advanced features of YAODL include hierarchical models, coordinate transformations (rotations, scales, translations, and so forth), and instancing (using the same data more than once).

File Organization

A YAODL file begins with the comment header #YAODL, followed by one or more YAODL objects. Each object can be completely independent of all other objects in the YAODL file, or it can rely on previously defined object data within the same file.

Each object may be written to a YAODL file using either an ASCII or binary data format. Therefore, a YAODL file may contain entirely ASCII data, entirely binary data, or a mixture of the two.

The advantage of ASCII objects is that they are easy to modify using a simple text editor and are portable between different machine platforms. And while binary objects are less portable, they are smaller and faster to load than their ASCII equivalents.

Comments may be inserted into ASCII YAODL files either by enclosing the comment in the Standard C comment tokens /* and */, or by including the UNIX shell-style comment # at the beginning of each line. All comments and white space characters are ignored by YAODL file parsers.

File Details

An ASCII YAODL file contains one or more YAODL objects, delimited by commas. Each object may have one of the following syntaxes:

```
(object_type  argument1, argument2, ... )
(object_type  argument1, ... : property1, property2, ... )
integer integer integer ...
float float float ...
"some character String"
name = { one of the forms above }
name
```

Each YAODL object may contain zero or more arguments and have zero or more properties. For example, a colors object contains a minimum of three arguments:

```
(colors 0.0 1.0 0.0),
```

The three arguments here are RGB float values and define the color green. It is possible for many objects to have multiple groups of arguments. For example,

Ex.1007 / Page 841 of 1158
APPLE INC.

a colors object may define more than one color, with each color represented by a set of three float values:

```
(colors 0.0 1.0 0.0  0.0 1.1 0.0  0.0 1.2 0.2),
```

An object may be used by other objects that take objects as their arguments or properties. For example, we may define a red polygon with four vertices in the following way:

```
(polygons
 (vertices -1. -1. 0. 1. -1. 0. 1. 1. 0. -1. 1. 0.),
 : colors  0.0 1.0 0.0
),
```

Here we have a polygons object that takes one vertex's object as its argument and a color object as its properties. We can shorten this declaration by assigning a name to the colors object and using the new name in the properties list of the polygons object:

```
green = (colors                                    0.0 1.0 0.0),
(polygons
 (vertices -1. -1. 0. 1. -1. 0. 1. 1. 0. -1. 1. 0.),   : green),
```

Note that the colors object, green, is a property of the polygons object, and object properties affect only the object to which they are assigned. The properties of each object must be explicitly specified or their default value(s) are assumed.

You probably have noticed that all the objects we've seen so far are described using their plural inflection, such as polygons rather than polygon. This is because it is possible to describe more than one rendered object within a YAODL object. For example, we could create three polygons, each red, green, and blue, using three polygon objects:

```
/* Three polygons */
red = (colors 1.0 0.0 0.0),
green = (colors                                    0.0 1.0 0.0),
blue = (colors 0.0 0.0 1.0),
(polygons
  (vertices -1. -1. 0. 1. -1.0.  1. 1. 0. -1. 1. 0.),   : red),
(polygons
  (vertices  1. -1. 0. 1. 1.0. -1. 1. 0. -1. -1. 0.),   : green),
(polygons
  (vertices  1. 1. 0. -1. 1.0. -1.-1. 0.  1. -1. 0.),   : blue),
```

Or we could nest all of these descriptions together to achieve exactly the same rendering using a single polygon object:

Ex.1007 / Page 842 of 1158
APPLE INC.

```
(polygons
 (vertices -1. -1. 0.  1. -1. 0.  1. 1. 0.  -1. 1. 0.),
 (vertices  1. -1. 0.  1. 1. 0. -1. 1. 0.  -1. -1. 0.),
 (vertices  1. 1. 0. -1. 1. 0. -1. -1. 0.  1. -1. 0.),
 : colors 1.0 0.0 0.0  0.0 1.0 0.0  0.0 0.0 1.0,
),
```

It is important to realize that a YAODL reader does not perform any data type conversion, so be careful not to use an integer when a float is needed, and so on.

Normally, each name and object defined in an ASCII YAODL file has a global scope across the entire file starting at the point where it is defined. Braces { } may be used to create limited, block-scope variables within a YAODL file. Names defined within this local scope have precedence over identical names declared in the global scope and do not exist outside of their block.

```
blue = (colors 0.0 0.0 1.0), /* Define the color blue */
(polygons
  (vertices  1. 1. 0.  2. 2. 0.), : blue), /* Draw a blue line */

{
blue = (colors 0.0 0.15 1.0),
          /* Redefine the color blue for this block only */
(polygons
  (vertices  2. 2. 0. 2.5 2.5 0.), : blue), /* Draw a blue line */ }
# This line is drawn in the original blue.
(polygons
  (vertices  2.5 2.5 0. 3. 3. 0.), : blue), /* Draw a blue line */
```

In all cases, names must be defined before they are referenced.

Hierarchies may be created within YAODL objects using the group object. A group object may have any number of arguments, each of which is a YAODL object. A group object may also contain properties, but each property must have a definition for each object in the group. For example:

```
(group V =
 (vertices -0.175000 0.350000 0.020000),
N =
 (normals 0.000000 0.000000 1.000000),
I =
 (indices 0 1 2),
half = (indexpolygons V, I, : N), half,
(group half,
 : (rotates 180.000000 0.000000 0.000000 1.000000),
),
),
```


Ex.1007 / Page 843 of 1158
APPLE INC.

As previously mentioned, objects within a YAODL file may also be stored using a binary format. Binary YAODL objects start with an @ character followed by a NULL-terminated ASCII string identifying the name of an object. Valid object names are:

colors	regularMesh
contours	rotates
indexpolygons	scales
indices	texcoords
group	textures
normals	translates
nurbs	trimcurves
polygons	vertices

Following the object name string is an 8-byte integer specifying the number of bytes of binary data that follow. The format of the binary data depends on the type of data used by the object. Integer and floating-point data are stored normally, using the native byte order of the machine. Indices are stored using the following format:

The number_of_lists specified is an integer indicating how many arguments (groups of integers) the object has. The array length is a list of integers specifying the length of each group of integers. Following this array is the integer data for each list.

The following is an example of a small YAODL file that renders a cube:

```
#YAODL
v =
(vertices 0.250000 -0.250000 -0.250000 0.250000 0.250000 -0.250000
-0.250000 0.250000 -0.250000 -0.250000 -0.250000 -0.250000
0.250000 0.250000 0.250000 0.250000 -0.250000 0.250000
-0.250000 0.250000 0.250000 -0.250000 -0.250000 0.250000 ,),
i =
(indices
0 1 2 3 ,
0 1 4 5 ,
4 1 2 6 ,
7 6 2 3 ,
0 5 7 3 ,
7 6 4 5 ,
)
,
(indexpolygons v,
i,
: (colors 1.000000 1.000000 0.400000 0.700000 0.500000 0.200000
```


Ex.1007 / Page 844 of 1158
APPLE INC.

```
0.000000 0.000000 1.000000 0.300000 1.000000 1.000000
0.200000 1.000000 0.700000 1.000000 0.700000 0.700000 ,),
(normals 0.000000 0.000000 -1.000000 1.000000 0.000000 0.000000
0.000000 1.000000 0.000000 -1.000000 0.000000 0.000000
0.000000 -1.000000 0.000000 0.000000 0.0000001.000000
,),
),
```

For Further Information

For further information about YAODL, see the YAODL(6D) manual page found on the Silicon Graphics system. For more information, contact:

Silicon Graphics Inc.
Attn: Visual Magic Marketing
2011 North Shoreline Blvd.
Mountain View, CA 94039-7311
Voice: 800-800-4SGI
FTP: *ftp://sgi.com/*
WWW: *http://www.sgi.com/*

Ex.1007 / Page 845 of 1158
APPLE INC.

SGO

NAME:	SGO
ALSO KNOWN AS:	Showcase, Silicon Graphics Object
TYPE:	3D vector
COLORS:	Unlimited
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	Silicon Graphics
PLATFORM:	UNIX
SUPPORTING APPLICATIONS:	SGI Showcase
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	SGI Inventor

USAGE: Used primarily with Silicon Graphics Showcase.

COMMENTS: A useful format to examine if you are interested in interchange strategies.

Overview

SGO (Silicon Graphics Object) is a binary format used to store 3D image-rendering information. SGO was originally created for internal use at Silicon Graphics, but is now associated with the Silicon Graphics Showcase application. SGO is used primarily as a way to import 3D models into Showcase, although the SGI Inventor format is generally preferred over SGO as an interchange medium. (See the SGI Inventor article for information about this format.)

File Organization

The SGO file format does not have an actual header as most image file formats do. The first two bytes of every SGO file is a magic number value of 5424h. This value identifies the file as being an SGO data file.

Ex.1007 / Page 846 of 1158
APPLE INC.

Following the magic number is a series of data objects. Each object begins with a data token value indicating the type of data the object stores. Valid data token values are:

01h Quadrilateral List
02h Triangle List
03h Triangle Mesh
04h End Of Data

An SGO data file may contain any number of these objects in any order, except that there is only one End Of Data object per file, and it must appear as the last object stored in the data file. An End Of Data object contains only a token value and no data.

File Details

The SGO Quadrilateral List and SGO Triangle List data objects have the same structure, shown below:

```
typedef struct _QuadTriListObjects
{
  LONG ObjectToken;          /* Object token identifier */
  LONG DataSize;             /* Size of the data in this object in WORDs */
  struct _Vertex             /* Object vertex structure(s) */
  {
    FLOAT XNormalVector;     /* X axis of the normal vector at the vertex */
    FLOAT YNormalVector;     /* Y axis of the normal vector at the vertex */
    FLOAT ZNormalVector;     /* Z axis of the normal vector at the vertex */
    FLOAT RedVertexComponent;   /* Red color component at the vertex */
    FLOAT GreenVertexComponent; /* Green color component at the vertex */
    FLOAT BlueVertexComponent;  /* Blue color component at the vertex */
    FLOAT XVertex;           /* X axis of the vertex */
    FLOAT YVertex;           /* Y axis of the vertex */
    FLOAT ZVertex;           /* Z axis of the vertex */
  } Vertices[DataSize / 9];

} SGOQUADLIST, SGOTRIANGLIST;
```

ObjectToken is the object token identification value. This value is 01h for Quadrilateral List objects and 02h for Triangle List objects.

DataSize is the size in WORDs of the data contained within this object. The size of the ObjectToken and DataSize fields are not included in this value.

Each vertex in the object is encoded as an array of one or more 18-byte structures. Each vertex structure contains nine fields of data defining a vertex in the object. The fields in each structure are defined as follows:

Ex.1007 / Page 847 of 1158
APPLE INC.

XNormalVector, YNormalVector, and ZNormalVector contain the position of the normal vector at this vertex.

RedVertexComponent, GreenVertexComponent, and BlueVertexComponent hold the RGB values for the color of this vertex.

XVertex, YVertex, and ZVertex contain the position of the vertex itself.

The structure of the SGO Triangle Mesh data object is similar to the List objects, but adds a few more fields of information:

```
typedef struct _TriMeshObject
{
  LONG ObjectToken;          /* Object token identifier */
  LONG DataSize;             /* Size of the data in this object in WORDs */
  LONG VertexSize;           /* Size of the vertex data in WORDs */
  struct _Vertex             /* Object vertex structure(s) */
  {
    FLOAT XNormalVector;     /* X axis of the normal vector at the vertex */
    FLOAT YNormalVector;     /* Y axis of the normal vector at the vertex */
    FLOAT ZNormalVector;     /* Z axis of the normal vector at the vertex */
    FLOAT RedVertexComponent;   /* Red color component at the vertex */
    FLOAT GreenVertexComponent; /* Green color component at the vertex */
    FLOAT BlueVertexComponent;  /* Blue color component at the vertex */
    FLOAT XVertex;           /* X axis of the vertex */
    FLOAT YVertex;           /* Y axis of the vertex */
    FLOAT ZVertex;           /* Z axis of the vertex */
    } Vertices[DataSize / 9];
  struct _MeshControl        /* Mesh Control structure(s) */
  {
    LONG MeshControlId;      /* Mesh Control identifier */
    LONG NumOfIndices;      /* Number of indices stored in the control */
    LONG Indices[NumOfIndices]; /* Index values */
  } Vertices[];

} SGOTRIANGMESH;
```

ObjectToken is the object token identification value. This value is 03h for Triangle Mesh objects.

DataSize is the size in bytes of the data contained within this object. The size of the ObjectToken and DataSize fields are not included in this value.

VertexSize is the number of WORDs required to store the vertex data in the object.

Each vertex in the object is encoded as an array of one or more 18-byte structures. Each vertex structure contains nine fields of data defining a vertex in the object. The fields in each vertex structure are defined as follows:

Ex.1007 / Page 848 of 1158
APPLE INC.

XNormalVector, YNormalVector, and ZNormalVector contain the position of the normal vector at this vertex.

RedVertexComponent, GreenVertexComponent, and BlueVertexComponent hold the RGB values for the color of this vertex.

XVertex, YVertex, and ZVertex contain the position of the vertex itself.

Following the vertices array is an array of mesh control structures. These structures hold data that specifies how the vertex data is to be linked together. The fields in each mesh control structure are defined as follows:

MeshControlId is the identifier for the type of mesh control. The valid values for this field are:

01h Begin Triangle Mesh
02h Swap Triangle Mesh
03h End Begin Triangle Mesh
04h End Triangle Mesh

NumOfIndices indicates how many indices are stored in this mesh control. Indices is an array of the index values for this mesh control.

For Further Information

For further information about the SGO file format, see the specification included on the CD-ROM that accompanies this book. In addition, see the references in the SGI Inventor article and the following article, available from SGI:

> Silicon Graphics, *IRIS Showcase User's Guide,* Appendix C, "Graphics Library Programming Guide."

Contact:

> Silicon Graphics Inc.
> Attn: Visual Magic Marketing
> 2011 North Shoreline Blvd.
> Mountain View, CA 94039-7311
> Voice: 800-800-4SGI
> FTP: *ftp://sgi.com/*
> WWW: *http://www.sgi.com/*

Ex.1007 / Page 849 of 1158
APPLE INC.

SPIFF

NAME:	SPIFF
ALSO KNOWN AS:	Still Picture Interchange File Format, JPG, SPF
TYPE:	Bitmap
COLORS:	Bitonal to 32-bit
COMPRESSION:	Modified Huffman, MR, MMR, JBIG, JPEG, uncompressed
MAXIMUM IMAGE SIZE:	4G×4G pixels, 64K×64K pixels for non-tiled baseline JPEG
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	ISO/IEC
PLATFORM:	All
SUPPORTING APPLICATIONS:	All that support the JFIF format
SPECIFICATION ON CD:	No
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	JFIF and the discussion of JPEG and JBIG compression in Chapter 9, *Data Compression*

USAGE: SPIFF is the official replacement for the JFIF file format for storing JPEG data. It is also the format to use for storing JBIG data, and it offers an alternative to CCITT Group 3, Group 4, and CALS for storing MR and MMR data.

COMMENTS: SPIFF is a new international standard and is currently supported by very few applications. Most JFIF readers, however, will have no problem interpreting SPIFF-JPEG files.

Overview

SPIFF is a generic bitmap file format defined by ITU (International Telecommunication Union) and ISO/IEC (International Standards Organization/International Electrotechnical Commission) for the storage, compression, and interchange of color or gray-scale, continuous-tone images, and bitonal image data.

Ex.1007 / Page 850 of 1158
APPLE INC.

SPIFF may be best described as the "official" JPEG file format. When the Joint Photographic Experts Group (ISO/IEC JTC1/SC29/WG1) established the JPEG compression standard in 1990, they didn't create a corresponding standard file format for the storage and interchange of JPEG data. Some five years later, SPIFF has been ratified by the JPEG committee to fill this omission.

Why was an official JPEG file format not created by the original JPEG committee? The official reason is that the JPEG Convener at that time realized that numerous other standards groups were defining file formats for various application environments, such as SC18 for the Office Document Architecture (ODA) and SC24 for image processing applications. Each of these groups was planning on storing JPEG-compressed data within file formats of its own design.

The Convener reasoned that a single file format covering the needs of all applications could not be adequately defined and concluded that the other standards bodies should be left to create their own formats to encapsulate JPEG data. The Convener also indicated that any file format work undertaken by the JPEG committee could be perceived as infringing upon the scope of work of other standards bodies.

One unofficial reason for the decision was that the JPEG committee was under pressure to release its standard and, with quite a bit of work left to do, could not see taking on another major task, such as that of defining a file format.

But raw JPEG data stored in a file does require some ancillary information to make it useful (such as the color space of the image), so creating a file format for JPEG data was something that someone needed to do, even if the format wasn't going to be officially sanctioned by the JPEG committee.

The format that emerged was the JPEG File Interchange Format (JFIF) created in 1992 by Eric Hamilton of C-Cube Microsystems and other developers as well. JFIF is the format typically used when software reads and writes what is more commonly referred to as a JPEG file. Although JFIF was a very simple format—containing little more than a header followed by a JPEG data stream—it was very portable across all operating systems, and it quickly became the de facto standard file format for storing JPEG image data.

When Eric Hamilton took over as WG1 Convener (JPEG and JBIG committees), he started to work on a completely defined file format for JPEG data. His rationale was that everybody else was working on large and complicated formats with lots of features, while the great majority of users only need something simple that allowed image interchange. The interchange of compressed

Ex.1007 / Page 851 of 1158
APPLE INC.

pictures definitely falls within the scope of the ISO project JTC 1.29.04 (JPEG), and, therefore, Hamilton reasoned that the committee could start work on SPIFF without going through the red tape of balloting a new work item proposal.

Why use SPIFF rather than JFIF? JFIF is small, simple, and widespread, and practically every JPEG image display program reads it. Why give it up?

One reason is that SPIFF is much more carefully designed, specified, and thought-out than JFIF. SPIFF is an official standard rather than an ad hoc one, and it has been through a more thorough review process.

SPIFF is also more flexible than JFIF. Extended features include support for more color spaces and a provision for specifying image gamma. JFIF took a shortcut by attempting to require that all JFIF images have a gamma of 1.0. That requirement has been widely ignored because many pre-existing images have other gamma values, and, as it turns out, a gamma of around 0.4 to 0.5 is technically superior.

The variation in gamma values means that JFIF images frequently come out too dark or too light, depending on their origins and the viewing system. SPIFF offers the opportunity to improve the situation by marking files with their image gamma. Viewers can then correct image brightness as needed for their display hardware.

SPIFF is part of the JPEG standard and therefore is very well-defined in format, application, and compliance testing. It is fully expected that SPIFF will eventually replace JFIF as the file format of choice for continuous-tone color and grayscale compressed image data.

SPIFF will also be supported by the Independent JPEG Group's (IJG) JPEG library (included on the CD-ROM that accompanies this book). What this means is that you can integrate SPIFF support into your image and graphics applications using a well-known, well-written, widely distributed, and freely available source code library that hundreds of applications already use.

The SPIFF specification does not define a standard file extension or type indicator for SPIFF files. IJG recommends that the extension ".JPG", and "JPEG" type indicator, be used for SPIFF files containing lossy (DCT) JPEG-compressed data, and that ".SPF" be used for all other variants of SPIFF. (Of course, the JBIG community might prefer a ".JBG" extension for SPIFF-JBIG files.)

The file extension .JPG is already commonly used for JFIF-format JPEG files. However, properly written JFIF-compatible software should read SPIFF-JPEG

Ex.1007 / Page 852 of 1158
APPLE INC.

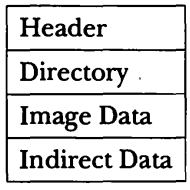
files without difficulty. The SPIFF format has been carefully designed to make this possible by defining the magic numbers and length fields to make the SPIFF header look like a series of JPEG APPn markers, which old JPEG decoders will just ignore.

It is also reasonable to expect that SPIFF-JPEG software will also read JFIF files for backwards compatibility. Because JFIF and SPIFF-JPEG are interoperable, there is no need to confuse the average user by introducing a new file extension for SPIFF files containing JPEG data.

The non-JPEG variants of SPIFF, however, are not interoperable with any existing software and, in fact, will confuse JFIF-only software considerably, so those variants need a different extension. Using the extensions ".JPG" and ".SPF" also offers the advantage of maintaining a clear distinction between lossy and lossless SPIFF image formats, which should help to minimize user confusion and unintentional degradation of data.

File Organization

SPIFF files are composed of four major sections: the header, the information directory, the image data, and an optional section containing indirect data (that is, information too large to fit in the information directory).



The header is typical of most bitmap file headers and contains information necessary to properly decode the image data. The directory may be thought of as a secondary header that contains optional fields of information called *directory entries.* The image data is stored immediately after the directory and is followed by any directory data that was too large to fit in a single directory entry.

File Details

This section describes the contents of the SPIFF header and directory and provides other details of the format.

Ex.1007 / Page 853 of 1158
APPLE INC.

SPIFF Header

The header is 36 bytes in length and has the following format:

```
typedef struct _SpiffHeader
{
    DWORD MagicNumber;          /* Primary identification value
                                   (FFD8FFE8h) */
    WORD  HeaderLength;         /* Header length (not including
                                   MagicNumber) */
    CHAR  Identifier[6];        /* Secondary ID value ("SPIFF\0") */
    WORD  Version;              /* SPIFF version */
    BYTE  ProfileId;            /* Application profile */
    BYTE  NumberComponents;     /* Number of color components */
    DWORD ImageHeight           /* Number of lines in image */
    DWORD ImageWidth;           /* Number of samples per line */
    BYTE  ColorSpace;           /* Color space used by image data */
    BYTE  BitsPerSample;        /* Number of bits per sample */
    BYTE  CompressionType;      /* Type of data compression used */
    BYTE  ResolutionUnits;      /* Type of resolution units */
    DWORD VerticalResolution;   /* Vertical resolution */
    DWORD HorizontalResolution; /* Horizontal resolution */
} SPIFFHEADER;
```

MagicNumber is the identification value for SPIFF files. This 4-byte field always contains the value FFD8FFE8h.

HeaderLength contains the length of the header excluding the MagicNumber field. In v1.0 of SPIFF, this value is always 32.

Identifier contains additional identification values. These values are 53h 50h 49h 46h 46h 00h (the NULL-terminated string "SPIFF").

Version contains the major and minor revision of SPIFF that the file conforms to. The most significant byte contains the value 01h and the least significant byte contains the value 00h. These values correspond to v1.0.

Differing minor version numbers represent backward-compatible changes in the SPIFF format. Differing major version numbers represent backward-incompatible changes in SPIFF. File readers should attempt to read SPIFF files even if the minor revision number is not recognized, but should give up if the major version is not recognized.

ProfileId specifies the features that the file reader must support to read the file. The possible values for this field are 0 (no profile specified), 1 (continuous-tone base profile), 2 (continuous-tone progressive profile), 3 (bi-level facsimile profile), and 4 (continuous-tone facsimile profile).

Ex.1007 / Page 854 of 1158
APPLE INC.

NumberComponents indicates the number of color components (channels) in the primary image. This value is 1 for a typical gray-scale image and 3 for an RGB or CMY image.

ImageHeight and ImageWidth store the size of the image. ImageHeight is the number of scan lines in the primary image. ImageWidth is the number of samples per line.

ColorSpace specifies the color space coordinate system used to define the samples. Allowed values for this field are:

0 Bi-level
1 YCbCr, ITU-R BT 709, video
2 No color space specified
3 YCbCr, ITU-R BT 601-1, RGB
4 YCbCr, ITU-R BT 601-1, video
5 Reserved
6 Reserved
7 Reserved
8 Gray-scale
9 PhotoYCC
10 RGB
11 CMY
12 CMYK
13 YCCK
14 CIELab

BitsPerSample specifies the number of bits per sample.

CompressionType indicates the type of compression algorithm used to encode the image data. The possible values for this field are:

0 Uncompressed, interleaved, 8 bits per sample
1 Modified Huffman
2 Modified READ
3 Modified Modified READ
4 JBIG
5 JPEG

ResolutionUnits indicates the type of units used to express the resolution of the image. Possible values for this field are 0 (aspect ratio defined by VerticalResolution and HorizontalResolution), 1 (dots or samples per inch), or 2 (dots or samples per centimeter).

Ex.1007 / Page 855 of 1158
APPLE INC.

VerticalResolution and HorizontalResolution contain the resolution of the image. If ResolutionUnits is 0, the values of these fields contain, respectively, the numerator and denominator of the aspect ratio of the samples. Otherwise, these fields contain the image resolution as fixed-point numbers.

Directory

Following the header is a directory of references to information stored within the SPIFF file. This directory may be thought of as a second header that contains one or more optional fields of information about the image data. See Figure SPIFF-1 for a diagram of the directory entry structure.

The directory will contain at least one directory entry; the End Of Directory is mandatory; all other entries are optional. Data associated with the directory entry may be stored "directly" within the directory entry or be stored "indirectly" outside of the directory and following the image data. The maximum size of a block of data that may be stored within a directory entry is 65,527 bytes.

The header of each directory entry is 12 bytes in length and has the following format:

```
typedef struct _SpiffDirectoryEntry
{
    WORD  EntryMagic;   /* Directory entry magic number (FFE8h) */
    WORD  EntryLength;  /* Length of entry */
    DWORD EntryTag;     /* Identification value of the entry */
} SPIFFDIRECTORYENTRY;
```

EntryMagic identifies the start of each directory entry. This value is always FFE8h.

EntryLength is the length of the entry, not including the EntryMagic field. The value of this field may be in the range 6 to 65534.

EntryTag is a bit field identifying the format and type of data stored or referenced by the directory entry. Each directory entry will always have a unique EntryTag value and a specific format of entry data.

The eight most significant bits (31:24) of EntryTag are always zero. The value of the next three bits (23:21) define the originating standards body to which the file data conforms. The possible values are:

0 SPIFF specification definition
1–3 ISO/IEC and common text generic standards

Ex.1007 / Page 856 of 1158
APPLE INC.

4 ISO application standards
5 ITU-T recommendations
6 National standards bodies
7 Application-specific

The remaining bits (20:0) are defined by the standards organization defining the particular tag. The SPIFF specification defines the possible values of these remaining 20 bits when the three identifier bits are set to zero. See the section below called "Entry Tags" for more detailed information.

Each directory entry must be a multiple of four bytes in length. An entry with no associated data is eight bytes in length. An entry containing an offset to indirect data is 12 bytes in length. And an entry containing direct data must have its data padded to end on the nearest 4-byte boundary.

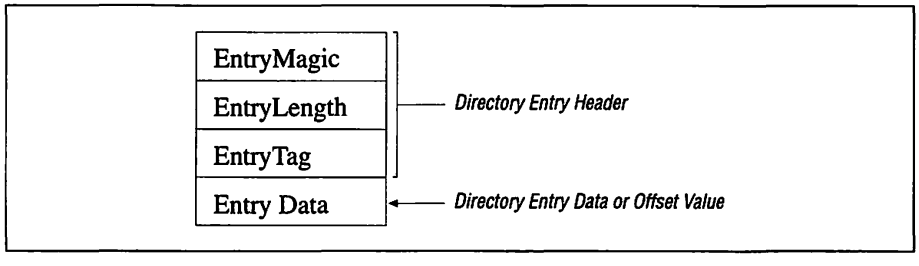


FIGURE SPIFF-1: *Format of a SPIFF directory entry*

Directory entries are not linked together by offset values in the way that TIFF image file directories are. Instead, entries occur in contiguous order after the header and before the image data.

The last entry in a directory is the EOD (End Of Directory) entry and marks the end of the directory. In an EOD entry, EntryMagic is FFE8h, EntryLength is always 8, and EntryTag is always 1.

Note that the EntryLength value of the EOD entry is two bytes larger than it seems it should be. This is because the length of the EntryMagic field is also added into this value, but only for the EOD entry. As we noted, the EntryMagic, EntryLength, and EntryTag fields are defined to appear as a JPEG SOI marker followed by a series of JPEG APP8 markers, so the SPIFF header and directory entries are ignored by JFIF readers. The EOD EntryLength value is two greater than it should be so that old JFIF decoders will also skip over the

Ex.1007 / Page 857 of 1158
APPLE INC.

SOI marker that appears at the beginning of the SPIFF data area. Otherwise, older decoders would see two SOI markers and complain.

Header
Directory Entry 1
Directory Entry 2
Directory Entry N
EOD Entry
Image Data
Indirect Data

Entry Tags

SPIFF v1.0 defines the format of 15 directory entries and EntryTag entries. All of these entries (except the End of Directory entry) are optional, and many may appear only once in a SPIFF file if used. The exact format of each entry is documented in the SPIFF specification and summarized in Table SPIFF-1:

TABLE SPIFF-1: ***Standard SPIFF Directory Entries***

Name	Use	EntryTag	Multiple Entries
End Of Directory	End of directory marker	1	No
Transfer Characteristics	Gamma correction	2	No
Component Registration	Location of components	3	No
Image Orientation	Rotated, flipped	4	No
Thumbnail Image	Thumbnail header	5	Yes
Image Title	Text	6	Yes
Image Description	Text	7	Yes
Time Stamp	Time and date	8	No
Version Number	Image version number	9	Yes
Creator Identification	Text	10	Yes
Protection Indicator	Level of authenticity	11	No
Copyright Information	Text	12	Yes
Contact Information	Text	13	Yes
Tile Index	Pointer to tiles	14	No
Scan Index	Pointers to scans	15	No
Set Reference	Relationship to other files	16	Yes

Ex.1007 / Page 858 of 1158
APPLE INC.

End of Directory indicates the end of the directory. This entry contains no associated data and is immediately followed by image data.

Transfer Characteristics describes the gamma correction value of the image.

Component Registration describes the positioning of samples within components (that is, color elements within a sample) relative to the samples within other components.

Image Orientation specifies which edge of the image is the top and whether the image is flipped.

Thumbnail Image is a lower resolution version of the primary image.

Image Title is a textual description for the image.

Image Description is an additional textual description of the image data.

Time Stamp is an ISO 8601 standard date and time string in the format *YYYY-MM-DD* and *HH:MM:SS.mmmZ*.

Version Number is the number of revisions of the image.

Creator Identification is textual information that describes the creator of the image data and file.

Protection Indicator specifies the usage rights of the image data.

Copyright Information contains the copyright text for the image data.

Contact Information is a textual description of how to contact the creator and/or owner of the image data.

Tile Index contains a listing of all the offset values for the tiles in the image data.

Scan Index contains a listing of all the offset values for the scans in the image data.

Set Reference is a reference number typically used to identify the file as related to other groups of files.

Indirect storage is only allowed for the Thumbnail Image, Scan Index, Tile Index, and all textual directory entries (Image Name, Image Description, and so forth). Each of these entries contains a field specifying the offset of the data from the beginning of the file. If this value is 0, then the data is stored directly within the entry. It is recommended that direct storage be used whenever possible (that is, when the entry data is less than 64K in size).

Ex.1007 / Page 859 of 1158
APPLE INC.

SPIFF requires that the Scan Index and Tile Index entries only be stored as indirect data. These indexes are only useful to decoders that can perform random access on a file and are likely to be built on the fly by encoders, so requiring them to be stored at the end of a SPIFF file is a reasonable thing to do.

The SPIFF format also provides for application-specific directory entries that would store any information not supported by the standard directory entries defined in the SPIFF specification. These directory entries are identified by setting the bits 23:21 in the EntryTag to all ones.

There is currently no process for registering or reserving application-specific directory tags, so it is recommended that additional identifying information be present in the entry data. This will help prevent data interpretation problems caused by duplicate application-specific directory tags defined by different organizations. SPIFF readers should, of course, ignore any directory entries with an unrecognized tag value.

Image Data

Although thought of as a file format for JPEG data, SPIFF is quite capable of supporting data compressed using the Huffman 1D, MR (Modified READ), MMR (Modified Modified READ), and JBIG data compression methods as well. And uncompressed data storage is also supported, as you might expect.

The data area of each variation of SPIFF contains the complete data stream defined by the underlying compression standards. For example, SPIFF-JPEG files contain a complete JPEG interchange data stream as defined by ITU-T T.81; SPIFF-JBIG files contain a complete JBIG bi-level image entity as defined by ITU-T T.82, and so on.

Application Profiles

An application profile is a predefined set of features that a SPIFF file reader must support to be able to interpret the contents of a SPIFF file. The ProfileId field of the header contains a value that specifies the profile required by the data stored in the SPIFF file. This field makes it possible for a file reader to determine the contents of the file without reading the entire directory.

The profiles currently defined apply to baseline JPEG data, progressive mode JPEG data (for low-speed communications applications), bi-level image data,

Ex.1007 / Page 860 of 1158
APPLE INC.

and continuous-tone, color, or gray-scale facsimile images. The following profile values are currently defined:

0 No profile

1 Continuous-tone base profile
Compression is 5 (JPEG)

ColorSpace is 3 (YCbCr RGB) or 8 (gray-scale)

No Image Orientation directory entry

No indirect directory data allowed

Image data is encoded with baseline JPEG as a single scan with interleaved components

2 Continuous-tone progressive profile
Continuous-tone base profile

Support for DCT progressive mode JPEG with spectral selection and full progressive

3 Bi-level facsimile profile
Support per ITU-T T.4 (Modifed Huffman and Modified READ), ITU-T T.6 (Modified Modified READ), or ITU-T T.82lISO/IEC 11544 (JBIG)

4 Continuous-tone facsimile profile
8-bits per sample (12-bits optional)
2:1 Chrominance subsampling in each direction (no subsampling optional)
CIE Standard Illuminant D50 (custom illuminant optional)
Default gamut range (custom gamut range optional)

The profile value should be 0 (no profile) if the file uses features that do not fall into any of the other profiles.

Converting JFIF to SPIFF-JPEG

When you first read through the SPIFF specification, you may conclude that it is easy to convert a JFIF file to a SPIFF-JPEG file. Just fill in the SPIFF header fields from information found in the JFIF file, write out the header and an End Of Directory entry, and then append the entire JFIF file itself to the SPIFF-JPEG file.

Ex.1007 / Page 861 of 1158
APPLE INC.

This technique for JFIF-to-SPIFF-JPEG conversion has the advantage of being a simple and quick conversion that also preserves the JFIF markers, allowing older decoders to read the resolution and thumbnail data present in the JFIF data stream. It also allows a SPIFF-JPEG-to-JFIF conversion program to recover the original JFIF data from the SPIFF file without the need to perform any type of data conversions.

This technique, however, has the disadvantage of violating the JFIF specification by including a JFIF APP0 marker that does not immediately follow the SOI marker. While many JFIF encoders may not care about this violation, it is possible that some JFIF decoders will complain. The greatest disadvantage, however, is this is not the proper way to use the SPIFF format.

The JFIF format was created to embed ancillary information directly within a raw JPEG data stream. JFIF accomplishes this by using APP0 markers followed by the ancillary data. Such data defined by the JFIF specification includes number of components, sample precision, image height and width, thumbnail data, and application-specific data.

A primary purpose of SPIFF-JPEG is to replace the function of JFIF's APP0 markers with SPIFF's header and directory information. While it is valid to store JPEG data in a SPIFF file that contains APP0 markers, it is not in the spirit of the design and use of the SPIFF format.

What are the rules for a proper JFIF-to-SPIFF-JPEG conversion? The goal is to convert all ancillary information in the JFIF file to the equivalent SPIFF information structures and only store a raw JPEG data stream in the SPIFF-JPEG file. We offer the following guidelines:

- Convert all APPn data to the equivalent SPIFF header information.

  The following JFIF and JPEG information must be used to initialize the fields of the SPIFF header:

Description	SOF0 (width)
SPIFF Header Field	Color space
JFIF or JPEG marker code	ColorSpace
Number of color components	SOF0 (components)
NumberComponents	Type of resolution units
SOF0 (components)	ResolutionUnits
Number of bits per sample	APP0 (units)

Ex.1007 / Page 862 of 1158
APPLE INC.

BitsPerSample	Vertical resolution
SOF0 (sample precision)	VerticalResolution
Number of lines in image	APP0 (Ydensity)
ImageHeight	Horizontal resolution
SOF0 (height)	HorizontalResolution
Number of samples per line	APP0 (Xdensity)
ImageWidth	

Note that the value of ColorSpace will be 3 if the JPEG number of color components value is 3, or 8 if the same JPEG marker data is 1. These ColorSpace values correspond to the two color spaces allowed by the JFIF spec. Non-JFIF, raw JPEG data files may convert to other SPIFF color-space codes; for example, Adobe Photoshop can emit CMYK JPEG files.

- Convert any thumbnail data stored in the JFIF APP0 marker segment or extension marker segment to SPIFF thumbnail directory entries.
- Convert JPEG comment blocks (COM markers) to SPIFF text entries.

  Here is the black art of JFIF-to-SPIFF-JPEG conversion. The JPEG standard does not define the type of information that is stored in a JPEG comment block. It can be anything from your name to the Gettysburg Address to a field of NULL values. It's up to the user and/or application creating the JPEG data stream.

  The storage of generic blocks of "unspecified" or "miscellaneous" text is not directly supported by the SPIFF format. The information content of textual data that may be stored in a SPIFF directory entry is somewhat rigidly defined to be the name of the SPIFF file creator, image title, image description, copyright information, and so on.

  A converter may choose to store any JPEG comment block information it finds in the SPIFF image description directory entry, but it may not always be the proper place for this information. Another possible solution is to store miscellaneous text in application-specific directory entries, as provided for by the SPIFF specification. This, however, will effectively hide the comment block information from every SPIFF reader that doesn't recognize your application-specific directory tag (which is probably most of them). Your last—and possibly best—solution is to simply leave the comment block in the JPEG data stream. At least this will make it possible for any program that reads JPEG comment blocks to retrieve the information.

Ex.1007 / Page 863 of 1158
APPLE INC.

- Do not write out any indirect directory entries. Indirect data requires random access of the SPIFF file. Many JPEG decoders read the data file strictly serially and therefore cannot conveniently handle indirect data. You should expect that a significant percentage of SPIFF readers will simply ignore any indirect directory entries. If you have a choice of direct or indirect storage for a directory entry, direct storage is the better option.

  It is worth noting that storing indirect data is not harmful. All JFIF readers should stop at the EOI (End of Image) JPEG marker at the end of the compressed data, and should therefore never reach the indirect data. The recommendation against indirect data is made just to accommodate simple-minded SPIFF decoders that don't handle indirect entries.

For Further Information

SPIFF is part of the International Standard and Recommendation "Digital Compression and Coding of Continuous-Tone Still Images," which is published by the ITU in three parts:

ITU-T T.81 Requirements and Guidelines ITU-T T.83 Compliance Testing ITU-T T.84 Extensions

The same standards are also published by the ISO/IEU:

ISO/IEC 10918-1 Requirements and Guidelines ISO/IEC 10918-2 Compliance Testing ISO/IEC 10918-3 Extensions

The actual document containing the SPIFF specification is the ISO/IEC 10918-3 standard, "Digital Compression and Coding of Continuous-Tone Still Images: Extensions." This document is also published as the ITU-T Recommendation T.84 under the same title. Recommendation T.84 is available directly from ITU:

International Telecommunication Union (ITU)
Information Services Department
Place des Nations
1211 Geneva 20
Switzerland
Voice: +41 22 730-6666 or 730-5554
Fax: +41 22 730 533
Email: *helpdesk@itu.ch*
X.400: S=helpdesk; A=arcom; P=itu; C=ch

Ex.1007 / Page 864 of 1158
APPLE INC.

You can also order these documents via the ITU and ISO Web pages at:

http://www.itu.ch
http://www.iso.ch

For information about ordering, you can also check out:

ftp://ftp.uu.net/graphics/jpeg/jpeg.documents.gz

A future version of the Independent JPEG Group's JPEG library (found on the CD-ROM that accompanies this book) will implement support for SPIFF and will be an excellent source of SPIFF code.

Ex.1007 / Page 865 of 1158
APPLE INC.

Sun Icon

NAME:	Sun Icon
ALSO KNOWN AS:	ICO
TYPE:	Bitmap
COLORS:	Mono
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	64×64 pixels
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	Sun Microsystems
PLATFORM:	SunOS
SUPPORTING APPLICATIONS:	Many UNIX-based
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	Sun Raster

USAGE: Used to store iconic images found in the Sun GUI environments.

COMMENTS: Sun Icon is an ASCII representation of a bitmap image format.

Overview

The icons found in the Open Look and SunView Graphical User Interfaces available on the Sun Microsystems UNIX-based platforms are stored in a simple format known as the Sun Icon format.

File Organization

Sun Icon files are ASCII text files that may be created and modified using a simple text editor. Sun icons are typically 64×64 pixels in size and contain black-and-white image data. Files contain an ASCII header followed by a hexadecimal representation of the bitmapped image data.

The header is found in the first 78 bytes of the icon file. The header contains five fields of information composed of printable ASCII characters. Each field has a *keyword=value* syntax and is delineated by a comma and a space character.

Ex.1007 / Page 866 of 1158
APPLE INC.

The header begins and ends with the standard C comment tokens /* */. A linefeed character (ASCII 0Ah) is preset at offset 49h within the header.

File Details

When you use a text editor to examine a Sun Icon header, you can see the format shown below:

```
/* Format_version=1, Width=64, Height=64, Depth=1,
   Valid_bits_per_item=16
 */
```

Format_version is the version of the icon file format and is always 1.

Width and Height are the size of the icon in pixels; both are typically set to a value of 64.

Depth is the number of bits per pixel in the icon image data and is usually 1.

Valid_bits_per_item is the number of bits of image data contained in each item of hexadecimal bitmapped data. Typical values for this field are 16 and 32.

The image data that follows the header is a series of hexadecimal numbers called *items*. Each item represents a number of pixels equal to the Valid_bits_per_item value divided by the Depth value. For images with a Valid_bits_per_item value of 16 and a Depth of 1, each hexadecimal number represents 16 pixels and is two bytes in size.

Items are separated by commas, and every eighth item is delimited by a linefeed character. All hexadecimal numbers begin with the standard C hexadecimal notational prefix 0x. The following illustration is a complete Sun Icon image file. The <LF> symbols indicate the location of a linefeed character.

```
/*
Format_version=1,Width=64,Height=64,Depth=1,Valid_bits_per_item=16<LF>
*/<LF>
0x0000,0x0000,0x0000,0x0000,0x01E0,0x3C00,0x0000,0x0000,<LF>
0x1F5C,0xEB00,0x0000,0x0000,0x7AAB,0xB5C0,0x0000,0x0000,<LF>
0xD555,0x6B00,0x0000,0x0000,0xFAAA,0xDC00,0x0000,0xC600,<LF>
0x0FF7,0x0000,0x0001,0x2900,0x0000,0x0000,0x0000,0x1000,<LF>
0x0000,0x0000,0x8000,0x0000,0xE000,0x0000,0x8000,0x0000,<LF>
0x5804,0x0001,0xC318,0x0000,0xAE06,0x0001,0x44A4,0x018C,<LF>
0x5587,0x0001,0xC040,0x0252,0xAE05,0x0001,0x6000,0x0020,<LF>
0xF006,0x8082,0xA000,0x0000,0x0005,0x4083,0x6000,0x0000,<LF>
0x0002,0xC0C2,0xB020,0x0000,0x0003,0x6145,0x5060,0x0000,<LF>
0x0002,0xB1A6,0xB060,0x0000,0x0003,0x5165,0x50A0,0x00C0,<LF>
0x0001,0xB9B6,0xA9A0,0x0340,0x0001,0x575D,0x5940,0x0D80,<LF>
0x0041,0xAAAA,0xAAC0,0x7B00,0x0071,0xD77F,0xD541,0xD500,<LF>
```


Ex.1007 / Page 867 of 1158
APPLE INC.

```
0x803C,0xAF80,0x3EC6,0xAA00,0x602B,0xD815,0x035D,0x5400,<LF>
0x7815,0xE140,0x28EA,0xA800,0x2E1A,0x8400,0x0235,0x5800,<LF>
0x15CB,0x1000,0x009A,0xB000,0x0ABC,0x4000,0x0005,0x6000,<LF>
0x055D,0x0000,0x0022,0xC000,0x0550,0x0000,0x0009,0x4000,<LF>
0x02A4,0x7800,0x0000,0xFFF0,0x01A0,0x8400,0x0002,0xAAA0,<LF>
0x0150,0x8000,0x0000,0x5540,0xFF80,0x8000,0x0001,0x2A80,<LF>
0x55A0,0x80E7,0x3700,0x3500,0x2B00,0x7842,0x1880,0x9B00,<LF>
0x1540,0x0442,0x1080,0x1600,0x1E00,0x0442,0x1080,0x4C00,<LF>
0x0A80,0x8442,0x1080,0x0C00,0x0600,0x8442,0x1080,0x2B80,<LF>
0x0680,0x783C,0x38C0,0x0AF0,0x3C00,0x0000,0x0000,0x155F,<LF>
0xED00,0x0000,0x0000,0x06AC,0x5400,0x0000,0x0000,0x1550,<LF>
0x2D00,0x0000,0x0000,0x06A0,0xFFD5,0x5555,0x5555,0x5FFF,<LF>
0xFF88,0x8888,0x8888,0xB7FF,0xDDE2,0x2222,0x2222,0x5DDD,<LF>
0xBBD4,0x4444,0x4444,0xFBBB,0xFFFF,0xFFFF,0xFFFF,0xFFFF,<LF>
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,<LF>
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,<LF>
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,<LF>
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,<LF>
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,<LF>
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000
```

For Further Information

For further information about the Sun Icon format, refer to the following files on SunOS systems (and included on our CD-ROM):

/usr/include/suntool/icon.h
/usr/include/suntool/icon_load.h

These files contain the declaration for the Sun Icon format, as well as other information about the format.

You can also contact Sun at:

Sun Microsystems Incorporated
2550 Garcia Avenue
Mountain View, CA 94043
Voice: 415-960-1300
FTP: *ftp://ftp.sun.com/*
WWW: *http://www.sun.com/*

There are also many available UNIX-based tools for reading, writing, and converting Sun Icon files. See the pbmplus package on the CD-ROM.

Ex.1007 / Page 868 of 1158
APPLE INC.

NAME:	Sun Raster
ALSO KNOWN AS:	RAS
TYPE:	Bitmap
COLORS:	Variable
COMPRESSION:	RLE
MAXIMUM IMAGE SIZE:	Variable
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	Sun Microsystems
PLATFORM:	SunOS
SUPPORTING APPLICATIONS:	Many UNIX-based
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	Sun Icon

USAGE: Sun Raster is the native bitmap format of the Sun UNIX platforms.

COMMENTS: A simple bitmap format with wide distribution, particularly in the UNIX world.

Overview

The Sun Raster image file format is the native bitmap format of the Sun Microsystems UNIX platforms using the SunOS operating system. This format is capable of storing black-and-white, gray-scale, and color bitmapped data of any pixel depth. The use of color maps and a simple Run-Length data compression are also supported. Typically, most images found on a SunOS system are Sun Raster images, and this format is supported by most UNIX imaging applications.

File Organization

The basic layout of a Sun Raster file is a header, followed by an optional color map, and then by the bitmapped image data.

Ex.1007 / Page 869 of 1158
APPLE INC.

File Details

The Sun Raster file header is 32 bytes in length and has the following format:

```
typedef struct _SunRaster
{
  DWORD MagicNumber;      /* Magic (identification) number */
  DWORD Width;            /* Width of image in pixels */
  DWORD Height;           /* Height of image in pixels */
  DWORD Depth;            /* Number of bits per pixel */
  DWORD Length;           /* Size of image data in bytes */
  DWORD Type;             /* Type of raster file */
  DWORD ColorMapType;     /* Type of color map */
  DWORD ColorMapLength;   /* Size of the color map in bytes */

} SUNRASTER;
```

MagicNumber is used to identify a file as a Sun Raster image and always contains the value 59a66a95h. This value is stored in big-endian byte order, as are the entire contents of every Sun Raster file. Reading this magic number using the little-endian byte order (as is possible on the Intel-based Sun 386i system) produces the value 956aa659h, a clue that you are not reading the raster file using the proper byte order.

Width and Height specify the size of the image in pixels. The width of a scan line is always a multiple of 16 bits, padded when necessary.

Depth is the number of bits per pixel of the image data. The typical values for this field are 1, 8, 24, and 32; a value of 32 indicates 24-bit values with a pad byte preceding the pixel values. Note that 24- and 32-bit pixel data (assuming no color map) is in BGR format, rather than RGB, unless the image type is RGB.

Length is the actual size of the bitmapped data in the bitmap file (that is, the file size minus the header and the color map length). Do not expect this value to always be accurate, however. In the original release of the Sun Raster format, this field indicated the type of encoding used on the bitmapped data and was always set to 00h (no encoding). In the second release, this field was renamed and was used to indicate the length of the bitmapped data. Therefore, older raster files will appear to have a length of 00h. In this case, the Length must be calculated by multiplying together the values of the Height, Width, and Depth fields.

Type is the version (or flavor) of the bitmap file. The following values are typically found in the Type field:

Ex.1007 / Page 870 of 1158
APPLE INC.

0000h	Old
0001h	Standard
0002h	Byte-encoded
0003h	RGB format
0004h	TIFF format
0005h	IFF format
FFFFh	Experimental

Both Old and Standard formats are the same. They indicate that the image data within the file is not compressed, and most Sun Raster files you will encounter are stored in this manner.

The Byte-encoded type indicates that the image data is compressed using a Run-length encoding scheme (described later in this section).

The TIFF and IFF format types indicate that the raster file was originally converted from either of these file formats.

The Experimental type is implementation-specific and is generally an indication that the image file does not conform to the Sun Raster file format specification.

ColorMapType indicates the type of color map included in the raster file, or whether a color map is included at all. The following values are typically found in the ColorMapType field:

0000h	No color map
0001h	RGB color map
0002h	Raw color map

ColorMapLength contains the number of bytes stored in the color map.

If ColorMapType is 0000h (no color map), ColorMapLength is 0000h. If ColorMapType is 0001h (RGB color map) or 0002h (raw color map), ColorMapLength is the number of bytes in the color map. In the case of an RGB color map, the colors are separated into three planes, stored in RGB order, with each plane being one-third the size of the ColorMapLength value. For example, a 256-element color map for a 24-bit image consists of three 256-byte color planes and has a length of 768 bytes (Depth = 24, ColorMapType = 01h, ColorMapLength = 768). A raw color map is any other type of color map not defined by the Sun Raster file format and is stored as individual byte values.

Bitmap files with a Depth of 1 contain 2-color image data. Typically, 1-bit bitmap images do not have a color map. Each bit in the bitmap represents a

Ex.1007 / Page 871 of 1158
APPLE INC.

pixel, with a value of 0 representing black and a value of 1 representing white (the bits are stored most significant bit first within each byte). If a color map is present in a 1-bit image file, it is a 2-color map, each color being 24 bits in length (Depth = 1, ColorMapType = 01h, ColorMapLength = 6). Each bit of image data is then an index pointing to one of these two colors in the map.

Raster files with a Depth of 8 may contain either color or gray-scale image data. Images with pixels eight or fewer bits in depth do not include a color map (Depth = 8, ColorMapType = 00h, ColorMapLength = 0). Each byte of image data contains the value of the color it stores. If a color map is present in an 8-bit raster file, the pixel values are index pointers into the color map. Such an image, although it may contain a 24-bit color map (Depth = 8, ColorMapType = 01h, ColorMapLength = 768), can contain only a maximum of 256 colors.

Raster files with a Depth of 24 (or 32) normally do not have color maps. Instead, the colors values are stored directly in the image data itself (truecolor bitmap). If a 24-bit image has a color map, it is either a raw color map, or an RGB color map that contains more than 256 elements.

The Run-length encoding (RLE) scheme optionally used in Sun Raster files (Type = 0002h) is used to encode bytes of image data separately. RLE encoding may be found in any Sun Raster file regardless of the type of image data it contains.

The RLE packets are typically three bytes in size:

- The first byte is a Flag Value indicating the type of RLE packet.
- The second byte is the Run Count.
- The third byte is the Run Value.

A Flag Value of 80h is followed by a Run Count in the range of 01h to FFh. The Run Value follows the Run count and is in the range of 00h to FFh. The pixel run is the Run Value repeated Run Count times.

There are two exceptions to this algorithm. First, if the Run Count following the Flag Value is 00h, this is an indication that the run is a single byte in length and has a value of 80h. And second, if the Flag Value is not 80h, then it is assumed that the data is unencoded pixel data and is written directly to the output stream.

For example, a run of 100 pixels with the value of 0Ah would encode as the values 80h 64h 0Ah. A single pixel value of 80h would encode as the values 80h 00h. The four unencoded bytes 12345678h would be stored in the RLE stream as 12h 34h 56h 78h.

Ex.1007 / Page 872 of 1158
APPLE INC.

Note also that the Sun Raster bitmap is read as if it is a single stream of data. Therefore, the encoding of pixel runs does not stop at the end of each scan line.

For Further Information

For further information about the Sun Raster format, see the descriptions included on the CD-ROM that accompanies this book and the SunOS manual page entitled *rasterfile*. The man page entry describes only the basic layout of the Sun Raster format. Information about the order of bit planes or the RLE encoding used on the image data is not included. The following file contains the Sun Raster header declaration and field values:

/usr/include/rasterfile.h

You can also contact Sun Microsystems at:

Sun Microsystems Incorporated
2550 Garcia Avenue
Mountain View, CA 94043
Voice: 415-960-1300
FTP: *ftp://ftp.sun.com/*
WWW: *http://www.sun.com/*

In addition, there are also many publicly available UNIX-based image file viewers and converters that support the Sun Raster format. See the FBM, ImageMagick, pbmplus, xli, xloadimage, and xv packages on the CD-ROM.

Ex.1007 / Page 873 of 1158
APPLE INC.

TDDD

NAME:	TDDD
ALSO KNOWN AS:	Turbo Silver 3D Data Description, T3D
TYPE:	3D vector and animation
COLORS:	16 million
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	Impulse
PLATFORM:	All
SUPPORTING APPLICATIONS:	Turbo Silver
SPECIFICATION ON CD:	Yes (summary description)
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	Interchange File Format

USAGE: Interchange of 3D image information, primarily on the Amiga.

COMMENTS: A slightly modified version of IFF that is widely used.

Overview

The TDDD (Turbo Silver 3D Data Description) format is used to store object data created by the Turbo Silver 3.0 application from Impulse. The TDDD format is actually the Electronic Arts Interchange File Format (IFF) with modifications to two of its chunks. For this reason, please read the IFF article before you read this article about TDDD.

File Organization

Like IFF files, TDDD files consist of a series of sections called chunks.

The FORM chunk of a TDDD file uses only two types of IFF chunks: INFO and OBJ.

Ex.1007 / Page 874 of 1158
APPLE INC.

The INFO chunk stores information describing observer data and appears in *.cell* files. Each INFO chunk contains standard IFF sub-chunks. The INFO chunk is optional and might not appear at all in the FORM chunk.

The OBJ chunk contains data which describes an object hierarchy and appears in both cell and object files. One or more OBJ chunks are contained in the FORM chunk and each OBJ chunk contains one or more sub-chunks.

There are three types of OBJ sub-chunks:

1. EXTR describes an "external" object in a separate file.
2. DESC describes a single node of a hierarchy.
3. TOBJ marks the end of a hierarchy chain.

Each hierarchy node is described either by an EXTR chunk or by a DESC and TOBJ chunk pair.

The TOBJ sub-chunks contain no data and have a length of zero. The DESC and EXTR sub-chunks contain sub-chunks as defined by the IFF file format. Unrecognized sub-chunks are skipped over by TDDD readers, and the default values are assumed for any missing sub-chunks.

The object hierarchy contains a head object and one or more brothers. Each brother may have child objects; the children may have grandchildren; and so on. The brother nodes are stored in a doubly linked list, and each node has a pointer to a doubly linked "child" list. (If no child is present, then the pointer is NULL.) Child lists point to grandchildren lists and back to their parent, and so on.

Each of the "head" brothers is written in a separate OBJ chunk, along with all its descendants. Each child, grandchild, and so on in the descendant hierarchy begins a DESC chunk and ends with a TOBJ chunk. Objects stored in external files are described only with a single EXTR chunk. The children and grandchildren of this external object are also stored in the same external file.

File Details

Several data types are used to represent fields within several sub-chunks. RGB values are always represented by an array of three BYTE values and the values are stored in RGB order. Fractional (FRACT) and point values are stored as LONG values. A VECTOR is an array of three FRACT values and a MATRIX is

Ex.1007 / Page 875 of 1158
APPLE INC.

an array of three VECTOR values. With this in mind, here are the structures for these data types:

```
typedef BYTE COLOR[3];  /* Red, Green, and Blue values */
typedef LONG FRACT;     /* Point */
typedef struct _Vectors
{
  FRACT X;
  FRACT Y;
  FRACT Z;

} VECTOR;
typedef struct _Matrices
{
  VECTOR I;
  VECTOR J;
  VECTOR K;

} MATRIX;
```

The following structure is used in generating animated cells from a single cell. It can be attached to an object or to the camera. It is also used for Turbo Silver's "extrude along a path" feature.

```
typedef struct _Story
{
  BYTE   Path[18];      /* Name of object */
  VECTOR Translate;     /* Translate vector */
  VECTOR Rotate;        /* Rotate vector */
  VECTOR Scale;         /* Scale vector */
  WORD  Info;           /* Coordinate flags */

} STORY;
```

Path is the name of a named object in the cell data.

Translate is not used.

Rotate specifies rotation angles about the X, Y, and Z axes of the vector.

Scale specifies X, Y, and Z factors of the scale vector.

Info contains a collection of bitfield flags with the following definitions:

0x0001	ABS_TRA	Translate in world coordinates (not used)
0x0002	ABS_ROT	Rotation in world coordinates
0x0004	ABS_SCL	Scaling in world coordinates
0x0010	LOC_TRA	Translate in local coordinates (not used)
0x0020	LOC_ROT	Rotation in local coordinates

Ex.1007 / Page 876 of 1158
APPLE INC.

0x0040	LOC_SCL	Scaling in local coordinates
0x0100	X_ALIGN	Not used
0x0200	Y_ALIGN	Align Y axis to path's direction
0x0400	Z_ALIGN	Not used
0x1000	FOLLOW_ME	Children follow parent on path

INFO Chunk

The following sub-chunk structures are found only in the INFO chunk. All of these INFO sub-chunks are optional, as is the INFO chunk itself. If a sub-chunk is not present, then its default value is assumed. The base default values for an INFO chunk are the following:

- No brushes, stencils, or textures defined
- No story for the camera
- Horizon and zenith and ambient light colors set to black
- Fade color set to (80, 80, 80)
- Unrotated, untracked camera at (-100, -100, 100)
- Global properties array set to [30, 0, 0, 0, 0, 100, 8, 0]
- Global properties array set to [30, 0, 0, 0, 0, 100, 8, 0]

The BRSH sub-chunk defines a brush. There may be up to eight brushes defined in an INFO chunk.

```
typedef struct _Brush
{
  WORD BrushNumber;      /* Brush number (0 to 7) */
  CHAR FileName[80];     /* IFF ILBM filename */

} BRSH;
```

BrushNumber is the identification number of the brush, and this value may be in the range of 0 to 7.

FileName is the name of the IFF file which stores the brush information.

The STNC sub-chunk defines a stencil. There may be up to eight stencils defined in an INFO chunk.

```
typedef struct _Stencil
{
  WORD StencilNumber;    /* Stencil number (0 to 7) */
```


Ex.1007 / Page 877 of 1158
APPLE INC.

```
    CHAR FileName[80];      /* IFF ILBM filename */

} STNC;
```

StencilNumber is the identification number of the stencil, and this value may be in the range of 0 to 7. FileName is the name of the IFF file which stores the stencil information.

The TXTR sub-chunk defines a text resource. There may be up to eight resources defined in an INFO chunk.

```
typedef struct _Text
{
  WORD TextNumber;       /* Text number (0 to 7) */
  CHAR FileName[80];     /* Code module name */

} TXTR;
```

TextNumber is the identification number of the resource, and this value may be in the range of 0 to 7.

FileName is the name of a code module that can be loaded using the LoadSeg() function found in Turbo Silver.

The OBSV sub-chunk specifies the location, position, and focal length of the camera observer. The rotation angles are in degrees and specify the degree of rotation around the X, Y, and Z axes.

```
typedef struct _Observer
{
  VECTOR Camera;         /* Camera position */
  VECTOR Rotate;         /* Camera rotation angles */
  FRACT  Focal;          /* Camera focal length */

} OBSV;
```

The OTRK sub-chunk specifies the name of an object; otherwise the camera always follows the tracked object.

```
typedef struct _ObjectTrack
{
  BYTE TrackName[18];    /* Name of tracked object */

} OTRK;
```

The OSTR sub-chunk contains the story information for the camera.

```
typedef struct _ObjectStory
{
  STORY CStory;          /* STORY structure for the camera */
} OSTR;
```


Ex.1007 / Page 878 of 1158
APPLE INC.

FADE contains the parameters for a fading operation.

```
typedef struct _Fade
{
  FRACT FadeAt;          /* Distance to start fade */
  FRACT FadeBy;          /* Distance of total fade */
  BYTE  Pad;             /* Pad byte (always 0) */
  COLOR FadeTo;          /* RGB color to fade to */

} FADE;
```

The SKYC sub-chunk defines the color of a rendered sky.

```
typedef struct SkyColor
{
  BYTE    Pad1;          /* Pad byte (always 0) */
  COLOR   Horizon;       /* Horizon color */
  BYTE    Pad2;          /* Pad byte (always 0) */
  COLOR   Zenith;        /* Zenith color */
} SKYC;
```

The AMBI sub-chunk defines the ambient light color of the rendering.

```
typedef struct _AmbientLightColor
{
  BYTE    Pad;           /* Pad byte (always 0) */
  COLOR   Ambient;       /* Ambient light color */

} AMBI;
```

The GLB0 sub-chunk contains an array of eight global property values used by Turbo Silver.

```
typedef struct _GlobalProperties
{
  BYTE Props[8];         /* Eight global properties */

} GLB0;
```

The elements are defined as follows:

0	GLB_EDGING	Edge level value
1	GLB_PERTURB	Perturbance value
2	GLB_SKY_BLEND	Sky blending factor
3	GLB_LENS	Lens type
4	GLB_FADE	Sharp/fuzzy focus
5	GLB_SIZE	Apparent size
6	GLB_RESOLVE	Resolve depth
7	GLB_EXTRA	Genlock sky flag

Ex.1007 / Page 879 of 1158
APPLE INC.

GLB_EDGING and GLB_PERTURB correspond to the edging and perturbance values heuristics control in ray tracing.

GLB_SKY_BLEND is zero for no blending and 255 for full blending.

The GLB_LENS value corresponds to the boxes in the "camera" requester, and may be 0 (manual), 1 (wide angle), 2 (normal), 3 (telephoto), or 4 (custom).

GLB_FADE turns the "fade" feature ON (non-zero) and OFF (zero).

GLB_SIZE is 100 times the "custom size" parameter in the camera requester and is used to set the focal length for a custom lens. GLB_RESOLVE specifies the number of rays the ray tracer will shoot for a single pixel.

The GLB_EXTRA flag indicates if the sky is colored or is set to the "genlock" color (color 0 to black) in the final picture. If "genlock" is set in TurboSilver, a "zero color" is written into the bitplanes for genlock video to show through.

DESC Chunk

The following sub-chunk structures are only found in the DESC chunk. Many of these sub-chunks are optional (the SHAP sub-chunk is required to appear), and all have default values if they are not present. Note that if there is a FACE chunk, there must also be a CLST, an RLST, and a TLST sub-chunk as well, all with matching Count fields.

The default for the DESC chunk sub-chunks are: Colors set to (240,240,240); reflection and transmission coefficients set to zero; illegal shape; no story or special surface types; position at (0,0,0); axes aligned to the world axes; size fields all 32.0; intensity at 300; no name; no points/edges or faces; texture parameters set to zero; refraction type 0 with index 1.00; specular, hardness and roughness set to zero; blending at 255; glossy off; phong shading on; not a light source and not brightly lit.

The NAME sub-chunk contains the name of the object itself and is used by a number of operations, including camera tracking and specifying story paths.

```
typedef struct _ObjectName
{
  BYTE Name[18];          /* The name of the object */

} NAME;
```


Ex.1007 / Page 880 of 1158
APPLE INC.

The SHAP sub-chunk defines the visible appearance of an object.

```
typedef struct _ObjectShape
{
  WORD Shape;           /* Object type */
  WORD Lamp;            /* Lamp type */

} SHAP;
```

Shape values include:

0 Sphere
1 Stencil
2 Axis
3 Facets
4 Surface
5 Ground

Lamp values include:

0 No lamp
1 Light is sunlight
2 Light is from a lamp, and intensity falls off with distance

The POSI sub-chunk specifies the position of an object in a rendering.

```
typedef struct _ObjectPosition
{
  VECTOR  Position;     /* The object's position in space */

} POSI;
```

Legal coordinates are in the range -32768 to 32767 and 65535/65536.

The AXIS sub-chunk describes a direction (orthogonal unit) vector for the object coordinate system.

```
typedef struct _VectorAxis
{
  VECTOR  XAxis;              /* X axis of vector */
  VECTOR  YAxis;              /* Y axis of vector */
  VECTOR  ZAxis;              /* Z axis of vector */

} AXIS;
```


Ex.1007 / Page 881 of 1158
APPLE INC.

The SIZE sub-chunk is used by a variety of operations requiring a vector size value.

```
typedef struct _Size
{
  VECTOR Size;                /* Object size */

} SIZE;
```

The PNTS sub-chunk stores all the points defining a custom objects.

```
typedef struct _Points
{
  WORD   PCount;              /* Point count */
  VECTOR Points[PCount];      /* Points */

} PNTS;
```

The EDGE sub-chunk contains the edge list for custom objects.

```
typedef struct _EdgeList
{
  WORD ECount;                /* Edge count */
  WORD Edges[ECount][2];      /* Edges */

} EDGE;
```

The FACE sub-chunk contains the triangle (face) list for custom objects.

```
typedef struct _FaceList
{
  WORD   TCount;              /* Face count */
  WORD   Connects[TCount][3]; /* Faces */

} FACE;
```

The COLR sub-chunk contains the main object color coefficients.

```
typedef struct _Color
{
  BYTE  Pad;                  /* Pad byte (always 0) */
  COLOR Color;                /* RGB color */

} COLR;
```

The REFL sub-chunk contains the main object reflection coefficients.

```
typedef struct _Reflection
{
  BYTE  Pad;                  /* Pad byte (always 0) */
  COLOR Color;                /* RGB color */

} REFL;
```


Ex.1007 / Page 882 of 1158
APPLE INC.

The TRAN sub-chunk contains the main object transmission coefficients.

```
typedef struct _Transmission
{
  BYTE  Pad;                    /* Pad byte (always 0) */
  COLOR Color;                  /* RGB color */

} TRAN;
```

The CLST sub-chunk contains the main object color coefficients for each face in custom objects.

```
typedef struct _ColorList
{
  WORD Count;                   /* Count of colors */
  COLOR Colors[Count];          /* Colors */

} CLST;
```

The count should match the face count in the FACE chunk and the ordering corresponds to the face order.

The RLST sub-chunk contains the main object reflection coefficients for each face in custom objects.

```
typedef struct _ReflectionList
{
  WORD Count;                   /* Count of colors */
  COLOR Colors[Count];          /* Colors */

} RLST;
```

The count should match the face count in the FACE chunk and the ordering corresponds to the face order.

The TLST sub-chunk contains the main object transmission coefficients for each face in custom objects.

```
typedef struct _TransmissionList
{
  WORD Count;                   /* Count of colors */
  COLOR Colors[Count];          /* Colors */

} TLST;
```

The count should match the face count in the FACE chunk and the ordering corresponds to the face order.

Ex.1007 / Page 883 of 1158
APPLE INC.

The TPAR sub-chunk contains a list of parameters for texture modules when texture mapping is used.

```
typedef struct _TextureParameters
{
  FRACT Params[16];          /* Texture parameters */

} TPAR;
```

The SURF sub-chunk contains an array of five surface property specifications.

```
typedef struct _SurfaceProperties
{
  BYTE SProps[5];            /* Object properties */

} SURF;
```

The elements are defined as follows:

0	PRP_SURFACE	Surface type
1	PRP_BRUSH	Brush number
2	PRP_WRAP	IFF brush wrapping type
3	PRP_STENCIL	Stencil number for stencil objects
4	PRP_TEXTURE	Texture number if texture mapped

PRP_SURFACE is the type of surface. Values for this element include 0 (normal), 4 (genlock), and 5 (IFF brush).

PRP_BRUSH is the brush identification number if the brush is mapped to an IFF file.

PRP_WRAP is the IFF brush-wrapping type. Values for this element may be 0 (no wrapping), 1 (wrap X), 2 (wrap Z), or 3 (wrap X and Z).

PRP_STENCIL is the stencil number for stencil objects.

PRP_TEXTURE is the texture number if the object is texture-mapped.

The MTTR sub-chunk contains refraction data for transparent or glossy objects. Type may have a value of 0 for air (refraction index of 1.00), 1 for water (1.33), 2 for glass (1.67), 3 for crystal(2.00), or 4 for a custom index (1.00 to 3.55). Index contains the value of the custom refraction index.

```
typedef struct _RefractionData
{
    BYTE Type; /* Refraction type (0-4) */
    BYTE Index;        /* Custom index of refraction */
} MTTR;
```


Ex.1007 / Page 884 of 1158
APPLE INC.

The SPEC sub-chunk contains specularity information.

```
typedef struct _SpecularInfo
{
BYTE Specularity;      /* Specular reflection (0 to 255) */
BYTE Hardness; /* Specular exponent (0 to 31) */
} SPEC;
```

Specularity contains the amount of specular reflection in the range of 0 (none) to 255 (fully specular).

Hardness specifies the "tightness" of the specular spots. A value of zero gives broad specular spots and a value of 31 gives smaller spots.

The PRP0 sub-chunk contains an array of object properties that programs other than Turbo Silver might support.

```
typedef struct _MiscProperties
{
    BYTE Props[6];     /* Object properties */
} PRP0;
```

The elements are defined as follows:

0	PRP_BLEND	Blending factor
1	PRP_SMOOTH	Roughness factor
2	PRP_SHADE	Shading flag
3	PRP_PHONG	Phong shading flag
4	PRP_GLOSSY	Glossy flag
5	PRP_QUICK	Quickdraw flag

PRP_BLEND controls the amount of dithering used on the object in the range of 0 to 255 (255 being fully dithered).

PRP_SMOOTH specifies how rough the object should appear in the range of 0 (completely smooth) to 255 (maximal roughness).

PRP_SHADE indicates how the object is shaded. If the object is a light source and the flag is ON, then the object casts a shadow; otherwise it does not. If the object is a normal object and the flag is ON, the object is always considered fully lit and is not affected by other light sources.

PRP_PHONG indicates that phong shading is on by default. Any non-zero value turns it off.

PRP_GLOSSY indicates if an object is to be rendered as glossy or normal.

PRP_QUICK flag, when set, indicates that the image should not be drawn with all the points and edges, but instead should be drawn as a rectangular solid centered at the object position using sizes determined by the axis lengths.

The INTS sub-chunk is the intensity field for light source objects.

```
typedef struct _Intensity
{
  FRACT Intensity;      /* Light intensity */

} INTS;
```

An intensity of 255 for a sun-like light fully lights object surfaces which are perpendicular to the direction to the light source. For lamp-like light sources, the necessary intensity will depend on the distance to the light.

The STRY sub-chunk contains the story information for the description.

```
typedef struct _Story
{
  STORY CStory;

} STRY;
```

The EXTR chunk only contain sub-chunks which are required to appear.

EXTR Chunk

MTRX is used to represent a set of matrix coordinates.

```
typedef struct _Matrix
{
  VECTOR Translate;     /* Translation vector */
  VECTOR Scale;         /* X, Y, and Z scaling factors */
  MATRIX Rotate;        /* Rotation matrix */

} MTRX;
```

Translate is the translation vector in world coordinates.

Scale is the scaling factors with respect to local axes.

Rotate is the rotation matrix with respect to the world axes.

The LOAD sub-chunk contains the name of an external FORM object file, which may contain any number of objects, possibly grouped into hierarchies.

```
typedef struct _FileName
{
    BYTE FileName[80]; /* External object file name */
} LOAD;
```


Ex.1007 / Page 886 of 1158
APPLE INC.

For Further Information

For further information about TDDD, see the summary description included on the CD-ROM that accompanies this book. See also the article about TTDDD, an ASCII format based on TDDD, and the article on IFF, the format on which TDDD is based.

Ex.1007 / Page 887 of 1158
APPLE INC.

TGA

NAME:	TGA
ALSO KNOWN AS:	Truevision Graphics Adapter, Targa Graphics Adapter Image File, VST, VDA, ICB, TPIC
TYPE:	Bitmap
COLORS:	8-bit, 16-bit, 24-bit, 32-bit
COMPRESSION:	RLE, uncompressed
MAXIMUM IMAGE SIZE:	None
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	Truevision, Inc.
PLATFORM:	MS-DOS, Windows, UNIX, Atari, Amiga, others
SUPPORTING APPLICATIONS:	Too numerous to list
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes
IMAGES ON CD:	Yes
SEE ALSO:	Lumena Paint, RIX

USAGE: Used for the storage and interchange of deep-pixel images, paint, and image manipulation programs.

COMMENTS: A well-defined, well-documented format in wide use, which is quick and easy to read and decompress. It lacks, however, a superior compression scheme.

Overview

The TGA (Truevision Graphics Adapter) format is used widely in paint, graphics, and imaging applications that require the storage of image data containing up to 32 bits per pixel. TGA is associated with the Truevision product line of Targa, Vista, NuVista, and Targa 2000 graphics adapters for the PC and Macintosh, all of which can capture NTSC and/or PAL video image signals and store them in a digital frame buffer. For this reason, TGA has also become popular in the world of still-video editing.

Early work on the the TGA file format was performed at the EPICenter division of AT&T. EPICenter (Electronic Photography and Imaging Center), established

Ex.1007 / Page 888 of 1158
APPLE INC.

in 1984 to manufacture graphics boards, was purchased by EPICenter employees from AT&T in 1987 and renamed Truevision.

The first product produced by EPICenter was called the VDA (Video Display Adapter), which had a resolution of 256×200 and a 24-bit palette providing 16 milllion colors. At the time, it competed with the CGA from IBM. EPICenter's second product was the ICB (Image Capture Board), which launched both EPICenter and AT&T into the realm of video graphics (that is, video capture, manipulation, and output).

At this time, EPICenter purchased a paint package, written by Island Graphics, that later came to be known as TIPS (Truevision Image Paint System). TIPS gave VDA and ICB (and later Targa and TrueVista) users the ability to capture live video images, to create and overlay graphics, and to perform a variety of image-processing functions on bitmap data.

Although there was only one original TGA file format, applications using it created many different filename extensions—one for every graphics display board that EPICenter, and later Truevision, produced. Therefore, .VDA, .ICB, .TGA, and .VST image files created by Truevision applications all are actually in TGA format. Today, the only filename extensions supported are TGA and TPIC on the Macintosh and .TGA on the PC and other platforms.

In 1989, the TGA format was revised, and Truevision has chosen to designate the old and new formats as original TGA format and new TGA format, respectively. The original TGA format is very simple in design and quite easy to implement in code. This makes it an appealing format for developers to work with. As the available hardware technology has become more complex, however, additional file format features, such as the storage of gamma and color correction information and pixel aspect ratio data, have become necessary. The new TGA format was created as a wrapper around the original TGA format to add functionality without sacrificing backwards compatibility with older applications.

Today the TGA format is used on many different platforms world-wide for a variety of image storage, processing, and analysis needs. The Truevision solutions source book lists more than 200 software applications that support the TGA format.

The TGA format became popular primarily because it was the first 24-bit, truecolor bitmap format generally available to the PC community, even predating 24-bit support in TIFF. Truevision also gave developers access to the file format

Ex.1007 / Page 889 of 1158
APPLE INC.

specification and provided support for developers when necessary, including working code and sample images.

TGA is device-dependent in that the structure of the format is designed to fit the requirements of certain display hardware manufactured by Truevision. In practice, this is not a severe limitation, with one minor caveat: TGA does not support the storage of image data as planes of color information.

TGA comes in several flavors, the most common of which are usually referred to as the Targa 16, Targa 24, and Targa 32 formats. These designations identify the type of Truevision hardware that created the file, and the numbers indicate the depth (number of bits per pixel) of the image data the files contain. Less commonly found variants include VDA, ICB, and Targa M8.

All of the Truevision adapters were originally designed to interface with the ISA bus found in the IBM PC platform. For this reason, all data in TGA format files, including the image data, is stored in little-endian format. This includes TGA files created by the NuVista card residing in the otherwise big-endian Macintosh, and the PCI-bus versions of the Targa 2000 series of boards, for which both PC and Macintosh versions currently exist.

As with many popular and useful formats, TGA variants have come into being, designed by third parties to incorporate proprietary extensions. Versions of the popular freeware program Fractint at one time created unreadable TGA files. A campaign by one of the authors of this book to get the Stone Soup Group, the creators of Fractint, to change the filename suffix (from TGA to something else) seems to have been successful.

File Organization

The original TGA format (v1.0) is structured as follows:

- Header containing information on the image data and palette
- Optional image identification field
- Optional color map
- Bitmap data

The new TGA format (v2.0) contains all of the structures included in the original TGA format and also appends several data structures onto the end of the original TGA format. The following structures may follow the bitmap data:

Ex.1007 / Page 890 of 1158
APPLE INC.

- Optional developer directory, which may contain a variable number of tags pointing to pieces of information stored in the TGA file
- Optional developer area
- Optional extension area, which contains information typically found in the header of a bitmap file
- Optional color-correction table
- Optional postage-stamp image
- Optional scan-line table
- Footer, which points to the developer and extension areas and identifies the TGA file as a new TGA format file

As you can see, both the new and original TGA format files are identical in structure from the header to the image data area. For this reason, applications that read only original TGA format image files should have no problem reading new TGA format images. All information occurring after the image data may be ignored.

The TGA format specification available from Truevision is detailed and well-written. It is, in fact, one of the best written format specifications that was reviewed for this book, and we heartily congratulate Truevision on their effort. The TGA format is complex, but the clarity of the description in Truevision's specification makes it easy to read and understand. Truevision also distributes on floppy disk the Truevision TGA Utilities, which is a collection of utilities and C source code used to manipulate both TGA-format files and Targa video-graphics display adapters.

File Details

This section describes the various components of a TGA file in greater detail.

Header

The TGA header is eighteen bytes in length and is identical in both versions of the TGA file format. The structure of the TGA header is as follows:

```
typedef struct _TgaHeader
{
  BYTE IDLength;        /* 00h  Size of Image ID field */
  BYTE ColorMapType;    /* 01h  Color map type */
  BYTE ImageType;       /* 02h  Image type code */
  WORD CMapStart;       /* 03h  Color map origin */
  WORD CMapLength;      /* 05h  Color map length */
```


Ex.1007 / Page 891 of 1158
APPLE INC.

```
    BYTE CMapDepth;        /* 07h  Depth of color map entries */
    WORD XOffset;          /* 08h  X origin of image */
    WORD YOffset;          /* 0Ah  Y origin of image */
    WORD Width;            /* 0Ch  Width of image */
    WORD Height;           /* 0Eh  Height of image */
    BYTE PixelDepth;       /* 10h  Image pixel size */
    BYTE ImageDescriptor; /* 11h  Image descriptor byte */

} TGAHEAD;
```

IDLength is the number of significant bytes in the image identification field starting at byte 12h (following the header) and may be in the range of 0 to 255. The IDLength set to 0 indicates that there is no image identification field in the TGA file.

ColorMapType indicates whether the TGA file includes a palette. A value of 1 indicates the presence of a palette, while a value of 0 indicates that no palette is included. If the value of this field is not 0 or 1, then it is probably a value specific to the program or developer that created the TGA file. Truevision reserves the ColorMapType values 0 to 127 for its own use and allots values 128 to 255 for use by developers.

ImageType indicates the type of image stored in the TGA file. There are currently seven TGA image types. Colormapped images (pseudocolor) use a palette. Truecolor images do not use a palette and store their pixel data directly in the image data, although truecolor TGA image files may contain a palette that is used to store the color information from a paint program. Truevision reserves the ImageType values 0 to 127 for its own use and allots values 128 to 255 for use by developers.

Valid ImageType values are listed below:

ImageType	Image Data Type	Colormap	Encoding
0	No image data included in file	No	No
1	Colormapped image data	Yes	No
2	Truecolor image data	No	No
3	Monochrome image data	No	No
9	Colormapped image data	Yes	Yes
10	Truecolor image data	No	Yes
11	Monochrome image data	No	Yes

The next three fields are known collectively as the Color Map Specification; the information contained in these fields is used to manipulate the image

Ex.1007 / Page 892 of 1158
APPLE INC.

palette. If the ColorMapType field value is zero, then all three of these fields have a value of zero.

CMapStart defines the offset of the first entry in the palette. Although all palette entries must be contiguous, the entries may start anywhere in the palette; for example, 16-color values may be stored in a 64-element palette starting at entry 31 rather than at entry 0.

CMapLength defines the number of elements in the colormap. If an image contains only 57 colors, then it is possible to construct a 57-element palette using this field.

CMapDepth contains the number of bits in each palette entry. The value is typically 15, 16, 24, or 32 and need not be the same value as the image data pixel depth. Table TGA-1 shows valid entries for different types of TGA palettes.

TABLE TGA-1: *TGA Palette Entry Sizes*

Truevision Bits Per Display Adapter	Attribute Colormap Entry	Bits Per Pixel	Color Formats Supported
Targa M8	24	0	Pseudo
Targa 16	15	1	True
Targa 24	24	0	True
Targa 32	32	8	True
ICB	0	0	True
VDA	16	0	Pseudo
VDA/D	16	0	Pseudo
Vista	24 or 32	0 or 8	True, Pseudo, Direct

The next six fields in the header (the last 10 bytes) are referred to collectively as the image specification. The data in these fields is used to describe the image data found in the TGA file.

XOffset and YOffset describe the position of the image on the display screen. Normally, the coordinate 0,0 defaults to the lower-left corner of the screen, but any of the four corners may be designated the origin point by the ImageDescriptor field (the last field in the header).

Width and Height are the size of the image in pixels. The maximum size of a TGA image is 512 pixels wide by 482 pixels high.

Ex.1007 / Page 893 of 1158
APPLE INC.

PixelDepth is the number of bits per pixel, including attribute bits, if any. Typical PixelDepth values are 8, 16, 24, and 32, although other depth values may be specified, as shown in Table TGA-1.

ImageDescriptor contains two pieces of information. Bits 0 through 3 contain the number of attribute bits per pixel. Attribute bits are found only in pixels for the 16- and 32-bit flavors of the TGA format and are called alpha channel, overlay, or interrupt bits. Bits 4 and 5 contain the image origin location (coordinate 0,0) of the image. This position may be any of the four corners of the display screen. When both of these bits are set to zero, the image origin is the lower-left corner of the screen. Bits 6 and 7 of the ImageDescriptor field are unused and should be set to 0.

Image ID

The Image ID (image description) field is an optional field that may appear immediately after the TGA header. The Image ID field stores information that identifies the image in some way (filename, author name, serial number, and so on). This field is not required to be NULL-terminated, although it should be if it is used to store string data. The size of this field is indicated by the value of the IDLength field in the header. This value may be in the range 0–255. A value of 0 indicates that no Image ID field is present in the TGA file.

Colormap

The TGA format defines three methods of arranging image data: psuedocolor, direct-color, and truecolor.

Pseudocolor images store an index value into a palette in each pixel value of data. It is the palette that contains the actual pixel values that are displayed. Pseudocolor image palettes store each pixel value as a single element in the palette. The color channels of each pixel value are not accessible individually.

Direct-color images are similar to pseudocolor images, except that each color channel (red, green, and blue) is stored in separate elements and may be individually altered. Each pixel value of direct-color image data contains three index values, one for each color channel in the colormap.

Truecolor images store the pixel color information directly in the image data and do not use a palette.

Ex.1007 / Page 894 of 1158
APPLE INC.

The presence of a palette and the format of the image data found in a TGA file is determined by the type of Truevision hardware that was used to create the image data (see Table TGA-1). TGA images created with a Targa 24 are only truecolor images and therefore never use a palette. TGA images created with a VDA/D card are only in the pseudocolor format and will therefore always use a palette. The Vista series of cards (ATVista and NuVista) may create and store TGA data in any of the three color formats. A palette is present in a TGA file if the ColorMapType field is set to 1. A value of 0 indicates that no palette is present in the TGA file.

It is important to realize that a palette may be present in a TGA image file even if it is not used by the image data. All TGA image files created by the TIPS paint program contain a palette that stores the 256 colors found in the TIPS color palette. This palette is not actually used to display the image data but is instead used by TIPS. A TGA reader should therefore never assume that truecolor TGA files never contain a palette.

The TGA format supports variable-size palettes. Most other formats require a palette to have a fixed number of entries based on the pixel depth of the image data. Thus, an 8-bit image contains a 256-element colormap even if only four colors are needed to reproduce the image. The TGA format, however, does not determine the number of colormap elements based on the pixel depth, so an image with 57 colors may only have a 57-element palette. The number of elements in the palette is contained in the CMapLength field in the header.

The size of each palette element in bits is found in the CMapDepth field of the header. The depth of a pixel and the depth of a palette element are not always the same. A 24-bit image may contain a 256-element palette, with each element having a depth of 24 bits, but it may contain pixel data with only an 8-bit depth. This is because 8 bits is all that is required to index a 256-element palette. It is also possible for a TGA image to contain a 4096-element palette where each element is eight bits in depth. Each pixel value of the image data therefore needs to have a minimum depth of 12 bits for indexing into the palette, although 16-bit pixel values are easier to read and write. The depth of a palette element always includes alpha channel, overlay, or interrupt bit information, if any.

Ex.1007 / Page 895 of 1158
APPLE INC.

Image Data Encoding

Image data stored in a TGA file is normally raw (unencoded). For this reason, TGA files tend to be quite large, especially when the bitmap data is 24 or 32 bits deep. To address this problem, the TGA specification incorporates a simple, but effective, RLE compression scheme. For more detailed information on run-length encoding, see Chapter 9, *Data Compression.*

The RLE encoding method used by the TGA format encodes runs of identical pixels rather than runs of identical bits or bytes. This achieves a higher compression ratio over a bit-wise or byte-wise RLE scheme, because TGA pixel data often occurs as multiple-byte values rather than single-byte values. Therefore, contiguous runs of identical bytes in TGA image data often occur only in very short lengths.

Data encoded using the TGA RLE scheme may contain two types of encoded data packets. The first type is a run-length packet that is used to encode multiple runs of the same pixel value into a single data packet. A run-length packet begins with a single byte used as the pixel count. The value for the lower seven bits of this byte is in the range 0 to 127, and the count is always one plus this value (1 to 128). There can never be an encoded run length of zero pixels. The high bit of the pixel count value is always set to 1 to indicate that this is a run-length encoded packet. Following the pixel count is the pixel data value. This value is the number of bits equal to the Pixel Depth value in the Image Specification section of the header. Because the size of TGA pixels can range from one to four bytes in size, this value varies from between two to five bytes in length, depending upon the type of TGA image data encoded.

The second type of packet is the raw or non-run-length encoded packet. When a run of pixel values is too short in length to justify using the run-length packet format, the run is encoded using the raw packet format. Raw packets start with a byte that is used as the pixel count. Just as with the encoded packet, the count value is in the range of 0 to 127, with the actual pixel count being one plus this value (1 to 128). A run length of zero pixels can never be encoded. Raw packets, however, have the high bit of the count byte always set to zero. This differentiates raw packets from encoded packets, in which the high byte is always one. Following the count byte is the number of pixels equal to the count. The number of bytes that follows the raw count is equal to the count value multiplied by the number of bytes per pixel.

Ex.1007 / Page 896 of 1158
APPLE INC.

The following TGA RLE pseudocode algorithm is used to encode a pixel run using an encoded packet:

```
Set counter to zero
Read a pixel of scan-line data
Read a second pixel of scan-line data
If the first pixel is the same as the second pixel
   increment the counter
Else
   write the counter value (with high bit ON)
   write the pixel value
```

The following TGA RLE pseudocode algorithm is used to encode a pixel run using a raw packet:

```
Set counter to zero
Read a certain number of pixels of scan-line data
Increment the counter for each pixel read
Write the counter value (with high bit OFF)
Write all pixel values read
```

Figure TGA-1 shows the RLE packet types for various pixel sizes.

Image Data

Image data is usually found following the header, but may occur after a palette or Image ID field if these are present in the TGA file. For this reason, never read image data from a TGA file without first checking for the presence of palette and Image ID fields. If you do, you will quickly notice that the displayed image is skewed, because the image was read starting at the wrong offset. (See the section called "Colormap" above.)

The size of a TGA image is limited to 65,535 pixels high by 65,535 pixels wide. This is because a 16-bit field is used to store the size of the image in the header. Otherwise, the size of a TGA image would be unlimited. A typical size for Targa 16, 24, and 32 images is 512×482 pixels; the NuVista is 640×480 pixels; and the ATVista is 756×486 pixels.

Figure TGA-2 shows different pixel data formats.

Most of the Truevision display adapters store pixel data in 8-, 16-, 24-, or 32-bit increments. Reading or writing pixel information for these formats is as simple as reading and writing bytes of data. Targa 16-bit pixel data, however, is slightly more complicated, as described below.

Ex.1007 / Page 897 of 1158
APPLE INC.

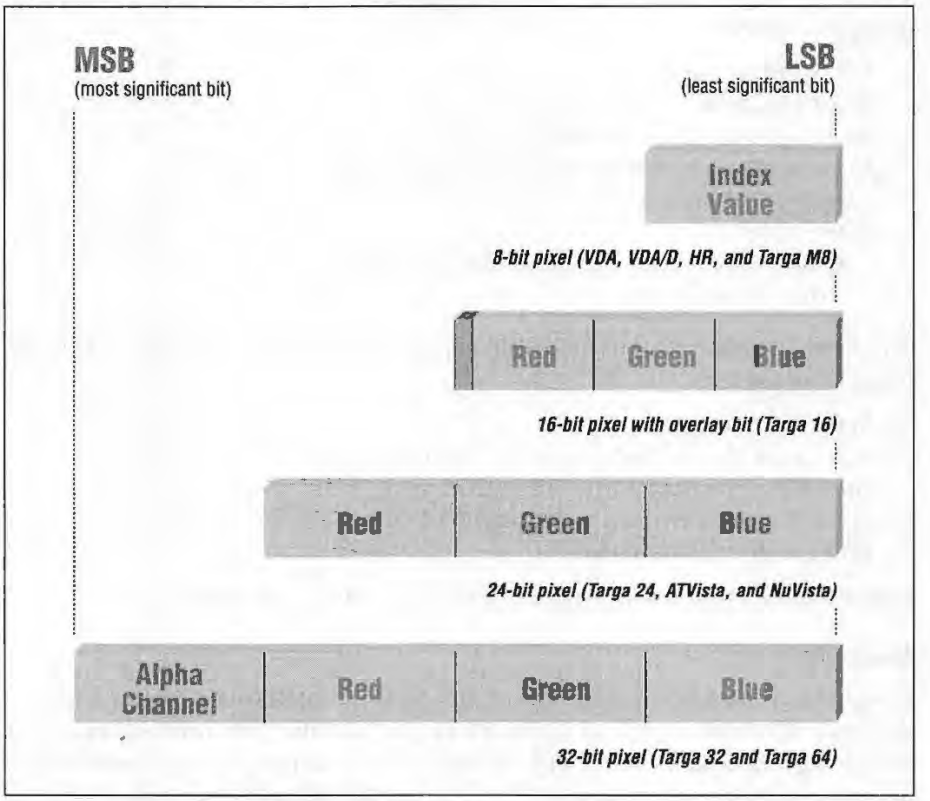


FIGURE TGA-1: *Run-length encoding packet types*

When a value of 15 appears in the PixelDepth field of the Image Specification section of the header, there are five bits each of red, green, and blue pixel data and one bit of overlay data in each pixel (see the section below called "Pixel Attribute Bits"). This is the format the Targa 16 uses to store data. Because these 16 bits are stored in two bytes of data, a little shifting and masking is required to read and write these pixel data values.

In the following example, a scan line of unencoded pixel data is stored in the byte array pixeldata[]. The red, green, and blue values are five bits in size, and the overlay attribute is a single bit in size. Data from the first pixel (array elements 0 and 1) are read and stored in the variables defined:

Ex.1007 / Page 898 of 1158
APPLE INC.

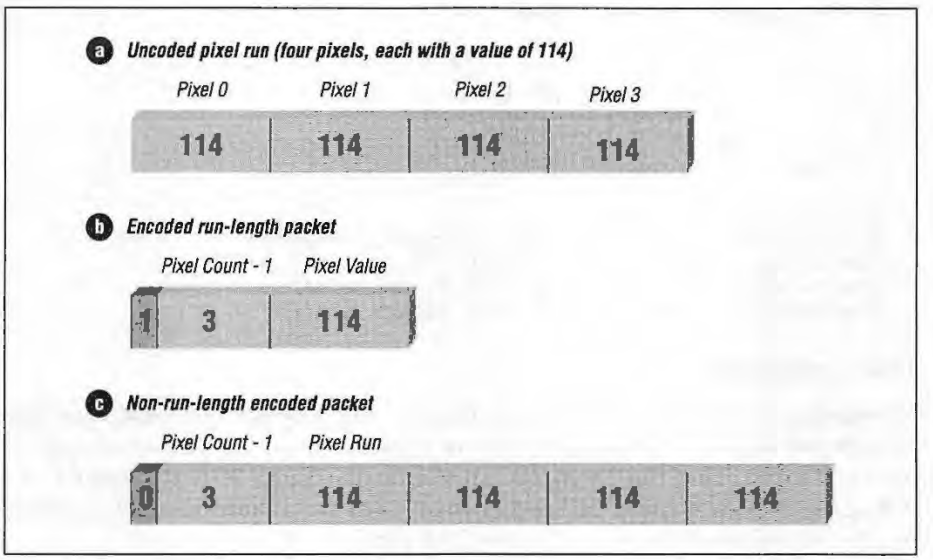


FIGURE TGA-2: *Pixel data formats*

```
/*
** Reading and writing 16-bit pixel data stored
** in an 8-bit BYTE array. Note that the green
** value is split between two bytes.
*/
BYTE red, green, blue, overlay;
BYTE pixeldata[SCAN_LINE_LENGTH];
/* Read */
red   =  (pixeldata[0] & 0xf8) >> 3;
green = ((pixeldata[0] & 0x07) << 2) | ((pixeldata[1] & 0xfb) >> 6);
blue  =  (pixeldata[1] & 0x3e) >> 1;
overlay =  pixeldata[1] & 0x01;

/* Write */
pixeldata[0] =  (red  << 3)          | (pixeldata[0] & 0x07);
pixeldata[0] = ((green & 0x1c) >> 2) | (pixeldata[0] & 0xf8);
pixeldata[1] = ((green & 0x03) << 6) | (pixeldata[1] & 0xf8);
pixeldata[1] =  (blue << 1)          | (pixeldata[1] & 0xc1);
pixeldata[1] =   overlay             | (pixeldata[1] & 0xfe);

/*
** Reading and writing 16-bit pixel data stored
** in a 16-bit WORD array
*/
```


Ex.1007 / Page 899 of 1158
APPLE INC.

```
BYTE red, green, blue, overlay;
WORD pixeldata[SCAN_LINE_LENGTH];

/* Read */
red      = (pixeldata[0] & 0xfc00) >> 11;
green    = (pixeldata[0] & 0x07e0) >> 6;
blue     = (pixeldata[0] & 0x003f) >> 1;
overlay  =  pixeldata[0] & 0x0001;

/* Write */
pixeldata[0] = (red   << 11) | (pixeldata[0] & 0x03ff);
pixeldata[0] = (green << 6)  | (pixeldata[0] & 0xfc1f);
pixeldata[0] = (blue  << 1)  | (pixeldata[0] & 0xffc1);
pixeldata[0] =  overlay      | (pixeldata[0] & 0xfffe);
```

Pixel Attribute Bits

The names of the Targa display adapters include a designation that indicates the number of bits per pixel that they are capable of storing. This seems logical for the Targa 16 and the Targa 24, but not for the Targa 32 and Targa 64. It's difficult to believe 32 and 64 bits per pixel until you realize that color data is not the only information you can store in a pixel.

The number of bits a pixel may contain that are not directly associated with the color value of the pixel are stored in bits 0 through 3 of the ImageDescriptor field of the header (attribute bits per pixel). In the case of the Targa 16, only 15 of the 16 pixel bits are used for color information. The sixteenth bit, also called the overlay bit, is used to indicate whether the pixel is transparent (invisible) or opaque (visible) when displayed on a video monitor. The ICB board also uses 15 bits per pixel for color information and a single bit for overlay control. The VDA/D board is similar in that it uses five bits per primary color and uses the sixteenth bit for interrupt control. The Targa 32 and the TrueVista boards (ATVista and NuVista) each use 32 bits per pixel. Color information is stored in 24 bits, and the additional eight attribute bits in each pixel are used as an alpha channel value.

Alpha channel is a nondescript name that indicates the degree of transparency of a displayed pixel. Alpha channel and overlay values are used when one image is overlaid onto another image or onto a live video picture. A single overlay bit (as in the Targa 16) can only indicate that the pixel is visible or invisible. Eight bits of precision can vary the visibility of a pixel from completely transparent (0) to completely opaque (255).

Ex.1007 / Page 900 of 1158
APPLE INC.

The alpha channel value also describes the degree to which a pixel from an image is mixed with a live video source. An alpha value of 0 displays the pixel entirely from the graphics image stored in the frame buffer memory. An alpha value of 255 displays the pixel entirely from the live video source. An alpha value of 84 displays the pixel as 33 percent graphics image (85 of 256) and 67 percent live video (171 of 256). A pixel with an alpha value of 84 appears as a translucent graphics image overlaid on a field of live video.

The ability to control graphical and live video images using 256 alpha channel levels allows the superimposing of graphical text over video, fading into and out of an image, and cross-fading between graphical images and live video. All of these effects can be rendered using the Truevision Targa+ and NuVista+ graphical display boards.

When storing pixel data or pixel size, the attribute bits are always included in any reads, writes, or calculations. Attribute bits are also stored in colormaps and lookup tables, although in these cases their values are usually set to zero and ignored.

The New TGA Format

Version 2.0 adds several features to the original TGA format, which increases the amount of information the TGA format can support. The original TGA format is fairly simple in design, which allows it to be quickly and easily implemented in software. However, it does not contain many features needed by developers—features that are found in several other image file formats. Therefore, extensions have been added to the TGA format to allow the storage of additional image information and to create a customizable area for the storage of developer-specific information.

The extensions added by the new TGA format are called the Developer Area, the Extension Area, and the TGA File Footer. These areas were appended to the original TGA format without making any changes to the TGA file header. Applications that read only original TGA image files should have no problems reading new TGA format files unless the Developer or Extension Areas contain information necessary to read or display the image data properly. For newer applications that support the new TGA format, it is important to correctly interpret as much of the Extension and Developer Areas information as possible.

Ex.1007 / Page 901 of 1158
APPLE INC.

Footer

The footer is 26 bytes in length and is always the last piece of information found in a v2.0 TGA format file. It contains a total of three fields that are represented in the following structure:

```
typedef struct _TgaFooter
{
  DWORD ExtensionOffset;        /* Extension Area Offse */
  DWORD DeveloperOffset;        /* Developer Directory Offset */
  CHAR Signature[18];           /* TGA Signature */

} TGAFOOT;
```

ExtensionOffset is a 4-byte offset value of the extension area of the TGA file. If this value is zero, then there is no extension area present in the TGA file.

DeveloperOffset is the number of bytes from the beginning of the file (byte 0) to the first byte of the developer directory. If this value is zero, then the TGA file does not contain a Developer Area.

Signature contains an identifying signature string. The TGA format does not contain a field in the header indicating the version of the format. Instead, v2.0 includes a footer containing a 16-byte character string that identifies the version of the file.

To determine the version of a TGA file, read the file footer and check bytes 8 through 23 of the footer for the presence of the character string TRUEVISION-XFILE. If this signature string is present, the TGA file is v2.0 and thus may contain a Developer and Extension Areas.

Following the signature is the ASCII value 2Eh (period) and the ASCII value 00h (NULL). All of these fields must contain the correct information for the file to be recognized as a v2.0 TGA format file.

A v1.0 TGA file may be converted to a v2.0 TGA file simply by appending a footer with the appropriate signature string, a period, and NULL characters. In this case, you could set the ExtensionOffset and DeveloperOffset to 0. Be cautious about doing this, however. This conversion adds nothing to the functionality of the file unless an Extension or Developer Area is added. It will not, however, interfere with the ability of older, pre-v2.0 TGA format software to read the file.

Ex.1007 / Page 902 of 1158
APPLE INC.

Developer Area

The Developer Area begins with a directory that resembles the structure of the Image File Directory found in the TIFF format. The first entry in the directory is the number of directory entries, or tags, that the directory contains. This field is two bytes in size. The offset value of this field is stored in the footer described in the previous section.

Following the number of directory entries is a series of 10-byte tags, one for each entry specified. The structure of a Developer Area tag is as follows:

```
typedef struct _TgaTag
{
  WORD  TagNumber;       /* ID Number of the tag */
  DWORD DataOffset;      /* Offset location of the tag data */
  DWORD DataSize;        /* Size of the tag data in bytes */

} TGATAG;
```

TagNumber is the identification number of the tag. Tag number values from 0 to 32767 are reserved for developer use, while tag number values 32768 to 65535 are reserved for use by Truevision only. Tags may be registered with the Truevision Developer Services to assure permanent and exclusive use by your application.

DataOffset contains the offset location of the data in the TGA file. Note that offsets are always calculated from the beginning of the file.

DataSize is the size of the data in bytes.

The tags in the developer directory are always stored as a contiguous block, and the tags do not have to be sorted by tag number. Tags do not indicate the type of data pointed to by the tag (BYTE-, WORD-, or DWORD-oriented data), so an application that is reading tag data is required to have prior knowledge of the type of data the tag points to.

Extension Area

The Extension Area can be thought of as a second header that contains information not found in the original TGA format header. The offset of the Extension Area is stored in the TGA footer. The size of the Extension Area in v2.0 is 495 bytes. The structure of the Extension Area is as follows:

```
typedef struct _TgaExtension
{
  WORD  Size;                    /* Extension Size */
  CHAR  AuthorName[41];          /* Author Name */
  CHAR  AuthorComment[324];      /* Author Comment */
```


Ex.1007 / Page 903 of 1158
APPLE INC.

```
    WORD  StampMonth;          /* Date/Time Stamp: Month */
    WORD  StampDay;            /* Date/Time Stamp: Day */
    WORD  StampYear;           /* Date/Time Stamp: Year */
    WORD  StampHour;           /* Date/Time Stamp: Hour */
    WORD  StampMinute;         /* Date/Time Stamp: Minute */
    WORD  StampSecond;         /* Date/Time Stamp: Second */
    CHAR  JobName[41];         /* Job Name/ID */
    WORD  JobHour;             /* Job Time: Hours */
    WORD  JobMinute;           /* Job Time: Minutes */
    WORD  JobSecond;           /* Job Time: Seconds */
    CHAR  SoftwareId[41];      /* Software ID */
    WORD  VersionNumber;       /* Software Version Number */
    BYTE  VersionLetter;       /* Software Version Letter */
    DWORD KeyColor;            /* Key Color */
    WORD  PixelNumerator;      /* Pixel Aspect Ratio */
    WORD  PixelDenominator;    /* Pixel Aspect Ratio */
    WORD  GammaNumerator;      /* Gamma Value */
    WORD  GammaDenominator;    /* Gamma Value */
    DWORD ColorOffset;         /* Color Correction Offset */
    DWORD StampOffset;         /* Postage Stamp Offset */
    DWORD ScanOffset;          /* Scan-Line Table Offset */
    BYTE  AttributesType;      /* Attributes Types */

} TGAEXTEN;
```

Size specifies the number of bytes in the extension area. This value is 495 for v2.0 of the TGA format.

AuthorName allows the name of the TGA file creator to be stored using up to 40 characters. Unused characters are padded with spaces and the field is NULL-terminated.

AuthorComment is also a string field containing 324 bytes (four 80-character lines) in which to store information. This field is similar to the Image ID field that follows the header. This field is also padded with spaces and is NULL-terminated.

The six Stamp fields contain the time and date the image was created or last modified. These fields may have the following values:

StampMonth	1 to 12
StampDay	1 to 31
StampYear	0000 to 9999
StampHour	0 to 23
StampMinute	0 to 59
StampSecond	0 to 59

Ex.1007 / Page 904 of 1158
APPLE INC.

Unused fields are set to zero.

JobName is a 4-byte, NULL-terminated string identifying the production job with which the image is associated. JobHour, JobMinute, and JobSecond indicate the amount of time expended on the job.

The SoftwareId field is a 40-character, NULL-terminated string that identifies the software application that created the file. The VersionNumber and VersionLetter fields contain the version of the software application.

KeyColor contains the background color of the image. This is the pixel color used to paint the areas of the display screen not covered by the image or the color used to clear the screen if the image is erased. The default value for this field is 0, which corresponds to black.

PixelNumerator and PixelDenominator store the aspect ratio of the pixels used in the image. If no aspect ratio is specified, then these fields are set to 0.

GammaNumerator and GammaDenominator are the gamma correction values to be used when displaying the image. If both fields are 0, then a gamma value is not used.

ColorOffset contains an offset value of the color correction table. If the file does not contain a color correction table, then the value of this field is set to 0.

The StampOffset field contains an offset value of the postage-stamp image included in the TGA file. If no postage-stamp image is present, the value of this field is 0.

ScanOffset contains the offset value of the scan-line offset table.

AttributesType describes the type of alpha channel data contained within the pixel data. A value of 00h indicates that the image data contains no alpha channel value. A value of 01h, 02, 04h, or 08h indicates the presence of alpha channel data. (See the TGA specification for more information on this field.)

Scan-line table, postage-stamp image, and color correction table

A new TGA format file may contain three additional data structures not found in the original TGA format. These structures are the scan-line table, the postage-stamp image, and the color correction table. There may only be one of each of these data structures per TGA file, and offsets to these structures appear in the Extension Area.

The scan-line table is a method of accessing scan lines at any location within raw or compressed image data. The table is an array of DWORD values. Each value is the offset location from the beginning of the file to the beginning of

Ex.1007 / Page 905 of 1158
APPLE INC.

the corresponding scan line in the image data. There is one entry in the scan-line table per scan line in the image. Entries are written to the table in the order in which the scan lines appear within the image.

The postage-stamp image is a smaller rendering of the primary image stored within the TGA file. The first byte of the postage-stamp data specifies the width of the stamp in pixels, and the second byte specifies the height, also in pixels. Postage stamps should not be larger than 64×64 pixels, are typically stored in the same format as the primary image, and are never compressed.

The color correction table is an array 2048 bytes in length, which contains 256 entries used to store values used for color remapping. The entire table has the following format:

```
typedef struct _TGAColorCorrectionTable
{
  SHORT Alpha;
  SHORT Red;
  SHORT Green;
  SHORT Blue;

} TGACCT[256];
```

The fields Alpha, Red, Green, and Blue store the color values for each entry. The range of each value is 0 to 65535. Black is 0,0,0,0, and white is 65535,65535,65535,65535.

For Further Information

For further information about the TGA format, see the specification and code example included on the CD-ROM that accompanies this book.

The TGA format is maintained by Truevision, Inc. Prior to this writing, Truevision was acquired by RasterOps, Inc., but for the time being Truevision remains an independent subsidiary. Copies of the latest TGA specification, including a sample code disk, may be obtained directly from Truevision:

Truevision Incorporated
7340 Shadeland Station
Indianapolis, IN 46256-3925
Voice: 317-841-0332
FAX: 317-576-7700
BBS: 317-577-8783
FTP: *ftp://ftp.truevision.com/*
WWW: *http://www.truevision.com/Truevision.html*

Ex.1007 / Page 906 of 1158
APPLE INC.

Also on their Internet sites, you can get the TGA v2.0 file format specifications, as well as tools in C that read and write the format, and sample images.

Ex.1007 / Page 907 of 1158
APPLE INC.

TIFF

NAME:	TIFF
ALSO KNOWN AS:	Tag Image File Format
TYPE:	Bitmap
COLORS:	1- to 24-bit
COMPRESSION:	Uncompressed, RLE, LZW, CCITT Group 3 and Group 4, JPEG
MAXIMUM IMAGE SIZE:	$2^{32}-1$
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	See article for discussion
ORIGINATOR:	Aldus
PLATFORM:	MS-DOS, Macintosh, UNIX, others
SUPPORTING APPLICATIONS:	Most paint, imaging, and desktop publishing programs
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes (in libtiff package)
IMAGES ON CD:	Yes
SEE ALSO:	Chapter 9, *Data Compression* (RLE, LZW, CCITT, and JPEG)

USAGE: Used for data storage and interchange. The general nature of TIFF allows it to be used in any operating environment, and it is found on most platforms requiring image data storage.

COMMENTS: The TIFF format is perhaps the most versatile and diverse bitmap format in existence. Its extensible nature and support for numerous data compression schemes allow developers to customize the TIFF format to fit any peculiar data storage needs.

Overview

The TIFF specification was originally released in 1986 by Aldus Corporation as a standard method of storing black-and-white images created by scanners and desktop publishing applications. This first public release of TIFF was the third major revision of the TIFF format, and although it was not assigned a specific version number, this release may be thought of as TIFF Revision 3.0. The first widely used revision of TIFF, 4.0, was released in April 1987. TIFF 4.0 added

Ex.1007 / Page 908 of 1158
APPLE INC.

support for uncompressed RGB color images and was quickly followed by the release of TIFF Revision 5.0 in August 1988. TIFF 5.0 was the first revision to add the capability of storing palette color images and support for the LZW compression algorithm. (See the sidebar on LZW compression in the section called "Compression" later in this article.) TIFF 6.0 was released in June 1992 and added support for CMYK and YCbCr color images and the JPEG compression method. (See the section called "Color" in Chapter 2, *Computer Graphics Basics*, for a discussion of these color images. See Chapter 9, for a discussion of JPEG compression.)

Today, TIFF is a standard file format found in most paint, imaging, and desktop publishing programs and is a format native to the Microsoft Windows GUI. TIFF's extensible nature, allowing storage of multiple bitmap images of any pixel depth, makes it ideal for most image storage needs.

The majority of the description in this chapter covers the current TIFF revision 6.0. Because each successive TIFF revision is built upon the previous revision, most of the information present in this chapter also pertains to TIFF Revision 5.0 as well. And, although more images are currently stored in the TIFF 5.0 format than in any other revision of TIFF, quite a few TIFF 4.0 image files are still in existence. For this reason, information is also included that details the differences between the TIFF 4.0, 5.0, and 6.0 revisions.

TIFF has garnered a reputation for power and flexibility, but it is considered complicated and mysterious as well. In its design, TIFF attempts to be very extensible and provide many features that a programmer might need in a file format. Because TIFF is so extensible and has many capabilities beyond all other image file formats, this format is probably the most confusing format to understand and use.

A common misconception about TIFF is that TIFF files are not very portable between software applications. This is amazing considering that TIFF is widely used as an image data interchange format. Complaints include, "I've downloaded a number of TIFF clip art packages from some BBSs and my paint program or word processor is able to display only some of the TIFF image files, but not all of them," "When I try to display certain TIFF files using my favorite image display program, I get the error message 'Unknown Tag Type' or 'Unsupported Compression Type'," and "I have a TIFF file created by one application and a second application on the same machine cannot read or display the image, even though TIFF files created by the second application can be read and displayed by the first application."

Ex.1007 / Page 909 of 1158
APPLE INC.

These complaints are almost always immediately blamed on the TIFF image files themselves. The files are labeled "bad," because they have been munged during a data file transfer or were exported by software applications that did not know how to properly write a TIFF file. In reality, most TIFF files that do not import or display properly are not bad, and the fault usually lies, instead, with the program that is reading the TIFF file.

If an application only uses black-and-white images, it certainly does not need to support the reading and writing of color and gray-scale TIFF image files. In this case, the application should simply, and politely, refuse to read non-black-and-white TIFF image files and tell you the reason why. By doing this, the application would prevent the user from trying to read unusable image data and would also cut down on the amount of TIFF code the application programmers need to write.

Some applications that read TIFF image files—or any type of image files, for that matter—may just return an ambiguous error code indicating that the file could not be read, leaving the user with the impression that the TIFF file itself is bad (not that the application could not use the image data the TIFF file contained). Such an occurrence is the fault of the application designer in not providing a clearer message informing the user what has happened.

Sometimes, however, you may have an application that should be able to read a TIFF file, and it does not, even though the type of image data contained in the TIFF file is supported by the application. There are numerous reasons why a perfectly good TIFF file cannot be read by an application, and most of them have to do with the application programmer's lack of understanding of the TIFF format itself.

A major source of TIFF reader problems is the inability to read data regardless of byte-ordering scheme. The bytes in a 16-bit and 32-bit word of data are stored in a different order on little-endian architectures (such as the Intel iAPX86), than on big-endian machines (such as the Motorola MC68000A). Reading big-endian data using the little-endian format results in little more than garbage.

Another major source of problems is readers that do not support the encoding algorithm used to compress the image data. Most readers support both raw (uncompressed) and RLE-encoded data but do not support CCITT T.4 and T.6 compression. It is also surprising how many TIFF readers support the reading of color TIFF files, which are either stored as raw or RLE-compressed data, but do not support the decompression of LZW-encoded data.

Ex.1007 / Page 910 of 1158
APPLE INC.

Most other TIFF reader problems are quite minor, but usually fatal. Such problems include failure to correctly interpret tag data, no support for color-mapped images, or the inability to read a bitmap scan line that contains an odd number of bytes.

File Organization

TIFF files are organized into three sections: the Image File Header (IFH), the Image File Directory (IFD), and the bitmap data. Of these three sections, only the IFH and IFD are required. It is therefore quite possible to have a TIFF file that contains no bitmapped data at all, although such a file would be highly unusual. A TIFF file that contains multiple images has one IFD and one bitmap per image stored.

TIFF has a reputation for being a complicated format in part because the location of each Image File Directory and the data the IFD points to—including the bitmapped data—may vary. In fact, the only part of a TIFF file that has a fixed location is the Image File Header, which is always the first eight bytes of every TIFF file. All other data in a TIFF file is found by using information found in the IFD. Each IFD and its associated bitmap are known as a TIFF *subfile.* There is no limit to the number of subfiles a TIFF image file may contain.

Each IFD contains one or more data structures called *tags.* Each tag is a 12-byte record that contains a specific piece of information about the bitmapped data. A tag may contain any type of data, and the TIFF specification defines over 70 tags that are used to represent specific information. Tags are always found in contiguous groups within each IFD.

Tags that are defined by the TIFF specification are called *public tags* and may not be modified outside of the parameters given in the latest TIFF specification. User-definable tags, called *private tags,* are assigned for proprietary use by software developers through the Aldus Developer's Desk. See the TIFF 6.0 specification for more information on private tags.

Note that the TIFF 6.0 specification has replaced the term *tag* with the term *field.* Field now refers to the entire 12-byte data record, while the term tag has been redefined to refer only to a field's identifying number. Because so many programmers are familiar with the older definition of the term tag, the authors have choosen to continue using tag, rather than field, in this description of TIFF to avoid confusion.

Figure TIFF-1 shows three possible arrangements of the internal data structure of a TIFF file containing three images. In each example, the IFH appears first

Ex.1007 / Page 911 of 1158
APPLE INC.

in the TIFF file. In the first example, each of the IFDs has been written to the file first and the bitmaps last. This arrangement is the most efficient for reading IFD data quickly. In the second example, each IFD is written, followed by its bitmapped data. This is perhaps the most common internal format of a multi-image TIFF file. In the last example, we see that the bitmapped data has been written first, followed by the IFDs. This seemingly unusual arrangement might occur if the bitmapped data is available to be written before the information that appears in the IFDs.

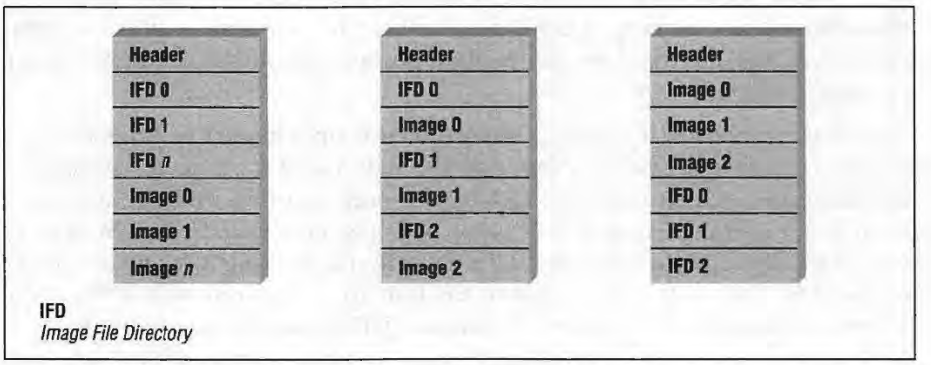


FIGURE TIFF-1: *Three possible physical arrangements of data in a TIFF file*

Each IFD is a road map of where all the data associated with a bitmap can be found within a TIFF file. The data is found by reading it directly from within the IFD data structure or by retrieving it from an offset location whose value is stored in the IFD. Because TIFF's internal components are linked together by offset values rather than by fixed positions, as with stream-oriented image file formats, programs that read and write TIFF files are often very complex, thus giving TIFF its reputation.

The offset values used in a TIFF file are found in three locations. The first offset value is found in the last four bytes of the header and indicates the position of the first IFD. The last four bytes of each IFD is an offset value to the next IFD. And the last four bytes of each tag may contain an offset value to the data it represents, or possibly the data itself.

NOTE

Offsets are always interpreted as a number of bytes from the beginning of the TIFF file.

Ex.1007 / Page 912 of 1158
APPLE INC.

Figure TIFF-2 shows the way data structures of a TIFF file are linked together.

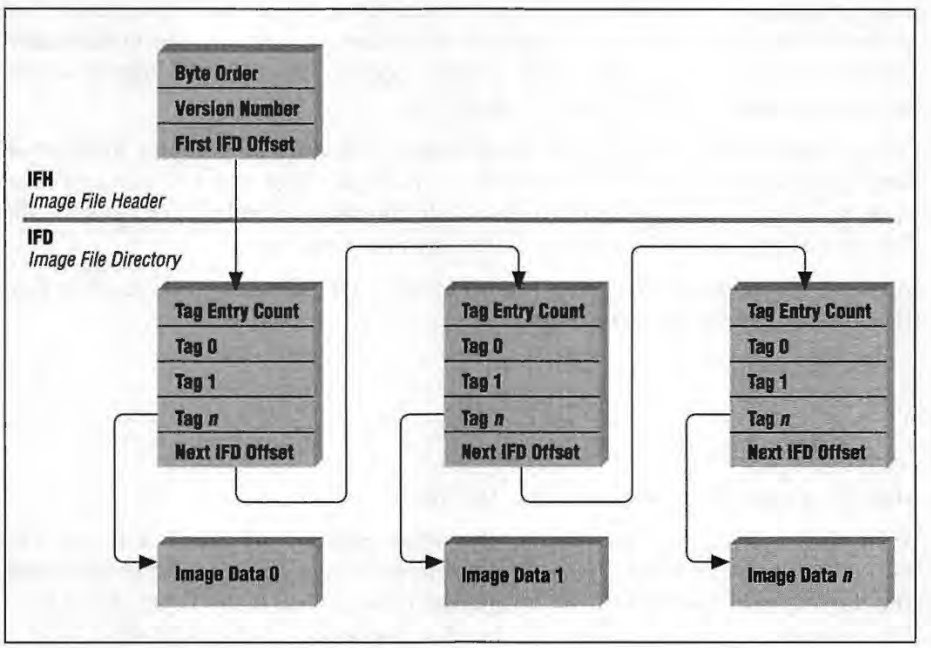


FIGURE TIFF-2: *Logical organization of a TIFF file*

File Details

This section describes the various components of a TIFF file.

Image File Header

TIFF, despite its complexity, has the simplest header of all of the formats described in this book. The TIFF Image File Header (IFH) contains three fields of information and is a total of only eight bytes in length:

```
typedef struct _TiffHeader
{
    WORD  Identifier;   /* Byte-order Identifier */
    WORD  Version;      /* TIFF version number (always 2Ah) */
    DWORD IFDOffset;    /* Offset of the first Image File Directory*/

} TIFHEAD;
```


Ex.1007 / Page 913 of 1158
APPLE INC.

Identifier contains either the value 4949h (II) or 4D4Dh (MM). These values indicate whether the data in the TIFF file is written in little-endian (Intel format) or big-endian (Motorola format) order, respectively. All data encountered past the first two bytes in the file obey the byte-ordering scheme indicated by this field. These two values were chosen because they would always be the same, regardless of the byte order of the file.

Version, according to the TIFF specification, contains the version number of the TIFF format. This version number is always 42, regardless of the TIFF revision, so it may be regarded more as an identification number, (or possibly the answer to life, the universe, etc.) than a version number.

A quick way to check whether a file is indeed a TIFF file is to read the first four bytes of the file. If they are:

```
49h 49h 2Ah 00h
```

or:

```
4Dh 4Dh 00h 2Ah
```

then it's a good bet that you have a TIFF file.

IFDOffset is a 32-bit value that is the offset position of the first Image File Directory in the TIFF file. This value may be passed as a parameter to a file seek function to find the start of the image file information. If the Image File Directory occurs immediately after the header, the value of the IFDOffset field is 08h.

Image File Directory

An Image File Directory (IFD) is a collection of information similar to a header, and it is used to describe the bitmapped data to which it is attached. Like a header, it contains information on the height, width, and depth of the image, the number of color planes, and the type of data compression used on the bitmapped data. Unlike a typical fixed header, however, an IFD is dynamic and may not only vary in size, but also may be found anywhere within the TIFF file. There may be more than one IFD contained within any file. The format of an Image File Directory is shown in Figure TIFF-1.

One of the misconceptions about TIFF is that the information stored in the Image File Directory tags is actually part of the TIFF header. In fact, this information is often referred to as the "TIFF Header Information." While it is true

Ex.1007 / Page 914 of 1158
APPLE INC.

that other formats do store the type of information found in the IFD in the header, the TIFF header does not contain this information. It is possible to think of the IFDs in a TIFF file as extensions of the TIFF file header.

A TIFF file may contain any number of images, from zero on up. Each image is considered to be a separate *subfile* (i.e., a bitmap) and has an IFD describing the bitmapped data. Each TIFF subfile can be written as a separate TIFF file or can be stored with other subfiles in a single TIFF file. Each subfile bitmap and IFD may reside anywhere in the TIFF file after the headers, and there may be only one IFD per image.

This may sound confusing, but it's not really. We have seen that the TIFF header contains an offset value that points to the location of the first IFD in the TIFF file. To find the first IFD, all we need do is seek to this offset and start reading the IFD information. The last field of every IFD contains an offset value to the next IFD, if any. If the offset value of any IFD is 00h, then there are no more images left to read in the TIFF file.

An IFD may vary in size, because it may contain a variable number of data records, called *tags*. Each tag contains a unique piece of information, just as fields do within a header. However, there is a difference. Tags may be added and deleted from an IFD much the same way that notebook paper may be added to or removed from a three-ring binder. The fields of a conventional header, on the other hand, are fixed and unmovable, much like the pages of this book. Also, the number of tags found in an IFD may vary, while the number of fields in a type header is fixed.

The format of an Image File Directory is shown in the following structure:

```
typedef struct _TifIfd
{
    WORD    NumDirEntries;    /* Number of Tags in IFD  */
    TIFTAG  TagList[];        /* Array of Tags  */
    DWORD   NextIFDOffset;    /* Offset to next IFD  */

} TIFIFD;
```

NumDirEntries is a 2-byte value indicating the number of tags found in the IFD. Following this field is a series of tags; the number of tags corresponds to the value of the NumDirEntries field. Each tag structure is 12 bytes in size and, in the sample code above, is represented by an array of structures of the data type definition TIFTAG. (See the next section for more information on TIFF tags.) The number of tags per IFD is limited to 65,535.

Ex.1007 / Page 915 of 1158
APPLE INC.

NextIFDOffset contains the offset position of the beginning of the next IFD. If there are no more IFDs, then the value of this field is 00h.

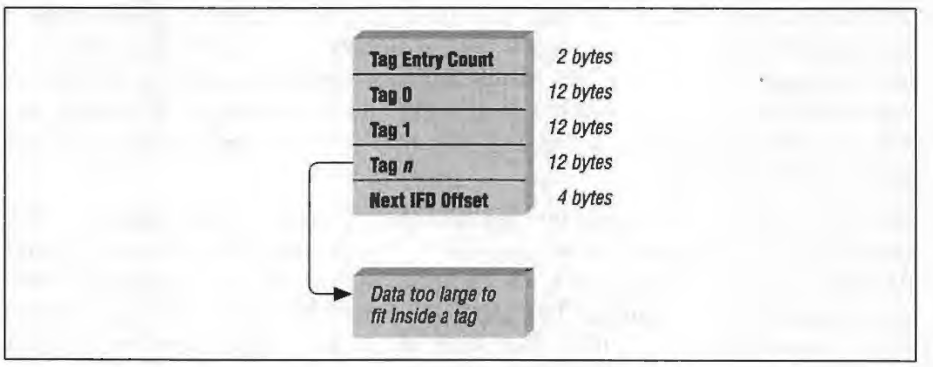


FIGURE TIFF-3: *Format of an Image File Directory*

Tags

As mentioned in the previous section, a tag can be thought of as a data field in a file header. However, whereas a header data field may only contain data of a fixed size and is normally located only at a fixed position within a file header, a tag may contain, or point to, data that is any number of bytes in size and is located anywhere within an IFD.

The versatility of the TIFF tag pays a price in its size. A header field used to hold a byte of data need only be a byte in size. A tag containing one byte of information, however, must always be twelve bytes in size.

A TIFF tag has the following 12-byte structure:

```
typedef struct _TifTag
{
    WORD   TagId;       /* The tag identifier  */
    WORD   DataType;    /* The scalar type of the data items  */
    DWORD  DataCount;   /* The number of items in the tag data  */
    DWORD  DataOffset;  /* The byte offset to the data items  */

} TIFTAG;
```

TagId is a numeric value identifying the type of information the tag contains. More specifically, the TagId indicates what the tag information represents.

Ex.1007 / Page 916 of 1158
APPLE INC.

Typical information found in every TIFF file includes the height and width of the image, the depth of each pixel, and the type of data encoding used to compress the bitmap. Tags are normally identified by their TagId value and should always be written to an IFD in ascending order of the values found in the TagId field.

DataType contains a value indicating the scalar data type of the information found in the tag. The following values are supported:

1	BYTE	8-bit unsigned integer
2	ASCII	8-bit, NULL-terminated string
3	SHORT	16-bit unsigned integer
4	LONG	32-bit unsigned integer
5	RATIONAL	Two 32-bit unsigned integers

The BYTE, SHORT, and LONG data types correspond to the BYTE, WORD, and DWORD data types used throughout this book. The ASCII data type contains strings of 7-bit ASCII character data, which are always NULL-terminated and may be padded out to an even length if necessary. The RATIONAL data type is actually two LONG values and is used to store the two components of a fractional value. The first value stores the numerator, and the second value stores the denominator.

The TIFF 6.0 revision added the following new data types:

6	SBYTE	8-bit signed integer
7	UNDEFINE	8-bit byte
8	SSHORT	16-bit signed integer
9	SLONG	32-bit signed integer
10	SRATIONAL	Two 32-bit signed integers
11	FLOAT	4-byte single-precision IEEE floating-point value
12	DOUBLE	8-byte double-precision IEEE floating-point value

The SBYTE, SSHORT, SLONG, and SRATIONAL data types are used to store signed values. The FLOAT and DOUBLE data types are used specifically to store IEEE-format single- and double-precision values. The UNDEFINE data type is an 8-bit byte that may contain untyped or opaque data and is typically used in private tags. An example of the use of this data type is to store an entire data structure within a private tag specifying the DataType as UNDEFINE (value of 7) and a DataCount equal to the number of bytes in the structure.

Ex.1007 / Page 917 of 1158
APPLE INC.

With the exception of the SMinSampleValue and SMaxSampleValue tags (which may use any data type), none of these newer data types is used by any TIFF 6.0 tags. They are therefore found only in private tags.

DataCount indicates the number of items referenced by the tag and doesn't show the actual size of the data itself. Therefore, a DataCount of 08h does not necessarily indicate that eight bytes of data exist in the tag. This value indicates that eight items exist for the data type specified by this tag. For example, a DataCount value of 08h and a DataType of 03h indicate that the tag data is eight contiguous 16-bit unsigned integers, a total of 32 bytes in size. A DataCount of 28h and a DataType of 02h indicate an ASCII character string 40 bytes in length, including the NULL-terminator character, but not any padding if present. And a DataCount of 01h and a DataType of 05h indicate a single RATIONAL value a total of eight bytes in size.

DataOffset is a 4-byte field that contains the offset location of the actual tag data within the TIFF file. If the tag data is four bytes or less in size, the data may be found in this field. If the tag data is greater than four bytes in size, then this field contains an offset to the position of the data in the TIFF file. Packing data within the DataOffset field is an optimization within the TIFF specification and is not required to be performed. Most data is typically stored outside the tag, occurring before or after the IFD (see Figure TIFF-3).

Table TIFF-1 lists all of the public tags included in the TIFF 4.0, 5.0, and 6.0 specifications. Note that some tags have become obsolete and are not found in the current revision of TIFF; however, we provide them because the TIFF 4.0 and TIFF 5.0 specs are still in some use. Also, note that several tags may support more than one data type.

In the table below, an asterisk (*) means that the tag is defined, a hyphen (-) means that the tag is not defined, and an "x" means that the tag is obsolete.

TABLE TIFF-1: ***TIFF Tag Types Listed Alphabetically by Name***

Tag Name	Tag ID	Tag Type	4.0	5.0	6.0
Artist	315	ASCII	-	*	*
BadFaxLines[a]	326	SHORT or LONG	-	-	-
BitsPerSample	258	SHORT	*	*	*
CellLength	265	SHORT	*	*	*
CellWidth	264	SHORT	*	*	*
CleanFaxData[a]	327	SHORT	-	-	-

Ex.1007 / Page 918 of 1158
APPLE INC.

Tag Name	Tag ID	Tag Type	4.0	5.0	6.0
ColorMap	320	SHORT	-	*	*
ColorResponseCurve	301	SHORT	*	*	x
ColorResponseUnit	300	SHORT	*	x	x
Compression	259	SHORT	*	*	*
Uncompressed	1		*	*	*
CCITT 1D	2		*	*	*
CCITT Group 3	3		*	*	*
CCITT Group 4	4		*	*	*
LZW	5		-	*	*
JPEG	6		-	-	*
Uncompressed	32771		*	x	x
Packbits	32773		*	*	*
ConsecutiveBadFaxLines[a]	328	LONG or SHORT	-	-	-
Copyright	33432	ASCII	-	-	*
DateTime	306	ASCII	-	*	*
DocumentName	269	ASCII	*	*	*
DotRange	336	BYTE or SHORT	-	-	*
ExtraSamples	338	BYTE	-	-	*
FillOrder	266	SHORT	*	*	*
FreeByteCounts	289	LONG	*	*	*
FreeOffsets	288	LONG	*	*	*
GrayResponseCurve	291	SHORT	*	*	*
GrayResponseUnit	290	SHORT	*	*	*
HalftoneHints	321	SHORT	-	-	*
HostComputer	316	ASCII	-	*	*
ImageDescription	270	ASCII	*	*	*
ImageHeight	257	SHORT or LONG	*	*	*
ImageWidth	256	SHORT or LONG	*	*	*
InkNames	333	ASCII	-	-	*
InkSet	332	SHORT	-	-	*
JPEGACTTables	521	LONG	-	-	*
JPEGDCTTables	520	LONG	-	-	*
JPEGInterchangeFormat	513	LONG	-	-	*
JPEGInterchangeFormatLength	514	LONG	-	-	*
JPEGLosslessPredictors	517	SHORT	-	-	*
JPEGPointTransforms	518	SHORT	-	-	*
JPEGProc	512	SHORT	-	-	*
JPEGRestartInterval	515	SHORT	-	-	*
JPEGQTables	519	LONG	-	-	*
Make	271	ASCII	*	*	*

Ex.1007 / Page 919 of 1158
APPLE INC.

TIFF (cont'd)

Tag Name	Tag ID	Tag Type	4.0	5.0	6.0
MaxSampleValue	281	SHORT	*	*	*
MinSampleValue	280	SHORT	*	*	*
Model	272	ASCII	*	*	*
NewSubFileType	254	LONG	-	*	*
NumberOfInks	334	SHORT	-	-	*
Orientation	274	SHORT	*	*	*
PageName	285	ASCII	*	*	*
PageNumber	297	SHORT	*	*	*
PhotometricInterpretation	262	SHORT	*	*	*
WhiteIsZero	0		*	*	*
BlackIsZero	1		*	*	*
RGB	2		*	*	*
RGB Palette	3		-	*	*
Tranparency Mask	4		-	-	*
CMYK	5		-	-	*
YCbCr	6		-	-	*
CIELab	8		-	-	*
PlanarConfiguration	284	SHORT	*	*	*
Predictor	317	SHORT	-	*	*
PrimaryChromaticities	319	RATIONAL	-	*	*
ReferenceBlackWhite	532	LONG	-	-	*
ResolutionUnit	296	SHORT	*	*	*
RowsPerStrip	278	SHORT or LONG	*	*	*
SampleFormat	339	SHORT	-	-	*
SamplesPerPixel	277	SHORT	*	*	*
SMaxSampleValue	341	Any	-	-	*
SMinSampleValue	340	Any	-	-	*
Software	305	ASCII	-	*	*
StripByteCounts	279	LONG or SHORT	*	*	*
StripOffsets	273	SHORT or LONG	*	*	*
SubFileType	255	SHORT	*	x	x
T4Options[b]	292	LONG	*	*	*
T6Options[c]	293	LONG	*	*	*
TargetPrinter	337	ASCII	-	-	*
Thresholding	263	SHORT	*	*	*
TileByteCounts	325	SHORT or LONG	-	-	*
TileLength	323	SHORT or LONG	-	-	*
TileOffsets	324	LONG	-	-	*
TileWidth	322	SHORT or LONG	-	-	*
TransferFunction[d]	301	SHORT	-	-	*

Ex.1007 / Page 920 of 1158
APPLE INC.

Tag Name	Tag ID	Tag Type	4.0	5.0	6.0
TransferRange	342	SHORT	-	-	*
XPosition	286	RATIONAL	*	*	*
XResolution	282	RATIONAL	*	*	*
YCbCrCoefficients	529	RATIONAL	-	-	*
YCbCrPositioning	531	SHORT	-	-	*
YCbCrSubSampling	530	SHORT	-	-	*
YPosition	287	RATIONAL	*	*	*
YResolution	283	RATIONAL	*	*	*
WhitePoint	318	RATIONAL	-	*	*

[a] Tags BadFaxLines, CleanFaxData, and ConsecutiveBadFaxLines are part of TIFF Class F now maintained by Aldus and are not actually defined by the TIFF 6.0 specification.
[b] Tag 292 was renamed from Group3Options to T4Options in TIFF 6.0.
[c] Tag 293 was renamed from Group3Options to T6Options by TIFF 6.0.
[d] Tag 301 was renamed from ColorResponseCurve to TransferFunction by TIFF 6.0.

Organization of TIFF Tag Data

To keep developers from having to guess which tags should be written to a TIFF file and what tags are important to read, the TIFF specification defines the concept of baseline TIFF images. These baselines are defined by the type of image data they store: bi-level, gray-scale, palette-color, and full-color. Each baseline has a minimum set of tags, which are required to appear in each type of TIFF file.

In the TIFF 5.0 specification, these baselines were referred to as TIFF classes. Each TIFF file consisted of a common baseline (Class X) and was modified by an additional class depending upon the type of image data stored. The classes were Class B (bi-level), Class G (gray-scale), Class P (palette-color), and Class R (full-color RGB).

The TIFF 6.0 specification redefines these classes into four separate baseline TIFF file configurations. Class X is combined with each of the other four classes to form the bi-level, gray-scale, color-palette, and full-color baselines. Although TIFF 6.0 largely does away with the concept of TIFF classes, it is likely that because more TIFF 5.0 format files exist than any other, TIFF files will be referred to by these class designations for many years to come.

It is worth mentioning that a de facto TIFF class, Class F, exists specifically for the storage of facsimile images using the TIFF format. This class of TIFF file, created by Cygnet Technologies, is used by Everex products to store facsimile

Ex.1007 / Page 921 of 1158
APPLE INC.

data, and is also known as the Everex Fax File Format. Although Cygnet Technologies is no longer in business, TIFF Class F remains in use and is considered by some to be an excellent storage format for facsimile data.

Table TIFF-2 lists the minimum required tags that must appear in the IFD of each TIFF 6.0 baseline. Note that several of these tags have default values that are used if the tag does not actually appear in a TIFF file:

- Bi-level (formerly Class B) and gray-scale (formerly Class G) TIFF files must contain the thirteen tags listed. These tags must appear in all revision 5.0 and 6.0 TIFF files regardless of the type of image data stored.
- Palette-color (formerly Class P) TIFF files add a fourteenth required tag that describes the type of palette information found within the TIFF image file.
- RGB (formerly Class R) TIFF files contain the same tags as bi-level TIFF files and add a fourteenth required tag, which describes the format of the bitmapped data in the image.
- YCbCr TIFF files add four additional tags to the baseline.
- Class F TIFF files add three tags.

TABLE TIFF-2: ***Minimum Required Tags for Each TIFF Class***

Class Name	Tag Type	Tag Name
Bi-level and	254	NewSubfileType
Gray-scale	256	ImageWidth
	257	ImageLength
	258	BitsPerSample
	259	Compression
	262	PhotometricInterpretation
	273	StripOffsets
	277	SamplesPerPixel
	278	RowsPerStrip
	279	StripByteCounts
	282	XResolution
	283	YResolution
	296	ResolutionUnit

The following classes contain the above 13 tags plus the following tags:

Ex.1007 / Page 922 of 1158
APPLE INC.

Class Name	Tag Type	Tag Name
Palette-color	320	ColorMap
RGB	284	PlanarConfiguration
YCbCr	529	YCbCrCoefficients
	530	YCbCrSubSampling
	531	YCbCrPositioning
	532	ReferenceBlackWhite
Class F	326	BadFaxLines
	327	CleanFaxData
	328	ConsecutiveBadFaxLines

All other tags found in the TIFF specification are available to meet developer's needs. While a TIFF reader must be able to support the parsing and interpretation of all tags it considers necessary, it is certainly not necessary for a TIFF writer to include as many tags as possible in every TIFF file written.

Image Data

TIFF files contain only bitmap data, although adding a few tags to support vector- or text-based images would not be a hard thing to do. As we have seen, the bitmapped data in a TIFF file does not always appear immediately after the image header, as with most other formats. Instead, it may appear almost anywhere within the TIFF file. And, because the majority of the work performed by a TIFF reader and writer is the creation and manipulation of the image data, a thorough understanding of how the image data is stored within a TIFF file is necessary, starting with the concept of *strips*.

NOTE

TIFF 6.0 images may contain tiles rather than strips.

Strips

It is always amusing to come across a TIFF reader or viewer whose author posts the caveat in the source code, "This TIFF reader does not support stripped images." A large proportion of TIFF readers who fail to read certain TIFF image files do so because the author of the reader did not quite understand the concept of how image data can be organized within a TIFF file. In this case, not only did the author of the reader fail to understand how to write code to read strips, he or she also failed to realize that every TIFF 5.0 (and earlier) image contains strips.

Ex.1007 / Page 923 of 1158
APPLE INC.

A *strip* is an individual collection of one or more contiguous rows of bitmapped image data. Dividing the image data into strips makes buffering, random access, and interleaving of the image data much easier. This concept exists in several other image file formats, and is given names such as *blocks*, *bands*, and *chunks*. TIFF strips differ from other such concepts in several important ways due to the structure of the TIFF format.

Three tags are necessary to define the strips of bitmapped data within a TIFF file. These three tags are RowsPerStrip, StripOffsets, and StripByteCounts.

The first required tag, RowsPerStrip, indicates the number of rows of compressed bitmapped data found in each strip. The default value for RowsPerStrip is $2^{32}-1$, which is the maximum possible size of a TIFF image. All of the strips in a TIFF subfile must use the same compression scheme and have the same bit sex, color sex, pixel depth, and so on. To find the number of strips in a non-YCbCr subfile image, we would use the RowsPerStrip tag, the ImageLength tag, and the following calculation:

```
StripsInImage =
      floor((ImageLength + RowsPerStrip - 1) / RowsPerStrip);
```

The second required tag, StripOffsets, is important because without it a TIFF reader has absolutely no hope of locating the image data within a TIFF file. This tag contains an array of offset values, one per strip, which indicate the position of the first byte of each strip within the TIFF file. The first value in the array is for the first strip, the second value for the second strip, and so on. If the image data is separated into planes (PlanarConfiguration = 2), StripOffsets contains a 2D array of values, which is SamplesPerPixel in width. All of the columns for color component (plane) 0 are stored first, followed by all the columns for color component (plane) 1, and so forth. The strips of planar image data may be written to the TIFF file in any order but are typically written by plane (RRRRGGGGBBBB) or by color component (RGBRGBRGBRGB). StripOffsets values are always interpreted from the beginning of the file.

The StripOffsets tag allows each strip in a TIFF file to have a location that is completely independent from all other strips in the same subfile. This means that strips may occur in any order and be found anywhere within the TIFF file. Many "quick and dirty" TIFF readers find the beginning of the first strip and then attempt to read in the image data as one large chunk without checking the StripOffsets array for the position of each additional strip. This technique works if all the strips in the TIFF file are contiguously written to the TIFF file

Ex.1007 / Page 924 of 1158
APPLE INC.

and are in the same consecutive order as the original rows in the bitmap. If the strips are stored out of sequence, perhaps in a planar or interlaced fashion, or are aligned on paragraph or page boundaries, the image data read will not be in its proper order, and the image will appear sliced up and rearranged on the display. If the strips are stored in a fairly random fashion, a large part of the data read might not be part of the image, or even the TIFF file itself. In this case, anything that is displayed would be mostly garbage.

The value of the RowsPerStrip tag and the size of each element in the array of the StripOffsets tag is usually a LONG (32-bit) value. TIFF 5.0 added the ability of this tag to use SHORT (16-bit) values instead. Very old TIFF readers may expect the values in this tag to always be LONG and will therefore read the offset values improperly if they are SHORT. This modification was made primarily for TIFF readers that read the StripOffsets values into an array in memory before using them. The TIFF 6.0 specification suggests that the offset values should not require such an array to be larger than 64K in size.

The third tag, StripByteCounts, maintains an array of values that indicates the size of each strip in bytes. And, like the StripOffsets tag, this tag is also an array of values, one per strip, 1D for chunky format and 2D for planar format, each of which is calculated from the number of bytes of compressed bitmapped data stored in each strip.

This tag is necessary because there are several cases in which the strips in an image may each contain a different number of bytes. The first case occurs when using compressed bitmapped image data. As we have said, the StripBytesCounts value is the size of the image data *after* it is compressed. Although there is a fixed number of bytes in an uncompressed row, the size of a compressed row varies depending upon the data it contains. Because we are always storing a fixed number of rows, not bytes, per strip, it is likely that most strips will be of different lengths because each compressed row will vary in size. Only when the bitmap data is not compressed will each strip be the same size.

Well, almost... The last strip in an image is sometimes an exception. TIFF writers typically attempt to create strips so that each strip in a TIFF image has the same number of rows. For example, a bitmap with 2200 rows can be divided into 22 strips, each containing 100 rows of bitmapped data. However, it is not always possible to divide the number of rows equally among the desired number of strips. For example, if we needed to divide a bitmap containing 482 rows into strips containing five rows each, we would end up with a total of 97 strips,

Ex.1007 / Page 925 of 1158
APPLE INC.

96 strips containing five rows of data and the 97th strip containing the remaining two rows. The RowsPerStrip tag value of 5 would be correct for all strip lengths except for the last strip.

The truth is that a TIFF reader does not need to know the number of rows in each strip to read the image data, only the number of bytes. Otherwise, the TIFF specification would require that every "short" strip be padded with additional rows of dummy data, but it doesn't. Instead, we simply read the last StripByteCounts value to determine how many bytes to read for the last strip. What the TIFF specification doesn't make clear is that the RowsPerStrip value specifies the *maximum* value, and not the required value, of the number of rows per strip. Many TIFF files, in fact, store a single strip of data and specify an arbitrarily large RowsPerStrip value.

There are several advantages to organizing bitmap data in strips.

First, not all applications can read an entire file into memory. Many desktop machines still have only one megabyte or less of memory available to them. And even if a system has gobs of memory, there is no guarantee that a TIFF reader will be able to use it. Such a TIFF reader can allocate the largest buffer it can manage and then read in the bitmapped data one strip at a time until the buffer is filled. If the image is panned or scrolled, data in the buffer can be discarded and more strips read in. If the entire image can fit in memory, all the strips in the TIFF file would then be read.

Because compressed strips can vary in size, the StripByteCounts values cannot be accurately used by an application to dynamically allocate a buffer in memory (unless every value is read and the largest value is used to allocate the buffer). Therefore, it is recommended that each strip be limited to about 8K in size. If a TIFF reader can allocate a much larger buffer than 8K, then multiple strips may be read into the buffer. Although TIFF strips can be larger, perhaps to support an image where each compressed or uncompressed row is greater than 8K in size, the size of a strip should never exceed 64K. Allocating a buffer greater than 64K can be a bit tricky when using certain system architectures.

Second, having access to a table of strip offsets makes random access of the bitmapped data easier. If a TIFF reader needed to display the last 100 rows of a 480-row image, and the bitmaps were divided into 48 strips of 10 rows each, the reader would skip over the first 380 rows and read in the strips stored at the last 10 offsets in the StripOffsets tag array. If no strip offsets were present, the entire image would need to be read to find the starting position of the last 100 rows.

Ex.1007 / Page 926 of 1158
APPLE INC.

And while it is possible that the bitmap in a TIFF file may be written out as one long strip—and many TIFF files are written this way—it is not a good idea to do so. These so-called unstripped images often fail because an application must attempt to allocate enough memory to hold the entire image. For large images, or small systems, enough memory may not be available. One can only hope that a TIFF file reader would exit from such a situation gracefully.

Tiles

Strips are not the only possible way to organize bitmapped data. TIFF 6.0 introduced the concept of *tiled*, rather than stripped, bitmapped data. A strip is a 1D object that only has a length. A tile can be thought of as a 2D strip that has both width and length, very similar to a bitmap. In fact, you can think of each tile in an image as a small bitmap containing a piece of a larger bitmap. All you need to do is fit the tiles together in their proper locations to get the image. (This concept only serves to remind me that I must replace the linoleum in my bathroom one day.)

Dividing an image into rectangular tiles rather than horizontal strips has the greatest benefit on very large high-resolution images. Many electronic document imaging (EDI) applications cannot manipulate images larger than E size (6848 pixels wide by 8800 pixels long) because of the large amount of memory required to buffer, decompress, and manipulate even a few hundred rows of image data. Even if you just wanted to display the upper-left corner of an image you would still be forced to decompress the entire strip and maintain it in memory. If the image data were tiled, however, you would only decompress the tiles that contained the image data for the part of the image you wanted to display.

Many compression algorithms, such as JPEG, compress data not as 1D strips, but instead as 2D tiles. Storing the compressed data as tiles optimizes the decompression of the data quite a bit. In fact, the support of 2D compression algorithms is perhaps the primary reason why the capability of tiling image data was added to TIFF 6.0.

When tiles are used instead of strips, the three strip tags, RowsPerStrip, StripByteCounts, and StripOffsets, are replaced by the tags TileWidth, TileLength, TileOffsets, and TileByteCounts. As you might have guessed, the tile tags are used in much the same way that the strips tags are. And, like strips, tiles are either all uncompressed, or all compressed using the same scheme. Also, TIFF images are either striped or tiled, but never both.

Ex.1007 / Page 927 of 1158
APPLE INC.

TileWidth and TileLength describe the size of the tiles storing the image data. The values of TileWidth and TileLength must be a multiple of 16, and all tiles in a subfile are always the same size. These are important compatibility considerations for some applications, especially when using the tile-oriented JPEG compression scheme. The TIFF 6.0 specification recommends that tiles should contain 4K to 32K of image data before compression. Finally, tiles need not be square. Rectangular tiles compress just as well.

The TileWidth and TileLength tag values can be used to determine the number of tiles in non-YCbCr image subfiles:

```
TilesAcross = (ImageWidth + (TileWidth - 1)) / TileWidth;
TilesDown = (ImageLength + (TileLength - 1)) / TileLength;
TilesInImage = TilesAcross * TilesDown;
```

If the image is separated into planes (PlanarConfiguration = 2), the number of tiles is calculated like this:

```
TilesInImage = TilesAcross * TilesDown * SamplesPerPixel;
```

The TileOffsets tag contains an array of offsets to the first byte of each tile. Tiles are not necessarily stored in a contiguous sequence in a subfile. Each tile has a separate location offset value and is independent of all other tiles in the subfile. The offsets in this tag are ordered left-to-right and top-to-bottom. If the image data is separated into planes, all of the offsets for the first plane are stored first, followed by the offsets for the second plane, and so on. The number of offset values in this tag are equal to the number of tiles in the image (PlanarConfiguration = 1) or the number of tiles multiplied by the SamplesPerPixel tag value (PlanarConfiguration = 2). All offset values are interpreted from the beginning of the TIFF file.

The final tag, TileByteCounts, contains the number of bytes in each compressed tile. The number of values in this tag is also equal to the number of tags in the image, and the values are ordered the same way as the values in the TileOffsets tag.

Normally, a tile size is chosen that fits an image exactly. An image 6400 pixels wide by 4800 pixels long may be divided evenly into 150 tiles, each 640 pixels wide by 320 pixels long. However, not all image dimensions are divisible by 16. An image 2200 pixels wide by 2850 long cannot be evenly divided into tiles whose size must be multiples of 16. The solution is to choose a "best-fit" tile size and fill out the image data with padding.

Ex.1007 / Page 928 of 1158
APPLE INC.

To find a best-fit tile size, we must choose a tile size that minimally overlaps the size of the image. In this example, we want to use tiles that are 256 pixels wide by 320 pixels long, roughly the same aspect ratio as the image. Using tiles this size requires that 104 extra pixels of padding be added to each row and that 30 additional rows be added to the image length. The size of the image data plus padding is now 2304 pixels wide by 2880 pixels long and can be evenly divided among 81 of our 256 by 320 pixel tiles.

In this example, you may notice that the total amount of padding added to the image before tiling is 365,520 pixels. For a 1-bit image, this equals an extra 45,690 bytes of image data. No appreciable gain in compression results from tiling small images. Also, avoid using tiles that are excessively large and require excessive amounts of padding.

Compression

TIFF supports perhaps more types of data compression than any other image file format. It is also quite possible to use an unsupported compression scheme just by adding the needed user-defined tags. TIFF 4.0 supported only Run-Length Encoding (RLE) and CCITT T.4 and T.6 compression. These compression schemes are typically only for use with 8-bit color, and gray-scale and 1-bit black-and-white images, respectively. TIFF 5.0 added the LZW compression scheme, typically for color images, and TIFF 6.0 added the JPEG compression method for use with continuous-tone color and gray-scale images. (All of these data compression schemes, including a variety of RLE algorithms, are discussed in Chapter 9.)

TIFF uses the PackBits RLE compression scheme found in the Macintosh toolbox. PackBits is a simple and effective algorithm for compressing data and is easy to implement. The name "PackBits" would lead a programmer to believe that it is a bit-wise RLE, packing runs of bits. However, PackBits is a byte-wise RLE and is most efficient at encoding runs of bytes.

PackBits actually has three types of data packets that may be written to the encoded data stream. The first is a two-byte *encoded run packet.* The first byte (the run-count byte) indicates the number of bytes in the run, and the second byte stores the value of each byte in the run. The actual run-count value stored is in the range 0 to 127 and represents the values 1 to 128 (run count + 1).

Ex.1007 / Page 929 of 1158
APPLE INC.

LZW Is Not Free

If you are creating or modifying software that implements the LZW algorithm, be aware that under certain circumstances, you will need to pay a licensing fee for the use of LZW.

Unisys Corporation owns the patent for the LZW codec (encoding/decoding algorithm) and requires that a licensing fee be paid for each software program which implements the LZW algorithm.

Many people have concluded that the Unisys licensing claim applies only to LZW encoders (software that creates LZW data) and not to LZW decoders (software that only reads LZW data). However, Unisys believes that its patent covers the full LZW codec and requires a licensing fee even for software that reads, but does not write, LZW data.

For more information about the entire issue of LZW licensing, refer to the section called "LZW Legal Issues" in Chapter 9. For a popular alternative to graphics file formats that use LZW, consider using the Portable Network Graphics (PNG) file format, described in Part Two of this book.

Another type of packet, the *literal run packet*, stores 12 to 128 bytes literally in the encoded data stream without compressing them. Literal run packets are used to store data with very few runs, as found in very complex or noisy images. The literal run packet's run count is in the range of −127 to −1, indicating that 2 to 128 run values (−(run count) + 1) follow the run count byte.

The last type of packet is the no-op packet. No-ops are one byte in length and have a value of −128. The no-op packet has no use in the PackBits compression scheme and is therefore never found in PackBits-encoded data.

Decompressing PackBits-encoded data is a simple matter of reading a packet of encoded data and converting it to the appropriate byte run. Once again, the run-count byte value is stored one less than the actual number of bytes in the run. It is therefore necessary to add one to the run-count value before using it.

Refer to the TIFF 6.0 specification for more information on PackBits compression, and to Chapter 9, for more information on RLE algorithms.

Ex.1007 / Page 930 of 1158
APPLE INC.

Problems with TIFF 6.0 JPEG

Commentary by Dr. Tom Lane of the Independent JPEG Group, a member of the TIFF Advisory Committee

TIFF 6.0 added JPEG to the list of TIFF compression schemes. Unfortunately, the approach taken in the 6.0 specification is a very poor design. A new design has been developed by the TIFF Advisory Committee. If you are considering implementing JPEG in TIFF, I strongly urge you to follow the revised design described in TIFF Tech Note #2 rather than that of the 6.0 spec.

The fundamental problem with the TIFF 6.0 JPEG design is that JPEG's various tables and parameters are broken out as separate fields, which the TIFF control logic must manage. This is bad software engineering—that information should be private to the JPEG compressor/decompressor. Worse, the fields themselves are specified without thought for future extension and without regard to well-established TIFF conventions. Here are some of the more significant problems:

- The JPEG table fields use a highly nonstandard layout: rather than containing data directly in the field structure, the fields hold pointers to information elsewhere in the file. This requires special-purpose code to be added to *every* TIFF-manipulating application. Even a trivial TIFF editor (for example, a program to add an ImageDescription field to a TIFF file) must be explicitly aware of the internal structure of the JPEG-related tables, or it will probably break the file. Every other auxiliary field in TIFF follows the normal TIFF rules and can be copied or relocated by standard code.
- The specification requires the TIFF control logic to know a great deal about JPEG details—for example, such arcana as how to compute the length of a Huffman code table. The length is not supplied in the field structure and can be found only by inspecting the table contents.
- The design specifies separate Huffman tables for each color component. This neglects the fact that baseline JPEG codecs may support only two sets of Huffman tables. Thus, an encoder must either waste space (by storing duplicate Huffman tables) or violate the TIFF convention that prohibits

Ex.1007 / Page 931 of 1158
APPLE INC.

duplicate pointers. Furthermore, baseline decoders must test to see which tables are identical—a waste of time and code space.

- The JPEGInterchangeFormat field again violates the proscription against duplicate pointers; it envisions having the normal strip/tile pointers pointing into the larger data area pointed to by JPEGInterchangeFormat. TIFF editing applications must be specifically aware of this relationship, because they must maintain it or, if they can't, must delete the JPEGInterchangeFormat field.
- The JPEGQTables field is fixed at a byte per table entry; there is no way to support 16-bit quantization values. This is a serious impediment to extending TIFF to use 12-bit JPEG.
- The design cannot support using different quantization tables in different strips/tiles of an image (so as to encode some areas at higher quality than others). Furthermore, because quantization tables are tied one-for-one to color components, the design cannot support table switching options that are likely to be added in future JPEG extensions.

In addition to these major design errors, the TIFF 6.0 JPEG specification is seriously ambiguous. In particular, several incompatible interpretations are possible for its handling of JPEG restart markers, and Section 22, "JPEG Compression," actually contradicts Section 15, "Tiled Images," about the restrictions on tile sizes.

Finally, the 6.0 design creates problems for implementations that need to keep the JPEG codec separate from the TIFF control logic—consider using a JPEG chip that was not designed specifically for TIFF. JPEG codecs generally want to produce or consume a standard JPEG datastream, not just raw data. (If they do handle raw data, a separate out-of-band mechanism must be provided to load tables into the codec.) With such a codec, the TIFF control logic must be prepared to parse JPEG markers to create the TIFF table fields (when encoding) or to synthesize JPEG markers from the fields (when decoding). Of course, this means that the control logic must know a great deal more about JPEG than we would like. The parsing and reconstruction

Ex.1007 / Page 932 of 1158
APPLE INC.

of the markers also represents a fair amount of unnecessary work.

Due to all these problems, the TIFF Advisory Committee has developed a replacement JPEG-in-TIFF scheme. The rough outline is as follows:

1. Each image segment (strip or tile) in a JPEG-compressed TIFF image contains a legal JPEG datastream, complete with all markers. This data forms an independent image of the proper dimensions for the strip or tile.
2. To avoid duplicate tables in a multi-segment file, segments may use the JPEG "abbreviated image data" datastream structure, in which DQT and DHT tables are omitted. The common tables are to be supplied in a JPEG "abbreviated table specification" datastream, which is contained in a newly defined "JPEGTables" TIFF field. Because the tables in question typically amount to 550 bytes or so, the savings are worthwhile.
3. All the field definitions in the existing Section 22, "JPEG Compression," are deleted. (In practice, those field tag values will remain reserved indefinitely, and this scheme will use a new Compression code, Compression = 7. Implementations that have TIFF 6.0-style files to contend with may continue to read them, using whatever interpretation of 6.0 they used before.)

This replacement design is described in TIFF Tech Note #2. The Tech Note is currently being distributed in draft form, because Adobe has not yet formally accepted it. However, I expect that the Note will be accepted as is, and that the design it describes will replace the existing Section 22 when version 7.0 of the TIFF spec is released.

If you are considering implementing JPEG in TIFF, please use the design of the Tech Note. The 6.0 JPEG design has not been widely implemented, and with any luck it will stay that way.

Ex.1007 / Page 933 of 1158
APPLE INC.

For Further Information

For further information about the TIFF format, see the specification included on the CD-ROM that accompanies this book.

TIFF was formerly maintained by the Aldus Developer's Association. Aldus has recently merged with Adobe Systems, which now holds the copyright to the TIFF specification and administers and maintains the TIFF format.

All information on the TIFF format may now be obtained through the Adobe Developer Support group. However, this group supplies only general TIFF information and does not provide any TIFF tutoring, sample TIFF source code, or sample TIFF files. Contact the Adobe Developer Support group, at *devsupperson@adobe.com*

Questions about the the Adobe Developer's Association should be directed to:

Adobe Developer's Association
1585 Charleston Road
P.O. Box 7900
Mountain View CA 94039-7900
Voice: 415-961-4111
FAX: 415-967-9231
FTP: *ftp://ftp.adobe.com*
WWW: *http://www.adobe.com/Support/ADA.html*
BBS: 206-623-6984

Adobe distributes the TIFF 6.0 specifiction in PDF format in the "Technical Notes for Developers" section on the Adobe homepage, at:

http://www.adobe.com/Support/TechNotes.html#tiff
http://www.adobe.com/PDFs/TN/TIFF6.pdf

or on the Adobe FTP server:

ftp://ftp.adobe.com/pub/adobe/DeveloperSupport/TechNotes/PDFfiles/TIFF6.pdf

or in paper form for $25US by calling 1-800-831-6395.

TIFF support in Europe may be obtained via email or from Adobe's BBS in Edinburgh, Scotland:

Email: *eurosupport@adobe.com*
BBS: +44 131 458 4666

Ex.1007 / Page 934 of 1158
APPLE INC.

The Adobe Acrobat reader for PDF files may be obtained for free from:

ftp://ftp.adobe.com/pub/adobe/Applications/Acrobat/

Technical information on Aldus products, including the TIFF Class F specification, is available from Adobe's Automated Technical Support for Aldus Products FAXback service in which information may be requested automatically via a FAX machine. This service may be reached at 800-288-6832 (toll-free), or 206-628-5728.

You will also find useful TIFF information and tools at:

ftp://ftp.sgi.com/graphics/tiff/ (maintained by Sam Leffler)
http://www-mipl.jpl.nasa.gov/~ndr/tiff/index.html (maintained by Niles Ritter)

See the following references for more information about TIFF:

Aldus Corporation. *TIFF Developer's Toolkit*, Revision 5.0. Seattle, WA, November 1988.

Aldus Corporation. *TIFF Developer's Toolkit*, Revision 6.0, Seattle, WA, June 1992.

Aldus Corporation. *Aldus Developer News*, Seattle, WA.

Campbell, Joe. *The Spirit of TIFF Class F*, Cygnet Technologies, Berkeley, CA. April 1990.

Hewlett-Packard Company. *HP TIFF Developer's Manual*, Greeley, CO, November 1988.

Katz, Alan, and Danny Cohen, "RFC 1314: A File Format for the Exchange of Images in the Internet," USC Information Sciences Institute, Marina Del Rey, CA, April 1992.

> This document outlines the use of TIFF as a standard format for exchanging facsimile images within the Internet.

Murray, James. "TIFF File Format," *C Gazette*, Winter 1990–91, pp. 27–42.

Ex.1007 / Page 935 of 1158
APPLE INC.

RFC 804, "Standardization of Group 3 Facsimile Apparatus for Document Transmission."

This document is a draft of the CCITT Recommendation T.4 explaining the CCITT Group 3 encoding scheme used by TIFF.

Ex.1007 / Page 936 of 1158
APPLE INC.

NAME:	TTDDD
ALSO KNOWN AS:	Textual Three-Dimensional Data Description
TYPE:	Vector/animation
COLORS:	16 million
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	ASCII
ORIGINATOR:	Glenn Lewis
PLATFORM:	All
SUPPORTING APPLICATIONS:	T3DLIB
SPECIFICATION ON CD:	Yes (documentation by the author)
CODE ON CD:	Yes
IMAGES ON CD:	Yes
SEE ALSO:	IFF, TDDD

USAGE: Used primarily as a method of editing and interchanging TDDD file format information.

COMMENTS: For a complete understanding of TTDDD you need to read about the IFF and TDDD file formats as well.

Overview

The TTDDD (Textual Three-Dimensional Data Description format) is an ASCII data representation of the TDDD (T3D) file format used by Impulse for its Turbo Silver and Imagine rendering and animation software products. In fact, so much of the information in the TDDD specification is relevant to TTDDD that no official separate TTDDD file format specification has ever been written.

TTDDD was created as part of two, now obsolete, programs, ReadTDDD and WriteTDDD. These programs were capable of reading a TDDD file and writing out a TTDDD file and vice versa. They have now been replaced by the newer T3DLIB shareware library and utilities. (See "For Further Information" later in this article for more details about T3DLIB.)

Ex.1007 / Page 937 of 1158
APPLE INC.

File Organization

All TTDDD files are simple ASCII files that may be read with any text editor. Each line contains a 4-letter case-insensitive keyword, followed by an optional parameter name and one or more values. Array subscripts need not be contained within double quotes. Comments may appear in TTDDD files in the form of the standard C comment tokens /* */ surrounding the comment, or the TeX token % appearing at the beginning of the comment. Here are some examples of valid keyword/parameter/value combinations:

```
NAME "Fred"                 % Keyword and value
NAME Fred                   % Same, but no double quotes
BRSH[0]="Brush 0"           % String assigned to array element 0
BRSH 0 =Brush 0             % Same, but no double quotes or brackets
OBSV Focal  2.82            % Keyword, parameter, and one value
OBSV Rotate Z=2.7 X=4.3     % Keyword, parameter, and two values
FADE FadeTo 23 99 254       % Keyword, parameter, and three values
OSTR Info ABS_TRA Z_ALIGN   % Keyword, parameter, and two flags
```

File Details

TTDDD files do not have a header, other than possibly a few comment lines identifying the file contents, author, and time and date of creation. The actual vector data is organized as a series of chunks, each enclosed by a BEGIN and END keyword pair. The INFO chunk usually appears first in older TTDDD files. Newer TTDDD files do not typically have an INFO chunk because the Imagine product does not use them.

An example of an INFO chunk is shown here:

```
INFO Begin
  NAME "Gizmo"
  BRSH[0]="This is the IFF ILBM filename of Brush 0"
  BRSH 1 ="Brush 1"                 % Brackets are optional.
  STNC[0] "Stencil 0 filename"      % etc.
  TXTR[0] Texture                   % Quotes not necessary here.
  OBSV Camera X=5.0 Y=2.7 Z=5.3     % Default: -100 -100 100
  OBSV Rotate Z=2.7 X=4.3           % In this case, Y=0.0
  OBSV Focal 2.82
  OTRK "Object for Camera Tracking"
                                    % If parameters are too long,
                                    % they are simply truncated.
  OSTR Path "Path object name"
  OSTR Translate X=2.8 Y=7.3 Z=2.1
                                    % Defaults to zero.
  OSTR Rotate Y=90                  % Defaults to zero.
  OSTR Scale 2.5
```


Ex.1007 / Page 938 of 1158
APPLE INC.

```
  OSTR Info ABS_TRA Z_ALIGN      % Keep all flags on THIS LINE!
  FADE FadeAt 50.0
  FADE FadeBy 500
  FADE FadeTo 23 99 254          % "R=23 G=99 B=254"
                                 % Defaults to 80 80 80
  SKYC Horizon R=12 B=30         % Defaults to zero.
  SKYC Zenith R=2 B=50
  AMBI 5 27 32                   % Also RGB
  GLB0[0] 5                      % Edging
  GLB0 1 2                       % Perturb
  GLB0[2] 255                    % Sky_Blend
  GLB0[3] 2                      % Lens
  GLB0 4 1                       % Fade
  GLB0[5]=9                      % Size
  GLB0 6 =2                      % Resolve
  GLB0[7] 0                      % Extra
End INFO
```

The order of data within each Begin/End block pair is completely arbitrary, except for variable-length arrays. These arrays must begin with the Count field, specifying the number of entries in the array, followed by the values for the array starting with the first element.

Following the INFO chunk will always be one OBJ chunk per object hierarchy stored in the TTDDD file. Each OBJ chunk contains one DESC sub-chunk or an EXTR sub-chunk. DESC sub-chunks describe object data stored internally within the TTDDD file. EXTR sub-chunks describe the same type of data stored in an external file. Newer TTDDD files typically do not have EXTR sub-chunks. External information is usually imported directly into the TTDDD file as a DESC sub-chunk.

Following is an example of a DESC and an EXTR sub-chunk:

```
OBJ Begin
/*
** DESC sub-chunk
*/
DESC Begin
  NAME "Gizmo"                   % Object name. Defaults to no name
  SHAP Shape 2                   % This must be supplied!
  SHAP Lamp 0                    % ditto
  POSI X=5.7 Y=200.9 Z=132.7     % Defaults to zero.
  AXIS XAxis X=1                 % Defaults to 1 0 0
  AXIS YAxis Y=1                 % Defaults to 0 1 0
  AXIS ZAxis Z=1                 % Defaults to 0 0 1
  SIZE X=1 Y=1 Z=1               % Defaults to 32.0
  PNTS PCount 3
  PNTS Point[0] 12 27 52         % Brackets and "X=..." optional
  PNTS Point 1  21 72 25         % ditto
```


Ex.1007 / Page 939 of 1158
APPLE INC.

```
  PNTS Point[2] 72 25 21          % ditto
  EDGE ECount 3
  EDGE Edge[0] 0 1                % Edge connection between two points
  EDGE Edge 1  1 2                % Brackets optional
  EDGE Edge 2  2 0
  FACE TCount 1
  FACE Connect[0] 0 1 2           % List of 3 edges to make a triangle
  COLR 87 23 232                  % RGB. Defaults to 240 240 240
  REFL G=12 R=240 B=97            % RGB. Defaults to zero.
  TRAN 25 72 53                   % RGB. Defaults to zero.
  CLST Count 1                    % Must match TCount above.
  CLST Color[0] 240 12 57         % RGB. Defaults to 240 240 240
  RLST Count 1                    % Must match TCount above.
  RLST Color[0] 120 24 23         % RGB. Defaults to zero.
  TLST Count 1                    % Must match TCount above.
  TLST Color[0] 255 92 87         % RGB. Defaults to zero.
  TPAR[0] 42.73
  TPAR[12]=72.67                  % Defaults to zero.
  SURF[2]=0
  SURF[4] 1                       % Defaults to zero.
  MTTR Type 4                     % Defaults to zero.
  MTTR Index 2.972                % Defaults to 1.0.
                                  % (1.00 <= Index <= 3.55)
  SPEC Specularity 28             % Defaults to zero.
  SPEC Hardness 16                % Defaults to zero.
  PRP0[0] 100                     % Blending factor. Defaults to 255.
  PRP0[1]=5                       % Roughness factor. Defaults to 0.
  PRP0 2 =1                       % Shading On/Off flag (1/0).
                                  % Defaults to 1.
  PRP0[3] 1                       % Phong shading flag. Defaults to 1
  PRP0[4]=0                       % Glossy flag. Defaults to 0
  PRP0 5  1                       % Quickdraw flag. Defaults to 1
  INTS 200                        % Defaults to 300
  STRY Path "Path object name"
  STRY Translate 2.8 7.3 2.1      % Defaults to zero.
  STRY Rotate Z=90                % Defaults to zero.
  STRY Scale 3.5
  STRY Info ABS_TRA X_ALIGN       % Keep all the flags on THIS LINE!
End DESC
% Possible child (including external) objects go here to
% build object hierarchy.
/*
** EXTR sub-chunk
*/
EXTR Begin
  MTRX Translate 34 72 56         % Defaults to zero.
  MTRX Scale 1 5 9                % Defaults to 1.
  MTRX Rotate 1 0 0 0 1 0 0 0 1       % A Matrix MUST be in the proper order!
  LOAD "External file name"
```


Ex.1007 / Page 940 of 1158
APPLE INC.

```
End EXTR
TOBJ    % Ends current object hierarchy.
End OBJ
```

For Further Information

For further information about TTDDD, see the documentation included on the CD-ROM that accompanies this book. TTDDD was created by Glenn M. Lewis for his shareware package T3DLIB. The current revision of T3DLIB contains many useful utilities and a platform-independent library for C programmers to use that allows easy manipulation of TDDD objects algorithmically.

You can obtain the T3DLIB package via FTP from any Aminet site, such as:

ftp://ftp.wustl1.edu/pub/Aminet/gfx/3d/
ftp://ftp.luth.se/pub/Aminet/gfx/3d/

The T3DLIB files have R42 in their names.

You can also contact the author for information:

Glenn M. Lewis
8341 Olive Hill Court
Fair Oaks, CA 95628
Voice: 916-721-7196
Email: *glewis@pcocd2.intel.com*

Ex.1007 / Page 941 of 1158
APPLE INC.

uRay

NAME:	uRay
ALSO KNOWN AS:	DBW_uRay, Microray
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	David B. Wecker
PLATFORM:	All
SUPPORTING APPLICATIONS:	uRay ray tracing application
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes
IMAGES ON CD:	No
SEE ALSO:	NFF, POV, PRT, QRT, Radiance

USAGE: Description of 3D scenes for ray tracing or other rendering applications.

COMMENTS: A simple scene description format useful for specifying 3D objects. This is a good format to look at if you are considering building your own ray-trace format or application.

Overview

uRay is the format created by David B. Wecker for use by the uRay (Microray) ray tracer. It provides a simple way of specifying a small number of 3D objects.

File Organization

Like those of many other ray-trace formats, uRay files consist of a number of ASCII lines implementing a proprietary command language, in this case the uRay scene description language.

Ex.1007 / Page 942 of 1158
APPLE INC.

File Details

Commands used in writing uRay files may consist of the following:

DEPTH	Recursion depth
COLS	Columns in an image
ROWS	Rows in an image
START	Row number at which to begin rendering operations
END	Row number at which to end rendering operations
BPP	Bits per pixel in output image (12 or 24)
AOV	View angle
ASPECT	Image aspect ratio
NEAR	Background color for "sky" near eye
FAR	Background color for "sky" far from eye
GROUND	Background color below horizon
BASE	Blackness of shadows

WAVES n where n is the number of lines following the WAVES keyword. Each line consists of the following:

x y z	Wave center
amp	Starting amplitude of the wave
phase	Starting phase shift
length	Wave length
damp	Damping between successive waves

ATTRIBUTES n

where n is the number of lines following the ATTRIBUTES keyword. Each line consists of the following:

r g b	Diffuse color
Kd	Diffuse coloring
Ks	Reflection
Kt	Transmission
Ir	Index of refraction
Kl	Self lighting
dist	Inverse square law distance
Kf	Fuzz

Ex.1007 / Page 943 of 1158
APPLE INC.

Wave	Wave number	
tex	Texture; may be one of the following:	
	0	No texture
	1 r g b x y z	Checks in color (r g b) at scale (x y z)
	2 r g b	Random mottling
	3 r g b a b c	Blend in x direction
	4 r g b a b c	Blend in y direction
	5 r g b a b c	Blend in z direction

The following objects are predefined:

`SPHERE`	Sphere
`QUAD`	Rectangle
`TRIANGLE`	Triangle
`RING`	Ring

See the uRay documentation for detailed information about parameters.

For Further Information

For further information about the uRay format, see the uRay documentation included on the CD-ROM accompanying this book. You can also contact the author:

David B. Wecker
Digital Equipment Corporation
Cambridge Research Lab
One Kendall Square
Cambridge, MA 02139
Voice: 617-621-6699
FAX: 617-621-6650
Email: *wecker@crl.dec.com*

Ex.1007 / Page 944 of 1158
APPLE INC.

NAME:	Utah RLE
ALSO KNOWN AS:	RLE
TYPE:	Bitmap
COLORS:	256
COMPRESSION:	RLE
MAXIMUM IMAGE SIZE:	32K×32K
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	University of Utah
PLATFORM:	UNIX, others
SUPPORTING APPLICATIONS:	Utah Raster Toolkit, others
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	Chapter 9, *Data Compression*

USAGE: Primarily used in support of the Utah Raster Toolkit, though other applications, mainly on UNIX platforms, occasionally choose to support it.

COMMENTS: A simple, well-defined and well-documented bitmap format. Too bad it's not in wider use. Consider using it in a pinch for interapplication exchange of 8-bit images if you have no alternative.

Overview

The Utah RLE format was developed by Spencer Thomas at the University of Utah Department of Computer Science. The first version appeared around 1983. The work was partially funded by the NSF, DARPA, the Army Research Office, and the Office of Naval Research. It was developed mainly to support the Utah Raster Toolkit (URT), which is widely distributed in source form on the Internet. Although superseded by more recent work, the Utah Raster Toolkit remains a source of ideas and bitmap manipulation code for many.

Utah RLE was intended to be device-independent. Documentation associated with the URT claims that Utah RLE format files often require about one-third of the available space necessary for most "image synthesis"-style images. If the image data does not compress well, the format accommodates storage as

Ex.1007 / Page 945 of 1158
APPLE INC.

uncompressed pixel data with little extra overhead. Despite its age, slightly idiosyncratic terminology, and some missing information, the Utah RLE format specification is well-written and reasonably clear. Aspiring format creators, take note!

File Organization

The file consists of a header, followed by palette information, a comment area, and the bitmap data.

File Details

The header looks like this:

```
typedef struct _RLE_Header
{
  WORD  Magic;          /* Magic number */
  SHORT Xpos;           /* Lower left x of image */
  SHORT Ypos;           /* Lower left y of image */
  SHORT Xsize;          /* Image width */
  SHORT Ysize;          /* Image height */
  BYTE  Flags;          /* Misc flags */
  BYTE  Ncolors;        /* Number of colors */
  BYTE  Pixelbits;      /* Number of bits per pixel */
  BYTE  Ncmap;          /* Number of color channels in palette */
  BYTE  Cmaplen;        /* Color map length  */
  BYTE  Redbg;          /* Red value of background color */
  BYTE  Greenbg;        /* Green value of background color */
  BYTE  Bluebg;         /* Blue value of background color */

} RLE_HEADER;
```

Magic is, naturally enough, a magic number identifying the file type. Unfortunately, the magic number is not documented in the specification.

Xpos and Ypos are the position of the lower-left corner of the image, in pixels.

Xsize and Ysize are the size of the image, in pixels.

Flags is an 8-bit field used by the format writer to store miscellaneous information.

Ncolors is the total number of colors in the image, limited to 256.

Pixelbits provides information on the number of bits per pixel in the image. This is currently limited to 8.

Ncmap denotes the number of colors channels in the palette.

Ex.1007 / Page 946 of 1158
APPLE INC.

Cmaplen provides the number of bits in the palette and limits ncolors (defined above), the total number of colors in the image. For example, a cmaplen value of 8 denotes a 256-color palette.

Redbg, Greenbg, and Bluebg together provide a 24-bit specification of the image background color.

A key concept used in the Utah RLE format specification is that of the color channel. Each color channel contains eight bits, and the format supports up to 255 of them. This makes for an extremely flexible color model, but one, perhaps, that has never been utilized fully. Gray-scale images, for example, normally use one color channel; 24-bit RGB images use three channels; and RGBA, where an alpha channel is included, uses four channels.

The comment area is a series of null-terminated ASCII strings, each of which is preceded by a 16-bit length value. Each comment is WORD-aligned and is padded if necessary. Comments have the form:

```
name=value
```

The bitmap data is in stream format and is amply documented in the specification and the sample code.

For Further Information

For further information about the Utah RLE format, see the following article included on the CD-ROM:

> Thomas, Spencer W., *Design of the Utah RLE Format*, University of Utah, Department of Computer Science.

See also the URT for actual implemention details.

The URT package may be obtained via FTP from the sites listed below.

ftp://cs.utah.edu/
ftp://weedeater.math.yale.edu/
ftp://freebie.engin.umich.edu/

You will find the package in the *urt-3.0.tar.Z* file, and sample images in *urt-img.tar.Z*.

URT is copyrighted, but is freely redistributable on a GNU-like basis. You can send questions about the URT or Utah RLE to either of the following:

toolkit-request@cs.utah.edu
urt-request@caen.engin.umich.edu

Ex.1007 / Page 947 of 1158
APPLE INC.

NAME:	VICAR2
ALSO KNOWN AS:	Planetary File Format
TYPE:	Bitmap
COLORS:	Unlimited
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	NASA
PLATFORM:	VAX/VMS
SUPPORTING APPLICATIONS:	Many
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	FITS, PDS

USAGE: Storage of bitmap data.

COMMENTS: You are unlikely to see the VICAR2 format unless you deal with planetary data from NASA or JPL.

Overview

The VICAR2 image file format is used primarily for storing planetary image data collected by both interplanetary spacecraft and Earth-based stations. Many astronomical and astrophysical organizations use and support the VICAR2 format.

File Organization

VICAR2 is very similar in construction to the FITS and PDS image file formats. A VICAR2 image file is divided into an ASCII header (called a label) and a collection of binary image data.

Ex.1007 / Page 948 of 1158
APPLE INC.

File Details

This section contains information on the label and image data in a VICAR2 file.

Labels

A VICAR2 label contains system label items and may contain history label items as well. History label items are added to the label during software processing, and it is possible for history label items not to be present in a VICAR2 label at all. If history label items do exist, they always occur following the last system label item.

The history label section may also contain other, software-specific keywords called user labels. They are for informational purposes for any software application that knows how to interpret them. If a VICAR2 reader does not recognize them, then they are ignored and always preserved whenever the file is written or read.

The label is always located at the beginning of the file, starting on the first byte, and is always arranged in system-/history-/user-section order.

All label sections contain one or more fixed-length records in the form of ASCII character strings. Because all records in the label must be the same length, the last record must be padded out to the proper size with NULL (ASCII 00h) characters if it is too short.

All records in the label use the standard *keyword=value* format for storing information about the image. Keywords are written in uppercase and have a maximum of eight characters. The value may be one of four data types:

- INTEGER (fixed-point)
- REAL (floating-point)
- DOUBLE (double-precision REAL)
- STRING

STRING data is delimited by single quotation marks (') containing any printable ASCII characters and spaces except for single quotation mark characters. Strings may be from 1 to 512 characters in length.

A *keyword=value* pair is always separated by an equals sign (=). There are never any spaces between the equal sign and the keyword/value data. Each *keyword=value* pair is separated from the next pair by one more spaces.

Ex.1007 / Page 949 of 1158
APPLE INC.

The following is a list of valid VICAR2 label keywords. Refer to the VICAR2 specification on the CD-ROM for an explanation of each keyword and the data format of its associated value:

BUFSIZE	N4
LBLSIZE	NB
DAT_TIM	NBB
DIM	NL
EOL	NLB
FORMAT	NS
HOST	ORG
INTFMT	REALFMT
N1	RECSIZE
N2	TASK
N3	USER

The first keyword of the system section of the label is always LBLSIZE, which specifies the total number of bytes in the label, including any NULL bytes used for padding out the last record. LBLSIZE is always a multiple of the file record size.

A system label may end with the keyword TASK, which marks the beginning of the history section of the label. If no history section exists, then the label ends with an ASCII NULL value, or simply after the number of characters specified by LBLSIZE.

If the EOL KEYWORD is present in the label and has a VALUE of 1, then an additional label record appears at the end of the file following the image data. Your guess is as good as ours as to the use of this End-Of-File trailing record.

Image Data

The image data begins on the next record boundary following the label. The image data contains one record per scan line in the image. The size of each record is a number of bytes equal to the number of bytes in one scan line.

The image data is represented by one of the following data types, indicated by the FORMAT keyword in the image label:

BYTE	8 bits
HALF	16 bits
FULL	32 bits

Ex.1007 / Page 950 of 1158
APPLE INC.

REAL	32 bits
DOUB	64 bits
COMPLEX	64 bits

The following is a typical VICAR2 label section, including history and user records. Note that in this example, carriage returns have been inserted for clarity; none actually exist in a real label.

```
LBLSIZE=1680 FORMAT='BYTE'  TYPE='IMAGE'  BUFSIZ=20480  DIM=3  EOL=0
RECSIZE=840  ORG='BSQ'  NL=738  NS=840  NB=1  N1=840  N2=738  N3=1
N4=0  NBB=0  NLB=0  HOST='VAX-VMS'  INTFMT='LOW'  REALFMT='VAX'
TASK='LOGMOS'  USER='HXS343'  DAT_TIM='Fri Nov  2 17:41:32 1990'
SPECSAMP=493943  PROJSAMP=4096  PROJ_L ON=350.7789496599263
PRODTYPE='F-MIDR'  PIXSIZ=75  SEAM='CORRECTED'  MAP_PROJ='SINUSOIDAL'
SEAMLOC='NO'  IMAGE='RADAR CROSS SECTION POWER'  DN_UNITS='DECI BELS'
SPECLINE=38724  LAT_UR=27.5  LAT_UL=27.5  LAT_LR=22.4102769981245
LAT_LL =22.4102769981245  LON_UL=347.5  LON_UR=354.0578993198525
LON_LL=347.6328878639 307  LON_LR=353.9250114559218
M_SPDN_1='MISSING DATA'  LOW_REP=-20.0  HI_REP=30 .0  LOW_DN=1
HI_DN=250  N_SPDN=1  SPDN_1=0  REV_STRT=460 REV_END=4 93
PRODUCT='F-MIDR.25N351;1'  TASK='COPY'  USER='DNJ345'
DAT_TIM='Sat Apr 20 12:29:24 1991'  TASK='STRETCH'  USER='DNJ345'
DAT_TIM='Sat Apr 20 12:29: 39 1991'  PARMS=' LINEAR STRETCH:
71 TO      0 AND  155 TO    255'
```

For Further Information

For further information about VICAR2, see the documentation included on the CD-ROM that accompanies this book. You can also contact:

National Aeronautics and Space Administration (NASA)
Attn: Bob Deen
Image Processing Laboratory
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
Email: *rgd059@mipl3.jpl.nasa.gov*
WWW: *http://stardust.jpl.nasa.gov/pds_home.html*

You can also obtain VICAR2 images from the FTP archive:

ftp://ames.arc.nasa.gov/pub/SPACE/VICAR/

Ex.1007 / Page 951 of 1158
APPLE INC.

NAME:	VIFF
ALSO KNOWN AS:	Khoros Visualization/Image File Format
TYPE:	Bitmap
COLORS:	Unlimited
COMPRESSION:	None, RLE
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	Any
ORIGINATOR:	Kh'oral Research
PLATFORM:	UNIX (X Window)
SUPPORTING APPLICATIONS:	Khoros
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	None

USAGE: Used by the Khoros System as its native format.

COMMENTS: The VIFF format is capable of storing any type of information generated by the Khoros System, making VIFF a very diverse and feature-rich format.

Overview

VIFF (Khoros Visualization/Image File Format) is the native file format of the Khoros System environment. Khoros is a visual programming and software development environment used to create image processing and visualization tools for commercial and scientific research. Khoros is implemented using UNIX and the X Window System.

Khoros contains a complete visual programming language, code generators, and a user interface editor. Khoros' capabilities include interactive image display; image, numerical, and signal processing and analysis functions; and 2D and 3D plotting.

Khoros is especially useful for conducting research in the areas of image and signal processing, pattern recognition, machine vision, remote sensing, and geographic information systems. Khoros is capable of converting many image

Ex.1007 / Page 952 of 1158
APPLE INC.

file formats to and from VIFF, including TIFF, TGA, FITS, PBM, XBM, DEM, DLG, MATLAB, BIG, ELAS, Sun raster, ASCII, and raw binary. This capability makes the Khoros source code distribution a prime source of image file format information (see "For Further Information" later in this article).

File Organization

All VIFF files contain a header, which is followed by either image data or one or more data (color) maps, or both. If both image and colormap data is present, the image data precedes the map data. VIFF image data need not use a colormap to be valid, and colormaps may be stored in VIFF files with only a header and no image data.

The VIFF specification uses the term "bands" to indicate color channels. An RGB image divided into its component color planes (R,G,B) is said to contain "three bands" of data. Also, multiband data is stored "by pixel" rather than "by plane." In other words, each pixel is stored as an RGB triplet, rather than three separate color planes.

Single-band VIFF images contain a single channel (or plane) of index values and a colormap. Many of the Khoros image-processing functions require that three-band images be converted into single-band pseudocolor images before they can be used.

File Details

This section describes the VIFF header, the data maps, and the location and image data in a VIFF file.

Header

The VIFF header is 1024 bytes in size and has the following format:

```
typedef struct _ViffHeader
{
 CHAR  FileId;            /* Khoros file ID value (always ABh)*/
 CHAR  FileType;          /* VIFF file ID value (always 01h) */
 CHAR  Release;           /* Release number */
 CHAR  Version;           /* Version number */
 CHAR  MachineDep;        /* Machine dependencies indicator */
 CHAR  Padding[3];        /* Structure alignment padding (always 00h)*/
 CHAR  Comment[512];      /* Image comment text */
 DWORD NumberOfRows;      /* Length of image rows in pixels */
 DWORD NumberOfColumns;   /* Length of image columns in pixels */
 DWORD LengthOfSubrow;    /* Size of any sub-rows in the image */
```


Ex.1007 / Page 953 of 1158
APPLE INC.

```
    LONG  StartX;              /* Left-most display starting position */
    LONG  StartY;              /* Upper-most display starting position */
    FLOAT XPixelSize;          /* Width of pixels in meters */
    FLOAT YPixelSize;          /* Height of pixels in meters */
    DWORD LocationType;        /* Type of pixel addressing used */
    DWORD LocationDim;         /* Number of location dimensions */
    DWORD NumberOfImages;      /* Number of images in the file */
    DWORD NumberOfBands;       /* Number of bands (color channels) */
    DWORD DataStorageType;     /* Pixel data type */
    DWORD DataEncodingScheme;/* Type of data compression used */
    DWORD MapScheme;           /* How map is to be interpreted */
    DWORD MapStorageType;      /* Map element data type */
    DWORD MapRowSize;          /* Length of map rows in pixels */
    DWORD MapColumnSize;       /* Length of map columns in pixels */
    DWORD MapSubrowSize;       /* Size of any subrows in the map */
    DWORD MapEnable;           /* Map is optional or required */
    DWORD MapsPerCycle;        /* Number of different maps present */
    DWORD ColorSpaceModel;     /* Color model used to represent image */
    DWORD ISpare1;             /* User-defined field */
    DWORD ISpare2;             /* User-defined field */
    FLOAT FSpare1;             /* User-defined field */
    FLOAT FSpare2;             /* User-defined field */
    CHAR  Reserve[404];        /* Padding */
} VIFFHEADER;
```

FileId is a magic number indicating that this is a VIFF file. This value is always ABh.

FileType is a value indicating the type of Khoros file. This value is always 01h, indicating a VIFF file.

Release indicates the release number of the *viff.h* file in which the VIFF file information structure is defined. This value is currently 01h and does not necessarily match the Khoros system release number.

Version indicates the version number of the *viff.h* file in which the VIFF file information structure is defined. This value is currently 03h and does not necessarily match the Khoros system version number.

MachineDep contains a value indicating the format of the image data and how the image data was last processed. Values currently defined for this field are:

02h IEEE and big-endian byte ordering
04h Digital Equipment Corporation/VAX byte ordering
08h NS32000 and little-endian byte ordering
0Ah Cray byte size and ordering

Padding is a 3-byte field containing the values 00h 00h 00h. This field is used only to word-align the header structure.

Ex.1007 / Page 954 of 1158
APPLE INC.

Comment is a 512-byte field typically containing ASCII plain-text information documenting the contents of a VIFF image data. This field may also be used for other purposes, but such uses are strictly user-defined and are not supported by Khoros.

NumberOfRows and NumberOfColumns indicate the size of the image data in pixels (the number of data items). If these two values are set to 00h, then no image data is present in the file, but a colormap may still exist.

LengthOfSubrow is the length in pixels of any subrows that may exist in the image data.

StartX and StartY specify the location of a sub-image within a parent image. The home coordinate (0,0) is always the upper-left corner of the image. If these values are equal to 00h, then the image is not a sub-image.

XPixelSize and YPixelSize indicate the actual size of the pixels in meters. The ratio of these two values gives the aspect ratio of the pixels in the image data.

LocationType indicates whether the image data contains implicit (01h) or explicit (02h) locations.

LocationDim has a value of 00h if the locations are stored implicitly (LocationType is 01h); in this case, there is no location data stored in the VIFF file. Location data is present in the image file if the locations are stored explicitly (LocationType is 02h); in this case, LocationDim indicates the number of dimensions represented by the location data (typically 00h, 01h, or 02h for 1-, 2-, and 3-dimensional data).

NumberOfImages contains a value equal to the number of images stored in the VIFF file.

NumberOfBands indicates the number of bands per image (or dimensions per vector). A three-band RGB image would have a value of 03h in this field.

DataStorageType specifies the data type used to store each pixel. Supported values include:

00h	Bit
01h	BYTE
02h	WORD
04h	DWORD
05h	Single-precision float
06h	Complex float

Ex.1007 / Page 955 of 1158
APPLE INC.

09h Double-precision float
0Ah Complex double

Note that when pixels are stored as bits, eight pixels are packed per BYTE, least significant bit first, and are padded out to end on a BYTE boundary.

DataEncodingScheme indicates the type of encoding used to compress the image data. Defined values for this field include:

00h No compression
01h ALZ
02h RLE
03h Transform-based
04h CCITT
05h ADPCM
06h User-defined

Only values 00h and 01h are currently supported by Khoros 1.0.

MapScheme indicates the type of mapping present in the VIFF file. Possible values are:

01h Each image band uses its own map
02h The image is displayed by cycling through two or more maps
03h All bands share the same map
04h All bands are grouped together to point into one map

The mapping scheme indicated by a MapScheme value of 04h is not supported by Khoros 1.0.

MapStorageType specifies the type of data in the map or the resulting pixel data type after the mapping has been performed. Valid values for this field are:

00h No data type
01h Unsigned CHAR
02h Short INT
04h INT
05h Single-precision float
06h Complex float
07h Double-precision float

MapRowSize and MapColumnSize indicate the width and height of the map.

Ex.1007 / Page 956 of 1158
APPLE INC.

MapSubrowSize stores the number of sub-rows in the map, if any.

MapEnable indicates whether the image data is valid if it is not mapped. Possible values for this field are:

01h Image data may be used without first mapping it
02h Map data must be applied to the image data before it can be used.

MapsPerCycle is the number of maps used to cycle the image when it is displayed. The value of this field is valid only if MapScheme is equal to a value of 02h.

ColorSpaceModel specifies the color model used to interpret the image data. The recognized values for this field are:

00h No color space model used
01h NTSC RGB
02h NTSC CMY
03h NTSC YIQ
04h HSV
05h HLS
06h IHS
07h CIE RGB
08h CIE CMY
09h CIE YIQ
0Ah CIE UCS UVW
0Bh CIE UCS SOW
0Ch CIE UCS Lab
0Dh CIE UCS Luv
0Eh User-defined
0Fh User-defined RGB

ISpare1, ISpare2, FSpare1, and FSpare2 are spare fields included for use by a user application. These fields are typically used to store data while the header is stored in memory and may not contain any information when the format is stored to a disk file.

Reserve is a chunk of padding which fills out the VIFF header to 1024 bytes in length. The bytes in this field are normally set to 00h, but may contain user-defined data as well.

Ex.1007 / Page 957 of 1158
APPLE INC.

Maps

Following the header may be a collection of one or more data maps. A data map contains information that is used to transform the image data in a specified way. By far the most common example of a map is a simple colormap, where each pixel contains an index value that references a color value stored in the map. Although VIFF data maps may contain any type of data necessary to interpret the image data properly, the image data always contains index values if a data map is present in the file.

All maps are stored as 1-dimensional data, and all map values are referenced as elements in an array. The size of the map is indicated in the MapRowSize and MapColumnSize header fields. The size of each element of the map is specified by the MapStorageType field.

Although there may be only one physical map data structure per VIFF file, the map may be divided into two more logical maps, each containing specific data. For example, 3-band image data might use three separate maps (one for each band), or it might use a single map for all three bands. Multiple maps may also be stored that are continually read in sequence as the image is displayed. Such a technique is used to render the image as an animation rather than as a still image.

Maps may also be specified as optional or forced. An optional map need not be used in the interpretation of the image data. A forced map is data that is required in order for the image data is to have any real validity. Forced maps are more common and usually indicate that the map contains color data and that the image data contains map index values.

Location and Image Data

Following any map data in a VIFF file may be a block of data called the location data. This data, if present, stores the coordinate information for each pixel in the image. Location data is a 1-dimensional array of float values and may contain one or more bands (dimensions) of location data. Each location value corresponds to a pixel in the image data. But before we can talk of pixel locations, we must understand how VIFF expresses the pixel data itself.

Pixels may be addressed as if they reside either in a 1-dimensional or in a multi-dimensional space. These addressing types are called implied locations and explicit locations, respectively. The implied location of a pixel is a reference to a pixel stored in a 1-dimensional array, but using the canonical 2-dimensional XY coordinates. For example, if we have a 256×256 image stored

Ex.1007 / Page 958 of 1158
APPLE INC.

in a 1-dimensional array and we need to reference the value of the pixel at location 100×135, we use its implicit coordinates:

```
WORD ImageWidth = 256;                /* Number of pixel in X axis */
WORD ImageLength = 256;               /* Number of pixel in Y axis */
BYTE PixelArray[ImageWidth * ImageLength];       /* 256x256 image */
BYTE PixelValue;

/* Get the value of the pixel at 100x135 */
PixelValue = PixelArray[ (100 * ImageWidth) + 135 ];
```

You can see that the formula array [(X * ImageWidth) + Y] yields the value we need. Note that this formula assumes that the pixels in the array are stored by row. If the pixel data is stored by column, then ImageLength value should be used in place of the ImageWidth.

Explicit pixel locations are used to reference pixel data stored in two or more dimensions. If explicit pixel locations are indicated, a block of location data appears in the VIFF file just prior to the image data. This location data stores the coordinates of each pixel in the rendered space. For example:

```
WORD  ImageWidth = 256;           /* Number of pixel in X axis */
WORD  ImageLength = 256;          /* Number of pixel in Y axis */
WORD  NumberOfDims = 2;           /* Dimensions of pixel locations */
BYTE  PixelArray[ImageWidth * ImageLength];    /* 256x256 image */
FLOAT LocationArray[ImageWidth * ImageLength * NumberOfDims];
BYTE  PixelValue;
FLOAT LocationX;
FLOAT LocationY;

/* Get the value of the pixel at 100x135 */
PixelValue = PixelArray[ (100 * ImageWidth) + 135 ];

/* Get the location of the pixel at 100x135 */
LocationX = LocationArray[ (100 * ImageWidth) + 135 ];
LocationY = LocationArray[ (100 * ImageWidth) + 135
          + (ImageWidth * ImageLength)];
```

As you can see, the image data is referenced in the same way for explicit data as it is for implicit data. If a map is present, the pixel values serve as indices into that map. If no map is present, then the pixel values are the actual color or intensity values of the image data.

Location data is stored as bands, but in a planar fashion. In the case of 2-dimensional data, all of the X location values are stored first, followed by all of the Y location values. And although a VIFF image can support thousands of dimensions, 1-, 2-, and 3-dimensional pixel locations are the most common.

Ex.1007 / Page 959 of 1158
APPLE INC.

For Further Information

For further information about VIFF, see the documentation and sample code included on the CD-ROM that accompanies this book.

The Khoros System is owned and copyright by Khoral Research, Inc. and is maintained by the Khoros Consortium. To obtain further information on the Khoros package and its distribution, or to support further research and development of software environments for data processing and visualization, contact:

Khoral Research, Inc.
6001 Indian School Road NE
Suite 200
Albuquerque, NM 87110
Voice: 505-837-6500
FAX: 505-881-3842
WWW: *http://www.khoros.unm.edu/khoros/*
Email: *khoros-request@chama.eece.unm.edu*

You can also contact the Khoros User Group at:

khoros@chama.eece.unm.edu

You can join the Khoros mailing list by sending email to:

khoros-request@bullwinkle.eece.unm.edu.

Khoros information may also be found posted in the USENET newsgroup *comp.soft-sys.khoros*, which is the home of the Khoros FAQ, posted weekly to this group and to *news.answers*. This and other Khoros FAQs, in addition to the complete Khoros distribution, may also be found in the */release* directory of the Khoros System distribution at the following FTP sites:

ftp://ftp.eece.unm.edu/pub/khoros
ftp://ftp.uu.net/pub/window-sys/khoros
ftp://popeye.genie.uottawa.ca/pub/khoros
ftp://ipifidpt.difi.unipi.it/pub/khoros
ftp://ftp.waseda.ac.jp/pub/khoros

Your best source of further VIFF information is the Khoros package itself. The Khoros System distribution is quite large (65 megabytes, not including executables), and it is not practical to download the entire package just to find VIFF information. Instead, you should obtain the files you need from an installed

Ex.1007 / Page 960 of 1158
APPLE INC.

Khoros System or one of the distribution sites previously listed. Information specifically about the VIFF format can be found in the following directories:

src/file_formats/no_format
Programs for converting VIFF files to and from raw binary and ASCII data

src/file_formats/remote_gis
Programs for converting VIFF files to and from remote sensing and GIS file formats.

src/file_formats/standard
Programs for converting VIFF files to and from many other file formats

include/viff.h
VIFF format header file

data/images
Sample VIFF image files

The VIFF format is also described in Chapter 1 of *Volume II, Programmer's Manual*, of the Khoros Manual Set. Both Chapter 1 and the *viff.h* file are included on the CD-ROM.

Ex.1007 / Page 961 of 1158
APPLE INC.

NAME:	VIS-5D
ALSO KNOWN AS:	Visualization-5D, McIDAS, Grid File Format
TYPE:	Vector
COLORS:	NA
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Big- and little-endian
ORIGINATOR:	University of Wisconsin - Madison
PLATFORM:	UNIX
SUPPORTING APPLICATIONS:	VIS-5D, McIDAS
SPECIFICATION ON CD:	Yes
CODE ON CD:	Yes
IMAGES ON CD:	No
SEE ALSO:	None

USAGE: Designed to support the VIS-5D application by providing for the storage of multi-dimensional data.

COMMENTS: Although VIS-5D was designed to support scientific applications, multi-dimensional data visualization is a growing area. This format may become more important outside of scientific circles in the future.

Overview

The VIS-5D (Visualization-5D) format is the native file format of the VIS-5D scientific visualization UNIX application. VIS-5D is used to store a 3D rendering of numerical data. Such data is typically acquired from scientific sources such as weather data and topographical measurements. Because the numerical data often contains a time component, VIS-5D file data may be animated to show changes over time, such as the movements of cloud patterns.

The VIS-5D application is actually a stand-alone subsystem of the proprietary McIDAS (Man-computer Interactive Data Access System) system maintained by the Space Science and Engineering Center of the University of Wisconsin at Madison. McIDAS also uses the VIS-5D format to store data, and users of both

Ex.1007 / Page 962 of 1158
APPLE INC.

systems often refer to VIS-5D files as "McIDAS 5D grid files", or simply as "grid files."

Because the VIS-5D system is distributed under the GNU public license, it is freely available with full source code. This makes the VIS-5D distribution itself your best source of information for this file format.

File Organization

All VIS-5D files contain three sections of data:

- The 5D file header
- A sequence of 3D grid information headers
- An array of 3D grid data sequences

The file header contains information about the contents of the entire file. The grid information headers are a directory of the grid data sequences stored in the file. The data is a sequence of one or more 3D grids, each of which defines a coordinate point in the numerical data.

There is one 3D grid information header per grid stored in the file. The information header stores the size and location, time and date stamp, variable name, and unit description of a grid or grid point. Each grid point is constructed of five floating-point values referred to as a five-dimensional data set. The five values stored are the three coordinate locations of the grid point, a time step value, and the physical variable name of the grid point.

The three coordinate locations store the spatial dimensions of latitude, longitude, and altitude (rows, columns, and levels) of a point in the 3D grid data set. The time step value indicates the moment in time that the point exists. The variable name is the label used to refer to the point.

Grid points are divided into logical groupings called grid sequences. A grid sequence is one or more grid points that all exist in the same moment in time. A grid sequence is and similar to a single still-image frame in an animation. Displaying the grid sequences in their orderly succession causes the numerical data to become animated.

The duration of time between each grid sequence is called a time step. The time step increments may be as short as one second or as long as years. All of the time steps in a grid data set are the same length of time, and the rate at which the data is displayed by an application may or may not depend on the size of these increments.

Ex.1007 / Page 963 of 1158
APPLE INC.

Time step values are derived from the time and date stamp data associated with each grid. Time stamps are constructed using the familiar *HHMMSS* (hours, minutes, seconds) format. For example, 183426 would be 6:34 p.m. and 26 seconds. Date stamps are constructed using the less familiar *YYDDD* format, where *YY* is the last two digits of the year and *DDD* is the number of the day of the year. For example, June 20th is the 171st day of the year 1994 and would be represented by the date format 94171.

In the following hypothetical example, we have a data set containing 20 grid points. Each point is identified by an integer starting with 1. The data set contains five grid sequences, each containing four grid points. There are five time steps, one per grid sequence, starting on April 7, 1994 at 11:00 p.m. and advancing in 30-minute increments. Each time step is composed of four grid points using the physical variable names A, B, C, and D. Each grid contains data in all three spatial dimensions:

Grid	Seq.	Date	Time	Var.	Latitude	Longitude	Altitude
1	1	94097	230000	A	34076.67	123543.90	4365.7
2	1	94097	230000	B	29086.56	135789.74	4405.6
3	1	94097	230000	C	32567.67	129086.56	4445.5
4	1	94097	230000	D	16095.34	116865.91	4495.4
5	2	94097	233000	A	34184.22	123543.90	4375.7
6	2	94097	233000	B	29006.56	135789.75	4415.6
7	2	94097	233000	C	32567.67	129186.56	4455.5
8	2	94097	233000	D	16567.34	115765.91	4505.4
9	3	94098	000000	A	34262.47	123543.90	4385.7
10	3	94098	000000	B	29008.22	135789.76	4425.6
11	3	94098	000000	C	32567.67	129286.56	4465.5
12	3	94098	000000	D	16456.34	114365.91	4515.4
13	4	94098	003000	A	34666.14	123543.90	4395.7
14	4	94098	003000	B	28055.46	135789.77	4435.6
15	4	94098	003000	C	32567.67	129386.56	4475.5
16	4	94098	003000	D	16234.34	112165.91	4525.4
17	5	94098	010000	A	34776.20	123543.90	4405.7
18	5	94098	010000	B	28006.01	135789.78	4445.6
19	5	94098	010000	C	32567.67	129486.56	4485.5
20	5	94098	010000	D	16123.34	120486.91	4535.4

As you can see, this data is animated by displaying each successive grid sequence as a frame in a movie. In this example, the four grid points are

Ex.1007 / Page 964 of 1158
APPLE INC.

displayed in sequence and would appear to move as their position changes over time.

Although it is possible to construct a simple data set consisting of one dimension, no time dynamic, and only one variable, it is more likely that data stored using the VIS-5D format will use all five dimensions.

Grid files are typically named using file mask GR3D*nnnn*, where *nnnn* is a zero-padded number in the range of 0001 to 9999. This number is referred to as the grid file number and is used to sequence and identify grid files associated with a project. Grid files typically do not use a file extension.

A single grid file may contain a maximum of 100,000,000 grid points, which is 400 megabytes of data. If this is not enough for your application, VIS-5D allows single data sets to span multiple disk files.

File Details

All VIS-5D files begin with a 256-byte header in the following format:

```
typedef struct _Vis5DHeader
{
    CHAR  Identifier[32];     /* File description field */
    LONG  ProjectNumber;      /* Project number */
    LONG  CreationDate;       /* Date file was created */
    LONG  MaximumSize;        /* No. of data points in largest 3D grid */
    LONG  NumberOfGrids;      /* No. of 3D grids in the data */
    LONG  FirstGrid;          /* Location of first grid */
    LONG  Padding[51];        /* Alignment padding */
} VIS5DHEADER;
```

Identifier is a 32-byte field used to store a NULL-terminated character string which is used to identify the file and its contents. This field, when blank, may be filled with all NULL (00h) or SPACE (20h) characters.

ProjectNumber is a user-defined value used to identify the project to which a VIS-5D file belongs. If a project number is not required, this value may then equal the grid file number used to construct the file name, or may be set to 00h.

CreationDate is the date the VIS-5D file was created in *YYDDD* format. A value of 01h indicates that no creation date was specified.

MaximumSize is the number of data points in the largest 3D grid. This value is the product of the number of rows, columns, and levels in the largest grid.

Ex.1007 / Page 965 of 1158
APPLE INC.

NumberOfGrids is the total number of 3D grids in the data. This value is the product of the number of time steps and parameters in the data.

FirstGrid is the offset location of the first grid in the data. This offset value is the number of 4-byte LONG values from the first grid to the beginning of the file. The first grid usually follows the last 3D grid information header in the file.

Padding is 204 bytes of filler used to pad the header out to a length of 256 bytes. This field is set to a value of 00h.

Following the header is a sequence of 3D grid information headers. Each header may be thought of as an entry in a directory of grid point data found in a grid file. There will be one information header per grid point stored in the file, and the number of grids is equal to the product of the number of time steps and physical parameters.

All 3D grid information headers are 256 bytes in length and have the following format:

```
typedef struct _3DGridInformationHeader
{
    LONG  Size;                /* Number of data points */
    LONG  NumberOfRows;        /* Number of rows */
    LONG  NumberOfColumns;     /* Number of columns */
    LONG  NumberOfLevels;      /* Number of levels */
    LONG  DataLocation;        /* Location of grid data */
    LONG  Date;                /* Grid date stamp */
    LONG  Time;                /* Grid time stamp */
    LONG  Padding1;            /* Alignment padding */
    CHAR  ParamName[4];        /* 4-character variable or parameter name */
    CHAR  UnitsDesc[4];        /* 4-character units description */
    LONG  Padding2[11];        /* Alignment padding */
    LONG  IType;               /* Always 04h */
    LONG  NorthLatitude;       /* North latitude * 10000 */
    LONG  WestLongitude;       /* West longitude * 10000 */
    LONG  LatitudeIncrement;   /* Latitude increment * 10000 */
    LONG  LongitudeIncrement;  /* Longitude increment * 10000 */
    LONG  Padding3[4];         /* Alignment padding */
    LONG  IhType;              /* Always 01h */
    LONG  TopAltitude;         /* Top altitude * 1000 */
    LONG  AltitudeIncrement;   /* Altitude increment * 1000 */
    LONG  Padding4[31];        /* Alignment padding */
} 3DGRIDINFOHEADER;
```

Size is equal to the number of 4-byte data points in the grid. This value is always equal to NumberOfRows * NumberOfColumns * NumberOfLevels.

Ex.1007 / Page 966 of 1158
APPLE INC.

NumberOfRows, NumberOfColumns, and NumberOfLevels are the number of rows, columns, and levels, respectively, in the grid data.

DataLocation is the location of the grid data in the file stored as the number of 4-byte LONG values from the beginning of the file.

Date is the date stamp of the grid data in *YYDDD* format.

Time is the time stamp of the grid in *HHMMSS* format.

Padding1 is four bytes of filler used to align the first seven fields of the header on a 32-byte boundary. This field is set to a value of 00h.

ParamName is a 4-character ASCII string which is the physical variable name used to represent the grid point. This field is not NULL-terminated and is padded with SPACE (20h) characters if needed.

UnitsDesc is a 4-character ASCII string which describes the unit of measure used by the grid point. This field is not NULL-terminated and is padded with SPACE (20h) characters if needed.

Padding2 is 44 bytes of filler used to align the previous ten fields. This field is set to a value of 00h.

IType is always set to the value 04h.

NorthLatitude is the northernmost latitude in the grid data multiplied by 10000.

WestLongitude is the westernmost longitude in the grid data multiplied by 10000.

LatitudeIncrement is the latitude increment multiplied by 10000.

LongitudeIncrement is the longitude increment multiplied by 10000.

Padding3 is 16 bytes of filler used to align the previous 16 fields. This field is set to a value of 00h.

IhType is always set to the value 01h.

TopAltitude is the highest altitude multiplied by 1000.

AltitudeIncrement is the altitude increment multiplied by 1000.

Padding4 is 124 bytes of filler used to pad the information header out to a length of 256 bytes. This field is set to a value of 00h.

Ex.1007 / Page 967 of 1158
APPLE INC.

The actual grid data is a five-dimensional array of floating-point values:

```
FLOAT GridArray[TimeSteps][Parameters][Levels][Columns][Rows];
```

TimeSteps is the number of time steps in this grid data set.

Parameters is the number of physical parameters used by the grid data set.

Levels is the number of levels (altitude).

Columns is the number of columns (longitude).

Rows is the number of rows (latitude).

An array containing only one grid using one time step and one variable would be declared as:

```
FLOAT GridArray[1][1][1][1][1];
```

while an array containing 1644 grids using 12 parameters and 137 time steps in each sequence would be declared as:

```
FLOAT GridArray[137][12][1644][1644][1644];
```

The Northwest-bottom corner of the data set is:

```
GridArray[TimeSteps][Parameters][0][0][0];
```

The Southeast-top corner is:

```
GridArray[TimeSteps][Parameters][Levels - 1][Columns - 1][Rows - 1];
```

Empty or missing data in a grid element is indicated by a value of 1.0e35.

For Further Information

For further information about VIS-5D, see the documentation and sample code on the CD-ROM that accompanies this book. For additional information or to be added to the VIS-5D mailing list, contact:

Space Science and Engineering Center
Attn: Bill Hibbard or Brian Paul
University of Wisconsin—Madison
1225 West Dayton Street
Madison, WI 53706
Bill Hibbard: *whibbard@macc.wisc.edu*
Brian Paul: *brianp@ssec.wisc.edu*
WWW: *http://java.meteor.wisc.edu/*

Ex.1007 / Page 968 of 1158
APPLE INC.

VIS-5D (cont'd)

The VIS-5D distribution is available via FTP at the following site:

ftp://iris.ssec.wisc.edu/pub/vis5d/

There is no official VIS-5D file format specification, but the files *README* and *util/sample.c* (included on the CD-ROM) should be of special interest.

Ex.1007 / Page 969 of 1158
APPLE INC.

NAME:	Vivid, Bob
ALSO KNOWN AS:	None
TYPE:	Scene description
COLORS:	NA
COMPRESSION:	No
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	NA
NUMERICAL FORMAT:	NA
ORIGINATOR:	Stephen Coy
PLATFORM:	All
SUPPORTING APPLICATIONS:	Vivid and Bob ray tracers, other ray-trace applications
SPECIFICATION ON CD:	No
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	DKB, NFF, P3D, POV, PRT, QRT, Radiance, Rayshade

USAGE: Vivid has been widely distributed on the Internet.

COMMENTS: You might want to look at these if you're thinking about writing a ray tracer, mainly because the code is available.

Overview

Vivid and Bob are two separate ray-trace applications originated and maintained by their author, Stephen Coy. They both use proprietary scene-input formats designed by the author. We are not able to include file format specifications for Vivid and Bob on the CD-ROM that accompanies this book, although we would like to, because these formats have had a substantial impact on other ray trace formats. About Vivid, Mr. Coy writes:

> Vivid's file format is constantly changing, so that anything I could give you would be out-of-date even before you could publish it. In the last year since version 2.0 came out, I've released 18 new test versions

Ex.1007 / Page 970 of 1158
APPLE INC.

> covering bug fixes and new features, with several hundred lines of notes and changes to the documentation. Vivid 3.0 will be released within the next couple of months, but until that time there's no single document fully describing the file format as it currently is.

About Bob, he writes:

> The Bob file formats are documented in the book *Photorealism and Ray Tracing in C*, which anyone who is legally using Bob already has.

File Organization and Details

We've pulled together what information we have been able to obtain about the Vivid and Bob formats here.

The Vivid input module is case-sensitive, and the system understands a right-hand coordinate system. Colors are expressed as RGB triplets, and each of R, G, and B falls into the range 0 to 1. Certain mathematical operations are available, and are designed for use with both vectors and numerical values. Vector operations include the following:

Addition
Subtraction
Scalar multiplication
Dot and cross products

Scalar operations include the following:

Addition
Subtraction
Multiplication
Division
Exponentiation

The system also supports the following functions:

Sine
Cosine
Tangent
Arcsine

Ex.1007 / Page 971 of 1158
APPLE INC.

Arccosin
Arctangent
Square roots

Operator precedence appears to be poorly developed, so liberal use of parentheses in complex expressions is recommended.

Comments are the same as in ANSI C, which means that both

```
/* a comment */
```

and

```
// a comment
```

are supported.

Normal files contain what the documentation refers to as a studio description, which consists of the image size, antialiasing state, and viewpoint. This description is followed by definitions of lights, surfaces, and objects. Lights and surface definitions persist until redefined. Objects, as in most systems of this type, are simply geometric descriptions.

The Vivid system is implemented partly through the use of a preprocessor. Some versions of Vivid leave a temporary file, *XYZZY.V* subsequent to invocation. This appears to be an error in the system, but it may have some utility in debugging. Macros are contained between normal parentheses, and multiline macros, as in the C preprocessor, must have each line terminated with the line continuation character (\).

For Further Information

For further information about the Vivid and Bob file formats, refer to the documentation that comes with the applications. In particular, for Bob, refer to the following book that comes with the system:

Photorealism and Ray Tracing in C

For information about obtaining these applications, contact the author:

Stephen Coy
Email: *coy@plato.ds.boeing.com*

Ex.1007 / Page 972 of 1158
APPLE INC.

Wavefront OBJ

NAME:	Wavefront OBJ*
ALSO KNOWN AS:	Wavefront Object, OBJ
TYPE:	3D Vector
COLORS:	Unlimited
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	NA
ORIGINATOR:	Wavefront
PLATFORM:	UNIX
SUPPORTING APPLICATIONS:	Advanced Visualizer
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	No
SEE ALSO:	Wavefront RLA

USAGE: Used to store and exchange 3D data.

COMMENTS: The Wavefront OBJ format is a useful standard for representing polygonal data in ASCII form.

Overview

Wavefront OBJ (object) files are used by Wavefront's Advanced Visualizer application to store geometric objects composed of lines, polygons, and free-form curves and surfaces. Wavefront is best known for its high-end computer graphics tools, including modeling, animation, and image compositing tools. These programs run on powerful workstations such as those made by Silicon Graphics, Inc.

Wavefront OBJ files are often stored with the extension ".obj" following the UNIX convention of lowercase letters for filenames. The most recently documented version of OBJ is v3.0, superseding the previous v2.11 release.

* Our thanks to John Foust for his contributions to this article.

Ex.1007 / Page 973 of 1158
APPLE INC.

In Wavefront's 3D software, geometric object files may be stored in ASCII format (using the ".obj" file extension) or in binary format (using the .MOD extension). The binary format is proprietary and undocumented, so only the ASCII format is described here.

The OBJ file format supports lines, polygons, and free-form curves and surfaces. Lines and polygons are described in terms of their points, while curves and surfaces are defined with control points and other information depending on the type of curve. The format supports rational and non-rational curves, including those based on Bezier, B-spline, Cardinal (Catmull-Rom splines), and Taylor equations.

File Organization

OBJ files do not require any sort of header, although it is common to begin the file with a comment line of some kind. Comment lines begin with a hash mark (#). Blank space and blank lines can be freely added to the file to aid in formatting and readability. Each non-blank line begins with a keyword and may be followed on the same line with the data for that keyword. Lines are read and processed until the end of the file. Lines can be logically joined with the line continuation character (\) at the end of a line.

The following keywords may be included in an OBJ file. In this list, keywords are arranged by data type, and each is followed by a brief description.

Vertex data:

v	Geometric vertices
vt	Texture vertices
vn	Vertex normals
vp	Parameter space vertices

Free-form curve/surface attributes:

deg	Degree
bmat	Basis matrix
step	Step size
cstype	Curve or surface type

Ex.1007 / Page 974 of 1158
APPLE INC.

Elements:

p	Point
l	Line
f	Face
curv	Curve
curv2	2D curve
surf	Surface

Free-form curve/surface body statements:

parm	Parameter values
trim	Outer trimming loop
hole	Inner trimming loop
scrv	Special curve
sp	Special point
end	End statement

Connectivity between free-form surfaces:

con	Connect

Grouping:

g	Group name
s	Smoothing group
mg	Merging group
o	Object name

Display/render attributes:

bevel	Bevel interpolation
c_interp	Color interpolation
d_interp	Dissolve interpolation
lod	Level of detail
usemtl	Material name
mtllib	Material library
shadow_obj	Shadow casting
trace_obj	Ray tracing
ctech	Curve approximation technique
stech	Surface approximation technique

Ex.1007 / Page 975 of 1158
APPLE INC.

File Details

The most commonly encountered OBJ files contain only polygonal faces. To describe a polygon, the file first describes each point with the "v" keyword, then describes the face with the "f" keyword. The line of a face command contains the enumerations of the points in the face, as 1-based indices into the list of points, in the order they occurred in the file. For example, the following describes a simple triangle:

```
# Simple Wavefront file
v 0.0 0.0 0.0
v 0.0 1.0 0.0
v 1.0 0.0 0.0
f 1 2 3
```

It is also possible to reference points using negative indices. This makes it easy to describe the points in a face, then the face, without the need to store a large list of points and their indexes. In this way, "v" commands and "f" commands can be interspersed.

```
v -0.500000 0.000000 0.400000
v -0.500000 0.000000 -0.800000
v -0.500000 1.000000 -0.800000
v -0.500000 1.000000 0.400000
f -4 -3 -2 -1
```

OBJ files do not contain color definitions for faces, although they can reference materials that are stored in a separate material library file. The material library can be loaded using the "mtllib" keyword. The material library contains the definitions for the RGB values for the material's diffuse, ambient, and specular colors, along with other characteristics such as specularity, refraction, transparency, etc.

The OBJ file references materials by name with the "usemtl" keyword. All faces that follow are given the attributes of this material until the next "usemtl" command is encountered.

Faces and surfaces can be assigned into named groups with the "g" keyword. This is used to create convenient sub-objects to make it easier to edit and animate 3D models. Faces can belong to more than one group.

The following demonstrate more complicated examples of smooth surfaces of different types, material assignment, line continuation, and grouping.

Ex.1007 / Page 976 of 1158
APPLE INC.

Cube with Materials

```
# This cube has a different material
# applied to each of its faces.

    mtllib master.mtl

    v   0.000000        2.000000        2.000000
    v   0.000000        0.000000        2.000000
    v   2.000000        0.000000        2.000000
    v   2.000000        2.000000        2.000000
    v   0.000000        2.000000        0.000000
    v   0.000000        0.000000        0.000000
    v   2.000000        0.000000        0.000000
    v   2.000000        2.000000        0.000000
    # 8 vertices

    g front
    usemtl red
    f 1 2 3 4
    g back
    usemtl blue
    f 8 7 6 5
    g right
    usemtl green
    f 4 3 7 8
    g top
    usemtl gold
    f 5 1 4 8
    g left
    usemtl orange
    f 5 6 2 1
    g bottom
    usemtl purple
    f 2 6 7 3
    # 6 elements
```

Bezier Patch

```
# 3.0 Bezier patch
v -5.000000 -5.000000 0.000000
v -5.000000 -1.666667 0.000000
v -5.000000 1.666667 0.000000
v -5.000000 5.000000 0.000000
v -1.666667 -5.000000 0.000000
v -1.666667 -1.666667 0.000000
v -1.666667 1.666667 0.000000
v -1.666667 5.000000 0.000000
v 1.666667 -5.000000 0.000000
v 1.666667 -1.666667 0.000000
v 1.666667 1.666667 0.000000
```


Ex.1007 / Page 977 of 1158
APPLE INC.

```
v 1.666667 5.000000 0.000000
v 5.000000 -5.000000 0.000000
v 5.000000 -1.666667 0.000000
v 5.000000 1.666667 0.000000
v 5.000000 5.000000 0.000000

# 16 vertices

cstype bezier
deg 3 3
# Example of line continuation
surf 0.000000 1.000000 0.000000 1.000000 13 14 \
                15 16 9 10 11 12 5 6 7 8 1 2 3 4
parm u 0.000000 1.000000
parm v 0.000000 1.000000
end
# 1 element
```

Cardinal Curve

```
# 3.0 Cardinal curve

v 0.940000 1.340000 0.000000
v -0.670000 0.820000 0.000000
v -0.770000 -0.940000 0.000000
v 1.030000 -1.350000 0.000000
v 3.070000 -1.310000 0.000000
# 6 vertices

cstype cardinal
deg 3
curv 0.000000 3.000000 1 2 3 4 5 6
parm u 0.000000 1.000000 2.000000 3.000000 end
# 1 element
```

Texture-Mapped Square

```
# A 2 x 2 square mapped with a 1 x 1 square
# texture stretched to fit the square exactly.

mtllib master.mtl

v  0.000000 2.000000 0.000000
v  0.000000 0.000000 0.000000
v  2.000000 0.000000 0.000000
v  2.000000 2.000000 0.000000
vt 0.000000 1.000000 0.000000
vt 0.000000 0.000000 0.000000
vt 1.000000 0.000000 0.000000
vt 1.000000 1.000000 0.000000
# 4 vertices
```


Ex.1007 / Page 978 of 1158
APPLE INC.

```
usemtl wood

# The first number is the point,
# then the slash,
# and the second is the texture point
f 1/1 2/2 3/3 4/4
# 1 element
```

For Further Information

For further information about the Wavefront OBJ format, see the specification included on the CD-ROM that accompanies this book.

You can also contact:

Wavefront Technologies
530 East Montecito Street
Santa Barbara, CA 93103
Voice: 805-962-8117
FAX: 805-963-0410
WWW: *http://www.aw.sgi.com/*

Wavefront also maintains a toll-free support number and a BBS for its customers. There are many Wavefront user groups, too.

Ex.1007 / Page 979 of 1158
APPLE INC.

NAME:	Wavefront RLA
ALSO KNOWN AS:	RLA, RLB, Run-length Encoded Version A, Run-length Encoded Version B
TYPE:	3D bitmap
COLORS:	16 million
COMPRESSION:	RLE
MAXIMUM IMAGE SIZE:	64K×64K pixels
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	Wavefront
PLATFORM:	UNIX
SUPPORTING APPLICATIONS:	Wavefront Advanced Visualizer
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	Ys
SEE ALSO:	None

USAGE: Storage of 3D output data.

COMMENTS: Proprietary, but associated with a well-known application, so likely to be important in the future.

Overview

The Wavefront RLA (Run-Length Encoded Version A) image file format is used to store three types of data:

- Graphics images
- Field-rendered images captured from live video
- Three-dimensionally rendered image data

RLA is used primarily by the Wavefront Advanced Visualizer animation package to store output data and to exchange graphical data with other software applications. There are actually three variations of the RLA image file format.

Ex.1007 / Page 980 of 1158
APPLE INC.

Prior to 1990, the original RLA format was in use. In this format, RLA was capable of storing only standard graphics images, such as those found in many other bitmap file formats.

In 1990, the capabilities of RLA expanded to include the storage of field-rendered images. Rather than revising RLA (and upsetting many customers in the process), Wavefront created the RLB (Run-Length Encoded Version B) image file format to incorporate these new features. RLB is essentially the original RLA format with a few extra fields added to the header.

Some time later, Wavefront released version 3.0 of the Advanced Visualizer animation package and updated the original RLA format to include all of the new fields found in the RLB format. Wavefront also added and expanded several fields in the header to include the capability of storing multichannel rendered image data—a feature not supported by the RLB format. Thus, the new RLA image file format was born.

Today, the new RLA image file format is the standard format for Wavefront software applications. Both RLB and the original RLA format are considered to be outdated, and future support of these two older formats by Wavefront is questionable.

File Organization

Both RLA formats and the RLB format are always stored using the big-endian byte order. Floating-point data is always stored as ASCII strings to avoid problems with machine-dependent representations of floating-point data. String data stored in the header is composed entirely of ASCII data and is always NULL-terminated. Blank character fields contain all NULL (ASCII 00h) values.

All three formats contain three major sections:

- Header
- Scan-line offset table
- Image data

File Details

This section contains information about each of the components in an RLA or RLB file.

Ex.1007 / Page 981 of 1158
APPLE INC.

Header

The RLA header is 740 bytes in length and has the following format:

```
typedef struct _WavefrontHeader
{
  SHORT WindowLeft;          /* Left side of the full image */
  SHORT WindowRight;         /* Right side of the full image */
  SHORT WindowBottom;        /* Bottom of the full image */
  SHORT WindowTop;           /* Top of the full image */
  SHORT ActiveLeft;          /* Left side of the viewable image */
  SHORT ActiveRight;         /* Right side of viewable image */
  SHORT ActiveBottom;        /* Bottom of the viewable image */
  SHORT ActiveTop;           /* Top of the viewable image */
  SHORT FrameNumber;         /* Frame sequence number */
  SHORT ColorChannelType;    /* Data format of the image channels */
  SHORT NumOfColorChannels;  /* Number of color channels in image */
  SHORT NumOfMatteChannels;  /* Number of matte channels in image */
  SHORT NumOfAuxChannels;    /* Number of auxiliary channels in image */
  SHORT Revision;            /* File format revision number */
  CHAR  Gamma[16];           /* Gamma setting of image */
  CHAR  RedChroma[24];       /* Red chromaticity */
  CHAR  GreenChroma[24];     /* Green chromaticity */
  CHAR  BlueChroma[24];      /* Blue chromaticity */
  CHAR  WhitePoint[24];      /* White point chromaticity*/
  LONG  JobNumber;           /* Job number ID of the file */
  CHAR  FileName[128];       /* Image file name */
  CHAR  Description[128];    /* Description of the file contents */
  CHAR  ProgramName[64];     /* Name of the program that created the file */
  CHAR  MachineName[32];     /* Name of machine used to create the file */
  CHAR  UserName[32];        /* Name of user who created the file */
  CHAR  DateCreated[20];     /* Date the file was created */
  CHAR  Aspect[24];          /* Aspect format of the image */
  CHAR  AspectRatio[8];      /* Aspect ratio of the image */
  CHAR  ColorChannel[32];    /* Format of color channel data */
  SHORT Field;               /* Image contains field-rendered data */
  CHAR  Time[12];            /* Length of time used to create the image
                                file */
  CHAR  Filter[32];          /* Name of post-processing filter */
  SHORT NumOfChannelBits;    /* Number of bits in each color channel pixel */
  SHORT MatteChannelType;    /* Data format of the matte channels */
  SHORT NumOfMatteBits;      /* Number of bits in each matte channel pixel */
  SHORT AuxChannelType;      /* Data format of the auxiliary channels */
  SHORT NumOfAuxBits;        /* Number of bits in each auxiliary channel
                                pixel */
  CHAR  AuxData[32];         /* Auxiliary channel data description */
  CHAR  Reserved[36];        /* Unused */
  LONG  NextOffset;          /* Location of the next image header in the
                                file */
} WAVEFRONT;
```


Ex.1007 / Page 982 of 1158
APPLE INC.

WindowLeft, WindowRight, WindowBottom, and WindowTop specify the absolute size of the image in pixels. The home position of the window is normally the bottom-left corner of the display. An image displayed starting from this position has WindowLeft and WindowBottom values of 0.

ActiveLeft, ActiveRight, ActiveBottom, and ActiveTop define the part of the image that is actually stored in the file. Normally, these values are the same as the WindowLeft, WindowRight, WindowBottom, and WindowTop values. However, if the stored image is a clip from a larger image, these values indicate the position of the stored image on the original image. Only the clipped portion of the original image is stored in the file.

The size of the image may be determined using the following calculations:

```
ImageHeight = (ActiveBottom - ActiveTop) + 1;
ImageWidth = (ActiveRight - ActiveLeft) + 1;
```

FrameNumber is the image number if the image is one frame in a sequence. Values for this field start at 01h.

ColorChannelType indicates the format of the color-channel data. A value of 0 indicates that the data is stored as 8-bit BYTEs. A value of 1 indicates that the data is stored as 16-bit WORDs. A value of 2 indicates that the data is stored as 32-bit DWORDs. And a value of 3 indicates that 32-bit IEEE floats are used to store the data.

NumOfColorChannels specifies the number of color channels in the file. There are typically three color channels (red, green, and blue) in an image.

NumOfMatteChannels specifies the number of matte information channels in the file. There is typically only one matte channel, or possibly one matte channel per color channel.

NumOfAuxChannels specifies the number of auxiliary information channels in the file. There are normally no auxiliary channels (unless the file contains 3D image data), and this field is set to 00h.

Revision holds the current revision identifier for the image format. This value is always FFFEh. This field holds an auxiliary data-mask value in original RLA format images.

Gamma contains an ASCII floating-point number representing the gamma correction factor applied to the image before it was stored. A value of 2.2 is considered typical. A value of 0.0 indicates no gamma setting.

Ex.1007 / Page 983 of 1158
APPLE INC.

RedChroma, GreenChroma, BlueChroma, and WhitePoint specify the X and Y chromaticity values for the red, green, and blue primary colors and the white-point reference value. These values are written as ASCII floating-point numbers and have the standard NTSC chromaticity values as their default:

	X	Y
Red	0.670	0.330
Green	0.210	0.710
Blue	0.140	0.080
White	0.310	0.316

JobNumber is a user-defined number that identifies the project or task of which the image is part.

FileName is the name of the image file which stores the data. A maximum of 128 characters may be stored in this field.

Description is a string describing the contents of the image file. A maximum of 128 characters may be stored in this field.

ProgramName is the name of the software program that created the file. A maximum of 64 characters may be stored in this field.

MachineName is the name of the computer system that created the image file. A maximum of 32 characters may be stored in this field.

UserName is the name of the user or system account that created the image file. A maximum of 32 characters may be stored in this field.

DateCreated is the date that the image file was created. A maximum of 20 characters may be stored in this field. Wavefront images typically use the date format *MMM DD hh:mm yyyy* (e.g., SEP 17 16:30 1994).

Aspect is a user-defined string describing the aspect ratio of the image. This string is used to locate size and aspect ratio information stored in a table. A maximum of 24 characters may be stored in this field. Following is a list of aspect description strings defined by Wavefront:

Description	Width	Height	Aspect
1k_square	1024	1024	1.00
2k_square	2048	2048	1.00
3k_square	3072	3072	1.00

Ex.1007 / Page 984 of 1158
APPLE INC.

Wavefront RLA (cont'd)

Description	Width	Height	Aspect
4k_square	4096	4096	1.00
5k_square	5120	5120	1.00
6k_square	6144	6144	1.00
7k_square	7168	7168	1.00
8k_square	8192	8192	1.00
Creator	640	484	1.33
ImageNode	512	486	1.33
Iris-8-10	3000	2400	1.25
Abekas	720	486	1.33
ccir_pal	720	576	1.33
full_1024	1024	768	1.33
full_1280	1280	1024	1.33
iris_1400	1022	768	1.33
iris_2400	1024	768	1.33
iris_ntsc	636	484	1.33
iris_pal	768	576	1.33
matrix_2k	2048	1366	1.50
matrix_2k_ntsc	1821	1366	1.33
matrix_4k	4096	2732	1.50
matrix_4k_ntsc	3642	2732	1.33
ntsc_4d	646	485	1.33
ntsc_512	512	484	1.33
ntsc_512_fld	512	242	1.33
ntsc_636	636	484	1.33
ntsc_636_fld	636	242	1.33
ntsc_640	640	486	1.33
ntsc_640_fld	640	243	1.33
pal_768	768	576	1.33
pal_780	780	576	1.33
pixar	1024	768	1.33
pixar_ntsc	640	486	1.33
pv_2k	2048	1638	1.25
pv_3k	3072	2457	1.25
pv_ntsc	646	486	1.33
pv_pal	768	576	1.33
qnt_pal	720	576	1.33
qtl_ntsc	720	486	1.33

Ex.1007 / Page 985 of 1158
APPLE INC.

Description	Width	Height	Aspect
screen	1280	1024	1.25
shiba_soku	1600	1045	1.33
sony_hdtv	1920	1035	1.855
tek_ntsc	720	486	1.33
tek_pal	720	576	1.33
texture_512	512	512	1.00
tga_486	512	486	1.33
tga_ntsc	512	482	1.33
vc_ntsc	640	486	1.33
vfr_comp	768	486	1.33
vfr_rgb	720	486	1.33
vst_hires	1024	768	1.33
vst_ntsc	756	486	1.33
vst_pal	738	576	1.33
vst_pal2	740	578	1.33
vst_targa	512	486	1.33
vtc_mvbhm	1440	1200	1.33
vte_720	720	576	1.33
window	1024	820	1.25

AspectRatio is an ASCII floating-point number used to determine the pixel aspect ratio of the image. This number is the display width divided by the display height.

ColorChannel is an ASCII string identifying the color space model used to represent the image data. Values for this field may be rgb, xyz, or sampled.

Field is set to 01h if the file contains a field-rendered image. Otherwise, the value of this field is 00h.

Time is a string storing the amount of CPU time in seconds that was required to create the image. A maximum of 12 characters may be stored in this field.

Filter is the name of the filter used to post-process the image data before storage. A maximum of 32 characters may be stored in this field.

NumOfChannelBits specifies the number of bits per pixel in each color channel. The value for this field may be in the range of 1 to 32.

Ex.1007 / Page 986 of 1158
APPLE INC.

MatteChannelType indicates the format of the matte channel data. A value of 0 indicates that the data is stored as 8-bit BYTEs. A value of 1 indicates that the data is stored as 16-bit WORDs. A value of 2 indicates that the data is stored as 32-bit DWORDs. And a value of 3 indicates that the data is stored as 32-bit IEEE floats.

NumOfMatteBits specifies the number of bits per pixel in each matte channel. The value for this field may be in the range of 1 to 32.

AuxChannelType indicates the format of the auxiliary channel data. A value of 0 indicates that the data is stored as 8-bit BYTEs. A value of 1 indicates that the data is stored as 16-bit WORDs. A value of 2 indicates that the data is stored as 32-bit DWORDs. And a value of 3 that indicates that the data is stored as 32-bit IEEE floats.

NumOfAuxBits specifies the number of bits per pixel in the auxiliary channel. The value for this field may be in the range 1 to 32.

AuxData indicates the format of the auxiliary channel data. Valid strings for this field are range and depth.

Reserved is unused space that is reserved for future header fields. All bytes in this field are always set to 00h.

NextOffset is the offset value to the header of the next image stored in the file. The value of this field is 00h if no other images appear in the file.

Scan-Line Offset Table

Immediately following the header is a scan-line offset table. This table is a 1D array of 4-byte integers indicating the starting position of each scan line in the image data.

Each scan line in an RLA and RLB image file is run-length encoded. Because of the variable lengths of the RLE records and packets, it is impossible to detect easily where any scan line begins unless the image is decoded from the beginning.

To locate easily any scan line in the image data, store the offsets to the beginning of each encoded scan line in the scan-line offset table. Each entry in the offset table is four bytes in size, and there is one entry per scan line in the image. All offsets are calculated from the beginning of the image file, even if a file contains multiple images.

Ex.1007 / Page 987 of 1158
APPLE INC.

Image Data

The image data in RLA and RLB files is separated into one or more color channels (also called *color planes*) and one or more matte channels (also called *alpha channels*). An image using the RGB color model contains three color channels, one each for red, green, and blue color information.

Typically, there is only one matte channel per image, although one matte channel per color channel is possible. The matte channel contains information on the visual appearance of each pixel in the image and indicates the degree of transparency or opaqueness of each pixel when the image is displayed.

The channel information is organized into scan lines. Each scan line of image data therefore contains four (or more) channels of information in the order of red, green, blue, and matte. When a scan line is read, all of the red information is read first, followed by the green information, then blue, and so on.

Multiple images may be stored in a single disk file by simply appending them together. The NextOffset value in the header of one image should contain the offset value of the first byte of the header of the next image. The last image in a file will have a NextOffset value of 00h. This is common way to store a full-sized image and a postage-stamp image (called a *swatch* in Wavefront lingo) in the same file.

Field-Rendered Images

The RLB and the newer RLA format have the capability of storing field-rendered image data. Normally, an image bitmap contains both odd- and even-numbered scan lines. Interlaced video signals, such as those used by television, display frames of video data as alternating fields of odd- and even-numbered scan lines. There are always two fields per frame.

When an interlaced video field is captured and stored as an image, only half of the scan lines in the frame are present in the field. It is therefore possible to store a captured video frame as two separate fields by using two RLA or RLB image files. Odd-numbered scan lines are stored in one file, and even-numbered scan lines are stored in the other. Images stored in this way usually have file names with odd and even numbers to indicate frame and field designations.

Note that in field-rendered images, the header values ActiveBottom and ActiveTop indicates the full size of the frame image. The actual number of scan lines stored in each field-image file is half the difference between these values. For example, a 640-line frame creates two 320-line field-image files.

Ex.1007 / Page 988 of 1158
APPLE INC.

3D Image Data

New RLA images normally contain four channels of information in the form of red, green, blue, and matte data. If a 3-dimensional scene has been rendered to an image, a fifth channel, known as the auxiliary channel, is present and contains information on the depth of each pixel relative to the camera's, or viewer's, location. Auxiliary channel information is stored as floating-point data in the range 0.00 to 1.00 inclusive.

Run-Length Encoding

Each channel of image data in RLA and RLB image files is always run-length encoded (RLE). Each channel within a scan line is encoded into a separate RLE record. Each record begins with a 2-byte value indicating the number of bytes of encoded data in the record. This count byte is followed by the encoded channel data itself. There may be a maximum of 65,535 bytes of encoded data in any record.

Image data with a pixel depth of one byte (one to eight bits) is encoded into packets containing a run-count byte and a run-value byte. If the run-count value is a positive value, then the run value is repeated "run count + 1" times. If the run-count value is negative, the following "run count" bytes are repeated literally. Only runs of three or more pixels are encoded into repeated runs.

Image data with a pixel depth of two bytes (nine to 16 bits) is encoded using a similar algorithm, but the actual bytes of pixel data are read in an interleaved fashion. Two separate passes are made over the pixel data in each channel. The first pass run-length encodes the least significant byte of each pixel in the channel; the second pass encodes the most significant byte of each pixel in the channel.

With image data that contains four bytes (17 to 32 bits) per pixel, a 4-pass process is used, encoding from the least to most significant byte in each pixel. The same algorithm is used for encoding each pass of 2- and 4-byte pixel data as is used for 1-byte pixel data.

For Further Information

For further information about the Wavefront RLA and RLB formats, see the specification included on the CD-ROM that accompanies this book.

Ex.1007 / Page 989 of 1158
APPLE INC.

The specification for the RLA and RLB file formats is also available in Appendix B of the *Wavefront Advanced Visualizer User Manuals* available from Wavefront:

Wavefront Technologies
530 East Montecito Street
Santa Barbara, CA 93103
Voice: 805-962-8117
FAX: 805-963-0410
WWW: *http://www.aw.sgi.com/*

Wavefront also maintains a toll-free support number and a BBS for its customers. Many Wavefront users' groups exist to support customers.

Ex.1007 / Page 990 of 1158
APPLE INC.

WordPerfect Graphics Metafile

NAME:	WordPerfect Graphics Metafile
ALSO KNOWN AS:	WPG
TYPE:	Metafile
COLORS:	256
COMPRESSION:	RLE
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	Big-endian
ORIGINATOR:	WordPerfect Corporation
PLATFORM:	MS-DOS, Microsoft Windows, Macintosh, UNIX
SUPPORTING APPLICATIONS:	WordPerfect, other word processing programs
SPECIFICATION ON CD:	No
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	None

USAGE: Used for storage of document and image data.

COMMENTS: WPG is supported by other applications mainly for compatibility, due to the widespread distribution of WordPerfect for MS-DOS, which is the number one word processing application on that platform in terms of unit sales. Not used much as an interchange format.

Overview

The WordPerfect Graphics Metafile (WPG) file format is a creation of WordPerfect Corporation (WPC) specifically for use with its line of software products. WPG image files are likely to be found in any environment that is supported by WPC products, including MS-DOS, UNIX, and the Apple Macintosh.

WPG files are capable of storing both bitmap and vector data, which may contain up to 256 individual colors chosen from a palette of more than one million total colors. It is also possible to store Encapsulated PostScript (EPS) code in a WPG file.

Ex.1007 / Page 991 of 1158
APPLE INC.

The particular version described in this article is the WordPerfect Graphic file format as created by the WPC products WordPerfect 5.*x* and DrawPerfect 1.*x*. For a complete description of the WPG format, refer to the WordPerfect Corporation Developer's Toolkit for IBM PC Products. Information on how to obtain this toolkit is provided in the "For Further Information" section later in this article.

A WPG-format file created using WordPerfect 5.0 can store either bitmap or vector image data, but not both at once. WPG files created under WordPerfect 5.1 and later can store both bitmap and vector image data in the same file. Unfortunately, there is no way to tell whether a WPG file contains both bitmap and vector data by reading the header. The actual record data from the body of the file must be read and interpreted.

File Organization

In WPC terminology, a WordPerfect Graphics Metafile contains a prefix area (the header) and a record area (the graphics data). All data in the metafile is written using the big-endian byte order.

File Details

This section contains information about the prefix and record areas of a WordPerfect Graphics Metafile.

Prefix

The prefix is 16 bytes in length and has the following format:

```
typedef struct _WordPerfectGraphic
{
  BYTE  FileId[4];     /* File Id Code (always FFh 57h 50h 43h) */
  DWORD DataOffset;    /* Stat of data in the WPG file (always 10h)*/
  BYTE  ProductType;   /* Product Code (always 1) */
  BYTE  FileType;      /* WPC File Code (always 16h) */
  BYTE  MajorVersion; /* Major Version Code (always 1) */
  BYTE  MinorVersion; /* Minor Version Code (always 0) */
  WORD  EncryptionKey;/* Password Checksum (0 = not encrypted) */
  WORD  Reserved;      /* Reserved field (always 0) */

} WPGHEAD;
```

FileId values are four contiguous bytes that contain the standard WPC File ID code. All WPC files starting with those created by WordPerfect 5.0 begin with this code. The values for these fields, in order, are FFh, 57h, 50h, and 43h.

Ex.1007 / Page 992 of 1158
APPLE INC.

DataOffset contains an offset value pointing to the start of the record data in the WordPerfect Graphics Metafile. Because the record data always immediately follows the prefix, and the prefix is always 16 bytes in length, this value is always 10h.

ProductType identifies the WPC software product that created the WPG file. This field always contains the value 01h, indicating that the file was created by the WordPerfect word processor. This value is always the same, even if the WPG file was created by a third-party software application.

FileType identifies the type of data the file contains. For WPG files, the value of this field is always 16h.

MajorVersion and MinorVersion contain the internal version number of the product for which the WPG file was created (which may not match the published, external version number of the product). For all WPG files, the MajorVersion field always contains a value of 01h, and the MinorVersion field always contains a value of 00h.

EncryptionKey normally contains a value of 00h if the file is not encrypted. If the value of this field is non-zero, then the value is used as the checksum of the password and is used to decrypt the file. In the current version, WPG files are never encrypted and therefore the value of this field is always 00h.

Reserved is not currently used and always contains a value of 00h.

Record Area

Following the prefix in a WordPerfect Graphics Metafile is the record area. This area contains a sequence of objects and their attributes; this information is used to render the image. Any colormaps, bitmaps, and sections of PostScript code are also considered objects within the WPG file record area.

Record prefix

Each record begins with a record prefix (a header in almost any other format). The record prefix may be two, four, or six bytes in length depending on the type of record it precedes. Here are the three possible record prefix formats:

```
/* Two-byte prefix */
typedef struct _TwoByteRecPrefix
{
  BYTE  RecordType;      /* The Record Type identifier */
  BYTE  RecordLength;    /* The length of the record in bytes (0-FEh)*/

} RECPREFIX2BYTE;
```


Ex.1007 / Page 993 of 1158
APPLE INC.

```
/* Four-byte prefix */
typedef struct _FourByteRecPrefix
{
  BYTE  RecordType;      /* The Record Type identifier */
  BYTE  SizeIndicator;   /* WORD or DWORD length follows (always FFh)*/
  WORD  RecordLength;    /* The length of the record in bytes */

} RECPREFIX4BYTE;

/* Six-byte prefix */
typedef struct _SixByteRecPrefix
{
  BYTE  RecordType;      /* The Record Type identifier */
  BYTE  SizeIndicator;   /* WORD or DWORD length follows (always FFh)*/
  DWORD RecordLength;    /* The length of the record in bytes */

} RECPREFIX6BYTE;
```

Record type

RecordType, the first field of each record, contains a value that identifies the type of data stored in the record as follows:

Record Type	Record Description
01h	Fill attributes
02h	Line attributes
03h	Marker attributes
04h	Polymarker
05h	Line
06h	Polyline
07h	Rectangle
08h	Polygon
09h	Ellipse
0Ah	Reserved
0Bh	Bitmap (Type 1)
0Ch	Graphics text (Type 1)
0Dh	Graphics text attributes
0Eh	Color map
0Fh	Start of WPG data (Type 1)
10h	End of WPG data
11h	PostScript data follows (Type 1)
12h	Output attributes
13h	Curved polyline

Ex.1007 / Page 994 of 1158
APPLE INC.

Record Type	Record Description
14h	Bitmap (Type 2)
15h	Start figure
16h	Start chart
17h	PlanPerfect data
18h	Graphics text (Type 2)
19h	Start of WPG data (Type 2)
1Ah	Graphics text (Type 3)
1Bh	PostScript data follows (Type 2)

The following is a listing of the record types, their formats, and the flags associated with them. For more information, please consult the Wordperfect documentation.

Fill Attributes

BYTE 0	Hollow
1	Solid
2	Finely spaced 45-degree lines
3	Medium spaced 45-degree lines
4	Coarsely spaced 45-degree lines
5	Fine 45-degree hatching
6	Medium 45-degree hatching
7	Coarse 45-degree hatching
8	Fine vertical lines
9	Medium vertical lines
10	Coarse vertical lines
11	Dots density 1 (least dense)
12	Dots density 2
11	Dots density 3
13	Dots density 4
14	Dots density 5
15	Dots density 6
16	Dots density 7 (densest)
18	Dots (medium)
19	Dots (coarse)
20	Fine horizontal
21	Medium horizontal
22	Coarse horizontal
23	Fine 90-degree cross-hatching

Ex.1007 / Page 995 of 1158
APPLE INC.

24	Medium 90-degree cross-hatching
25	Coarse 90-degree cross-hatching
26	Fine 45-degree lines
27	Medium 45-degree lines
28	Coarse 45-degree lines
29	Brick pattern (horizontal)
30	Brick pattern (vertical)
31	NA
32	Interweaving
33	NA
34	NA
35	Tile pattern
36	Coarse lines (thick)
37	Alternating dark and light squares
BYTE	Fill-color palette index (0–ffh)

Line Attributes

BYTE 0	None
1	Solid
2	Dash 1 (long)
3	Dots
4	Dash-dot
5	Dash 2 (medium)
6	Dash-dot-dot
7	Dash 3 (short)
BYTE	Line color (0–ffh)
WORD	Line width (arbitrary units)

Marker Attributes

BYTE 0	None
1	Dots
2	Plus sign
3	Star
4	Circle
5	Square
6	Triangle
7	Inverted triangle

Ex.1007 / Page 996 of 1158
APPLE INC.

8	Diamond
9	45-degree cross
BYTE	Marker color (0–ffh)
WORD	Marker height (arbitrary units)

Polymarker
The first area, two bytes in length, holds the number of points. This is followed by a list of WORD coordinate pairs denoting the position of the actual points in arbitrary units.

Line

WORD	X value of start of line
WORD	Y value of start of line
WORD	X value of end of line
WORD	Y value of end of line

These are all in arbitrary units.

Polyline
The first area, two bytes in length, holds the number of points. This is followed by a list of WORD coordinate pairs denoting the position of the actual points in arbitrary units.

Rectangle

WORD	X of lower left of rectangle
WORD	Y of lower left of rectangle
WORD	Width
WORD	Height

These are all in arbitrary units.

Polygon
The first area, two bytes in length, holds the number of vertices. This is followed by a list of WORD coordinate pairs denoting the position of the actual vertices in arbitrary units.

Ex.1007 / Page 997 of 1158
APPLE INC.

Ellipse

WORD	X value of center
WORD	Y value of center
WORD	X radius
WORD	Y radius
WORD	Rotation angle measured from the x axis
WORD	Start of arc (degrees)
WORD	End of arc (degrees)
WORD	Flags: bit 0 connect ends of arc to center, bit 1 connect to each other

Bitmap Type 1

WORD	Width (pixels)
WORD	Height (pixels)
WORD	Bits-per-pixel (1,2,4,8)
WORD	X-resolution of source (pixels/inch)
WORD	Y-resolution of source (pixels/inch)

This is followed by the bitmap data in BYTE format. Note that this may be RLE compressed.

Graphic Text Type 1

WORD	Text length in bytes
WORD	X value of text position
WORD	Y value of text position

This is followed by the text string in BYTE format.

Graphics Text Attributes

WORD	Font character width (arbitrary units)
WORD	Font character height (arbitrary units)
WORD	Reserved
WORD	Reserved
WORD	Reserved
WORD	Reserved
WORD	Reserved
WORD	Font type—e.g., 0df0 Courier, 1150 Helvetica, 1950 Times
BYTE	Reserved
BYTE	Alignment, vertical (0 left, 1 center, 2 right)

Ex.1007 / Page 998 of 1158
APPLE INC.

BYTE	Alignment, horizontal (0 base, 1 center, 2 cap, 3 bottom, 4 top)
BYTE	Color (0–ffh)
WORD	Rotation (degrees from horizontal)

Colormap

WORD	Start color (0-ffh)
WORD	Number of colors
BYTE	Red value of first color
BYTE	Green value of first color
BYTE	Blue value of first color
⋮	
BYTE	Red value of last color
BYTE	Green value of last color
BYTE	Blue value of last color

Start of WPG Data

BYTE	Version number
BYTE	Flags (bit 0 PostScript, maybe bitmap, bit 1 PostScript, no bitmap
WORD	Width of image (arbitrary units)
WORD	Height of image (arbitrary units)

End of WPG Data

This record has no data associated with it. It is used to signal the end of a data section in the file and acts as an end-of-file marker.

PostScript Data Follows

BYTE	Actual PostScript data
⋮	
BYTE	

Output Attributes (WordPerfect 5.0 only)

BYTE	Background color (0–ffh)
BYTE	Foreground color (0–ffh)
WORD	X value of lower left of clipping window
WORD	Y value of lower left of clipping window

Ex.1007 / Page 999 of 1158
APPLE INC.

WORD	Clip window width
WORD	Clip window height

Size and position values are in arbitrary units.

Curved Polyline (WordPerfect 5.1 and later)

DWORD	Size of equivalent data in pre-5.1 files
WORD	Number of points
WORD	X value of first point
WORD	Y value of first point
WORD	X value of first control point
WORD	Y value of first control point
⋮	
WORD	X value of last point
WORD	Y value of last point
WORD	X value of last control point
WORD	Y value of last control point

Bitmap Type 2 (WordPerfect 5.1 and later)

WORD	Rotation angle from horizontal (degrees)
WORD	X value of lower left
WORD	Y value of lower left
WORD	X value of upper right
WORD	Y value of upper right
WORD	Width (pixels)
WORD	Height (pixels)
WORD	Pixel depth (bits)
WORD	Horizontal resolution (pixels/inch)
WORD	Vertical resolution (pixels/inch)

This is followed by the actual bitmap data, which is RLE compressed, although there appear to be some (possibly illegal) variants produced by third-party programs which are not.

Ex.1007 / Page 1000 of 1158
APPLE INC.

Start Figure

DWORD	Length of object data
WORD	Rotation angle from horizontal (degrees)
WORD	X value of lower left
WORD	Y value of lower left
WORD	X value of upper right
WORD	Y value of upper right

This is followed by the figure data.

Start Chart

DWORD	Length of chart data in file
WORD	X value lower left
WORD	Y value lower left
WORD	X value upper right
WORD	Y value upper right

This is followed by the actual chart data.

PlanPerfect Data

This is data associated with WordPerfect Corporation's PlanPerfect application. Please contact WordPerfect for more information.

Graphics Text Type 2 (WordPerfect version 5.1 and later)

DWORD	Size of equivalent data written by version prior to 5.1
WORD	Rotation angle from horizontal (degrees)
WORD	Length of text (characters)
WORD	X value of text start
WORD	Y value of text start
WORD	X value of text end
WORD	Y value of text end
WORD	X scale factor
WORD	Y scale factor
BYTE	Type (0 window, 1 line, 2 bullet chart, 3 simple chart, 4 free-format chart)

This is followed by the string data.

Ex.1007 / Page 1001 of 1158
APPLE INC.

Start of WPG Data Type 2
BYTE Type
WORD Length of data in file

This is followed by the actual data.

RecordLength
RecordLength, the second field of each record, may be a BYTE, WORD, or DWORD in size, depending upon the value stored in the first BYTE of this field (SizeIndicator above). Because it is possible for the same RecordType to have a different size each time it appears in the same WPG file, each record cannot be assigned a RecordType field of a fixed size. You must therefore determine the size of the RecordLength field when you read the record prefix.

If the BYTE value read after the RecordType field is in the range of 00h to FEh, the RecordLength field is a BYTE in size, and this value is used as the number of bytes in the record. If the BYTE is the value FFh, then the RecordLength field is either a WORD or a DWORD in size.

The next WORD of the prefix is then read. If the high bit of this WORD is 0, then this value is the length of the record. If the high bit is 1, then this value is the upper WORD value of a DWORD length value. The next WORD is read and is used as the lower WORD value in the DWORD. This DWORD value is then the length of the record. The following code should help to clarify this logic:

```
BYTE  RecordType;
DWORD RecordLength;
FILE *fp;

RecordType = GetByte(fp);          /* Read the RecordType */
RecordLength = GetByte(fp);        /* Read the RecordLength */

if (RecordLength == 0xFF)          /* Not a BYTE value */
{
  RecordLength = GetWord(fp);      /* Read the next WORD value */
  if(RecordLength & 0x8000)     /* Not a WORD value */
  {
    RecordLength <<= 16;       /* Shift value into the high WORD */
    RecordLength += GetWord(fp);    /* Read the low WORD value */
    }
}
```


Ex.1007 / Page 1002 of 1158
APPLE INC.

Example Records

The following is a description of several of the records found in the WPG format. For a complete listing of all records and values, refer to the WordPerfect Developer's Toolkit.

The first record of a WPG file is always the Start WPG Data (0Fh) record. This record contains information on the size of the image and the version number of the WPG file and has the following format:

```
typedef struct _StartWpgRecord
{
  BYTE Version;          /* WPG Version Flags (always 01h) */
  BYTE WpgFlags;         /* Bit flags */
  WORD Width;            /* Width of image in WP Units */
  WORD Height;           /* Height of image in WP Units */

} STARTWPGREC;
```

Version indicates the WPG file version. This value is currently defined to be 01h.

The eight bits in the WpgFlags field are used as flag values. If Bit 0 is set to 0, then there is no PostScript code included in this WPG file. If Bit 0 is set to 1, then PostScipt code is included in this file. Bits 1 through 7 are reserved and always set to 0.

Width and Height contain the size of the image in WP Units (WPU), each of which is equal to 1/1200th of an inch.

A ColorMapRecord (0Eh) normally follows the StartWpgRecord, unless the image is black and white. If no ColorMapRecord is present, then the default colormap is used instead. There is only one ColorMapRecord per WPG file, regardless of how many bitmap or vector objects the file contains. The current WPG format does not provide a way to assign separate colormaps to specific vector objects and bitmaps.

All images stored in a WPG file, both bitmap and vector, use index values into the colormap to define their colors. This record may define an entire color map unique to this image, or it may define only a smaller colormap used to overlay a portion of the default colormap. To avoid problems with WPC products, the first 16 colors in the colormap should never be changed from their default values. The ColorMapRecord has the following format:

Ex.1007 / Page 1003 of 1158
APPLE INC.

```
typedef struct _ColorMapRecord
{
  WORD  StartIndex;      /* The starting index of this color map */
  WORD  NumberOfEntries;/* The number of entries in this color map*/
  BYTE  *ColorMap[][3].; /* Color map triples */

} COLORMAPREC;
```

StartIndex indicates the starting color index number of this map.

NumberOfEntries indicates the number of contiguous entries in the colormap from the starting index. If entries 178 though 244 in the default colormap were being replaced by this colormap, the value of StartIndex would be 178, and the value of NumberOfEntries would be 66. If the entire colormap were being replaced, the values of these fields would be 0 and 256 respectively.

These two fields are followed by a sequence of three-byte triples, which hold the actual colormap data. The number of triples is equal to the value stored in the NumberOfEntries field. The number of bytes in this field is calculated by multiplying the value of the NumberOfEntries field by 3. The default colormap for WPG files is the same as the IBM VGA standard color table defined in the PS/2 Display Adapter manual.

The VGA colormap structure is also shown in Chapter 2, in the section called "Examples of Palettes."

This colormap contains 256 color entries, each with a 1-byte red, green, and blue color value for a total of 768 map elements. The first 16 colors are those of the IBM EGA color table. Colors 17 through 32 are 16 gray-scale shades. The remaining 224 colors are a palette of 24 individual colors, each with three different intensity levels and three different saturation levels. The WPG color map uses eight bits for red and six bits each for green and blue.

When displaying WPG images using a display adapter, such as the VGA, with fewer bits per primary color, the color values are truncated starting with the least significant bits. For a VGA adapter that has only 6 bits for red, all 8-bit red values in the color table are shifted to the right twice before the value is used. The green and blue values are not changed.

As previously mentioned, a WPG file created with WordPerfect 5.0 can store either bitmap or vector image data, but not both. This is due to a limitation of the Bitmap (0Bh) record structure. This record is now considered obsolete and should not be used when you create new WPG files. The structure of this record is as follows:

Ex.1007 / Page 1004 of 1158
APPLE INC.

```
typedef struct _BitmapType1
{
  WORD  Width;          /* Width of image in pixels */
  WORD  Height;         /* Height of image in pixels */
  WORD  Depth;          /* Number of bits per pixel */
  WORD  HorzRes;        /* Horizontal resolution of image */
  WORD  VertRes;        /* Vertical resolution of image */

} BITMAP1REC;
```

Width and Height describe the size of the bitmap in pixels.

Depth contains the number of bits per pixel. The possible values of this field are 1, 2, 4, or 8 for 2-, 4-, 16-, and 256-color images.

HorzRes and VertRes are the horizontal and vertical resolution of the original bitmap in pixels per inch. These values can also describe the minimum resolution of the screen required to display the image.

The bitmap data follows this record structure. The Bitmap Type 1 (0Bh) record was superseded by the Bitmap Type 2 (14h) record introduced with WordPerfect 5.1. This new record added five fields not found in the Bitmap Type 1 record. These fields contain information on the position of the bitmap on the output device. If you use a Bitmap Type 2 record, it is also possible to store multiple bitmaps in a single WPG file.

The structure of the Bitmap Type 2 record is shown below:

```
typedef struct _BitmapType2
{
  WORD  RotAngle;       /* Rotation angle of bitmap (0-359) */
  WORD  LowerLeftX;     /* Lower-left X coordinate of image */
  WORD  LowerLeftY;     /* Lower-left Y coordinate of image */
  WORD  UpperRightX;    /* Upper-right X coordinate of image */
  WORD  UpperRightY;    /* Upper-right Y coordinate of image */
  WORD  Width;          /* Width of image in pixels */
  WORD  Height;         /* Height of image in pixels */
  WORD  Depth;          /* Number of bits per pixel */
  WORD  HorzRes;        /* Horizontal resolution of image */
  WORD  VertRes;        /* Vertical resolution of image */
} BITMAP2REC;
```

RotAngle is the rotation angle of the bitmap in degrees. This value may be in the range of 0 to 359, with 0 indicating the image is not rotated.

LowerLeftX and LowerLeftY describe the location of the lower-left corner of the image in WPUs.

Ex.1007 / Page 1005 of 1158
APPLE INC.

UpperRightX and UpperRightY describe the location of the upper-right corner of the image in WPUs. Note that the origin point (0,0) of all WPG images is the lower left-hand corner of the output device.

The remaining five fields, Width, Height, Depth, HorzRes, and VertRes, are identical to those in the Bitmap Type 1 record.

It is possible to store two or more images in a WPG file by using multiple Bitmap records. The coordinate information found in a Bitmap Type 2 record will allow the images to be positioned on the output device so they do not overlap. The size of a bitmap in bytes may be determined by multiplying the Height, Width, and Depth fields and then dividing the product by 8:

```
SizeInBytes = (Height * Width * Depth) / 8;
```

Bitmap data is always stored in a WPG file using a byte-wise run-length encoding (RLE) algorithm. (See Chapter 9, *Data Compression*, for more information on run-length encoding algorithms.) Each scan line is encoded separately.

There are four possible types of RLE packets in the WPG algorithm:

- Encoded packet
- Literal packet
- All-bits-on packet
- Repeat scan-line packet

An encoded packet may encode a run of from 1 to 127 bytes in length. An encoded packet always has the most significant bit (MSB) as 1 and the seven least significant bits (LSBs) are a non-zero value. The length of the run is the value of the seven LSBs. If the MSB of this byte is 1, but the seven LSBs are set to 0, then the next byte is read as the run count and the byte value FFh is repeated "run count" times. If the MSB of the byte read is 0, and the seven LSBs are a non-zero value, then this is a literal run. The seven LSBs hold the run-count value and the next "run count" bytes are read literally from the encoded data stream. If the run count is 0, then the next byte is read as the run count and the previous scan line is repeated "run count" times.

The pseudocode for the WPG RLE algorithm is shown below:

```
Read a BYTE
If the Most Significant Bit is ON
  If the 7 LSB are not 0
    The RunCount is the 7 least significant bits
    Read the next BYTE and repeat it RunCount times
```


Ex.1007 / Page 1006 of 1158
APPLE INC.

If the 7 LSB are 0
  Read the next BYTE as the RunCount
  Repeat the value FFh RunCount times
If the Most Significant Bit is OFF
  If the 7 LSB are not 0
    The RunCount is the 7 least significant bits
    The next RunCount BYTEs are read literally
  If the 7 LSB are 0
    Read the next BYTE as the RunCount
    Repeat the previous scan line RunCount times

Encapsulated PostScript (EPS) data may be included in a WPG file by using the PostScript Data Type 1 (11h) record or the PostScript Data Type 2 (1Bh) record. The PostScript Data Type 1 record contains a set of output commands needed to print the EPS code included in the WPG file on a PostScript printer. The structure for the PostScript Data Type 1 record is as follows:

```
typedef struct _PsDataType1
{
  WORD  BbLowerLeftX;    /* Lower left X coordinate of image  */
  WORD  BbLowerLeftY;    /* Lower left Y coordinate of image  */
  WORD  BbUpperRightX;   /* Upper right X coordinate of image */
  WORD  BbUpperRightY;   /* Upper right Y coordinate of image */

} PSTYPE1REC;
```

The four fields in this record contain the bounding-box values of the PostScript image in points. These are the values found in the %%BoundingBox field in the EPS header. The EPS data immediately follows this record. The PostScript Data Type 2 record is used to store one or more EPS images. If the EPS data also contains a TIFF, PICT, WMF, or EPSI image, as is found in a Display PostScript file, this data is converted to a Bitmap Type 2 record that follows the PostScript Data Type 2 record.

The structure for the PostScript Data Type 2 record is shown below:

```
typedef struct _PsDataType2
{
  DWORD RecordLength;        /* Length of the following record */
  WORD  RotAngle;            /* Angle of roation of image */
  WORD  LowerLeftX;          /* Lower-left X coordinate of image */
  WORD  LowerLeftY;          /* Lower-left Y coordinate of image */
  WORD  UpperRightX;         /* Upper-right X coordinate of image */
  WORD  UpperRightY;         /* Upper-right Y coordinate of image */
  BYTE  FileName[40];        /* File name of original EPSF file */
  WORD  BbLowerLeftX;        /* Lower-left X coordinate of bounding box */
```


Ex.1007 / Page 1007 of 1158
APPLE INC.

```
    WORD  BbLowerLeftY;       /* Lower-left Y coordinate of bounding box */
    WORD  BbUpperRightX;      /* Upper-right X coordinate of bounding box */
    WORD  BbUpperRightY;      /* Upper-right Y coordinate of bounding box */

} PSTYPE2REC;
```

RecordLength indicates the number of bytes occuring in the Bitmap Type 2 record following the EPS data. If the EPS data does not have an associated Bitmap Type 2 record, then the value of this field is 0.

The RotAngle, LowerLeftX, LowerLeftY, UpperRightX, and UpperRightY fields have the same meaning as in the Bitmap Type 2 (14h) record.

FileName contains the name of the original EPSF file from which this EPSF code was derived.

The BbLowerLeftX, BbLowerLeftY, BbUpperRightX, and BbUpperRightY fields are the same as in the PostScript Data Type 1 (11h) record.

The EPSF code immediately follows this record. The PostScript Data Type 2 record found in WordPerfect 5.1 and DrawPerfect supersedes the PostScript Data Type 1 record found only in WordPerfect 5.0 and DrawPerfect 1.0. You should always use the Type 2 record rather than the Type 1 when creating new WPG files.

The last record in every WPG file is the End of WPG Data (10h) record. This record has a NULL body; it merely marks the end of the WPG record stream.

For Further Information

The WordPerfect Graphics Metafile format was created and is maintained by WordPerfect Corporation. You can try to get information from:

WordPerfect Corporation
1555 North Technology Way
Orem, UT 84057
Voice: 801-222-4477
Voice: 800-526-5068
FAX: 801-222-5077
BBS: 801-225-4414
FTP: *ftp://ftp.wordperfect.com/*
WWW: *http://www.wordperfect.com/*

Ex.1007 / Page 1008 of 1158
APPLE INC.

WordPerfect was recently acquired by Corel. You can contact Corel at:

Corel Corporation
1600 Carling Avenue
Ottawa, ON, Canada K1Z 8R7
Voice: 613-728-8200
FAX: 613-761-9176
BBS: 613-728-4752
Email: *custserv@corel.ca*
FTP: *ftp://ftp.corel.com/*
WWW: *http://www.corel.com/*

Corel has a page discussing their recent purchase of WordPerfect, and the issues and questions that may arise as a result, at:

http://www.corel.com/novell/menu.htm

A complete description of the WPG format and other technical information associated with WordPerfect software applications may be found in the WordPerfect Corporation Developer's Toolkit for IBM PC Products. This toolkit is available directly from WordPerfect by calling:

WordPerfect Information Services
Voice: 801-225-5000

You can submit technical questions regarding the toolkit to:

WordPerfect Manufacturer/Developer Relations Department
Voice: 801-228-7700
FAX: 801-228-7777
CompuServe: 72567,3612

Please direct all FAX and CompuServe correspondence to "Developer's Toolkit."

Ex.1007 / Page 1009 of 1158
APPLE INC.

NAME:	XBM
ALSO KNOWN AS:	X BitMap
TYPE:	Bitmap
COLORS:	Mono
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	Unlimited
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	ASCII
ORIGINATOR:	X Consortium
PLATFORM:	Any supporting X Window System
SUPPORTING APPLICATIONS:	BRL-CAD
SPECIFICATION ON CD:	No
CODE ON CD:	Yes
IMAGES ON CD:	Yes
SEE ALSO:	XPM

USAGE: Primarily used for the storage of cursor and icon bitmaps for use in the X graphical user interface.

COMMENTS: XBM is a monochrome bitmap format in which data is stored as a C language data array.

Overview

Normally, we think of images as data being stored as binary information in a file. In many cases, however, it is more convenient to represent smaller bitmapped images as collections of ASCII data rather than binary data. If such a small bitmapped image is being used by a software application, such as the cursors and icons found in all graphical user interfaces, the images may be stored as an array of ASCII characters, or even as an array of data values stored in the actual software source code.

Storing small amounts of image data directly as C language source code is the philosophy behind the XBM (X BitMap) format. Small images that will be compiled into a software program are stored as simple arrays of data values, with one array used per stored image. XBM files are therefore nothing more than C

Ex.1007 / Page 1010 of 1158
APPLE INC.

language source files that are read by a compiler, rather than by a graphical display program or bitmap editor, as are most other graphical files.

XBM bitmap data is mostly found in C source header files (with a .h file extension) and in separate XBM bitmap files (with no file extension). Multiple XBM image-data arrays may be stored in a single file, but none of the images may have the same name, or a naming conflict will result.

The XPM (X PixMap) format is similar to XBM. XPM is a cousin of XBM and is capable of storing color bitmap image data and a colormap. XPM is also an ASCII format and is described in the XPM article.

File Organization

XBM files have a height and width, and may define an optional hotspot within the image. The hotspot is used for bitmapped cursors and indicates the absolute position of the cursor on the screen. The hotspot on an arrow cursor is the tip of the arrow, which is usually located at position 0,0 in the bitmap.

In place of the usual image file format header, XBM files have two or four #define statements. The first two #defines specify the height and width of the bitmap in pixels. The second two specify the position of the hotspot within the bitmap, and are not present if no hotspot is defined in the image.

The labels of each #define contain the name of the image. Consider an image that is 8×8 pixels in size, named FOO, with a hotspot at pixel 0,7. This image contains the following #define statements:

```
#define FOO_width 8
#define FOO_height 8
#define FOO_x_hot 0
#define FOO_y_hot 7
```

The image data itself is a single line of pixel values stored in a static array. Data representing our FOO image appears as follows:

```
static unsigned char FOO_bits[] = {
    0x3E, 0x80, 0x00, 0x7C, 0x00, 0x82, 0x41, 0x00};
```

Because each pixel is only one bit in size, each byte in the array contains the information for eight pixels, with the first pixel in the bitmap (at position 0,0) represented by the high bit of the first byte in the array. If the image width is not a multiple of eight, the extra bits in the last byte of each row are not used and are ignored.

Ex.1007 / Page 1011 of 1158
APPLE INC.

XBM files are found in two variations: the older X10 format and the newer (as of 1986) X11 format. The only difference between these formats is how the pixel data is packed. The X11 flavor stores pixel data as 8-bit BYTEs. The older X10 flavor stores pixel data as 16-bit WORDs. There are no markers separating the rows of image data in either of these formats, and the size of an XBM array is limited only by the compiler and machine using the bitmap.

The X10 XBM is considered obsolete. Make sure that any X software you write is able to read both the XBM X10 and X11 formats, but when you write data, use only the X11 XBM format.

File Details

Following is an example of a 16×16 XBM bitmap stored using both its X10 and X11 variations. Note that each array contains exactly the same data, but is stored using different data word types:

```
/* XBM X10 format */
#define xlogo16_width 16
#define xlogo16_height 16

static unsigned short xlogo16_bits[] = {
    0x0f80, 0x1e80, 0x3c40, 0x7820, 0x7810, 0xf008, 0xe009, 0xc005,
    0xc002, 0x4007, 0x200f, 0x201e, 0x101e, 0x083c, 0x0478,
    0x02f0};

/* XBM X11 format */
#define xlogo16_width 16
#define xlogo16_height 16

static unsigned char xlogo16_bits[] = {
    0x0f, 0x80, 0x1e, 0x80, 0x3c, 0x40, 0x78, 0x20, 0x78, 0x10,
    0xf0, 0x08, 0xe0, 0x09, 0xc0, 0x05, 0xc0, 0x02, 0x40, 0x07,
    0x20, 0x0f, 0x20, 0x1e, 0x10, 0x1e, 0x08, 0x3c, 0x04, 0x78,
    0x02, 0xf0};
```

For Further Information

For further information about the XBM format, see the code examples included on the CD-ROM that accompanies this book.

The XBM format is part of the X Window System created by the X Consortium. The X11 source code distribution contains many XBM files (in the */bitmaps* directory) and C language source code functions (such as XCreateBitmapFromData, XCreatePixmapFromBitmapData, XReadBitmapFile, and

Ex.1007 / Page 1012 of 1158
APPLE INC.

XWriteBitmapFile), which operate upon XBM data. The central FTP site for X11 distribution is:

ftp://ftp.x.org/

Other references containing information on XBM include the following:

Gettys, James, and Robert W. Scheiffler, *X Window System*, Digital Press, 1992.

Gettys, James, Robert W. Scheiffler, et al. *Xlib-C language X Interface*, X Consortium Standard, X Version 11, Release 5, First Revision, August 1991.

Nye, Adrian, *Xlib Programming Manual*, third edition, O'Reilly & Associates, Inc., Sebastopol, CA, 1992.

Ex.1007 / Page 1013 of 1158
APPLE INC.

NAME:	XPM
ALSO KNOWN AS:	X PixMap
TYPE:	Bitmap
COLORS:	Unlimited
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	NA
MULTIPLE IMAGES PER FILE:	Yes
NUMERICAL FORMAT:	NA
ORIGINATOR:	Groupe Bull
PLATFORM:	X Window
SUPPORTING APPLICATIONS:	XPM Library
SPECIFICATION ON CD:	No
CODE ON CD:	Yes
IMAGES ON CD:	Yes
SEE ALSO:	XBM

USAGE: Used to store X Window pixmap information to a disk file.

COMMENTS: XPM is a de facto format only. There is no official standard file format for storing X multibit bitmap data.

Overview

The XPM (X PixMap) format is the current de facto standard for storing X Window pixmap data to a disk file. This format is supported by many image editors, graphics window managers, and image file converters. (See "For Further Information" later in this article.)

XPM is capable of storing black-and-white, gray-scale, or color image data. Hotspot information for cursor bitmaps may also be stored. Although small collections of data, such as icons, are typically associated with XPM files, there is no limit to the size of an image or the number of colors that may be stored in an XPM file.

Ex.1007 / Page 1014 of 1158
APPLE INC.

File Organization

XPM stores image data in the form of ASCII text formatted as a Standard C character string array. This type of format allows XPM files to be edited easily with any text editor, to have comments inserted at any point within the file, to be included as data in C and C++ programs, and to be easily transmitted via electronic mail.

Also, because of its human-readable, plain-text format, XPM does not support any native form of data compression. External compression programs, such as the UNIX *compress* program, must be used to reduce the physical size of an XPM file.

File Details

The basic syntax of an XPM file is:

```
/* XPM */
static char * <pixmap_name>[] = {
<Values>
<Colors>
<Pixels>
<Extensions>
};
```

XPM files always start with the string XPM, delimited by Standard C comment tokens. This is an identifier indicating that the file contains an XPM data structure. Following this identification comment is a Standard C array containing the actual pixmap data in the form of character strings. The data in this array is arranged into four sections: Values, Colors, Pixels, and Extensions.

The <Values> section is similar to an image file format header. It contains values indicating the width and height of the pixmap, the number of colors in the image, the number of characters per pixel, the hotspot coordinates in the image, and a marker indicating whether the XPM file contains an optional extension section. The hotspot values and the extension marker are optional values and need not appear if there is no hotspot or extension section.

The expanded syntax of the <Values> section is shown below:

```
<width><height><numcolors><cpp> [ <x_hotspot><y_hotspot> ] [ XPMEXT ]
```

The <Colors> section defines the ASCII characters that represent the pixmap data in the <Pixels> section of this array. There is one string in this section per

Ex.1007 / Page 1015 of 1158
APPLE INC.

color in the pixmap. Each string in the <Colors> section may be defined using the following expanded syntax:

```
<character> { <key> <color> } { <key> <color> }
```

The <character> is the character(s) used to present a single pixel. The actual number of characters in this field equals the <cpp> value in the <Values> section. The <character> is followed by one or more groups of values. These groups define the type of color(s) each <character> represents.

The <key> indicates the type of color or data represented, and may have one of the following values:

m	Mono
s	Symbolic name
g4	Four-level gray scale
g	Gray scale (more than four levels)
c	Color

The <color> is any of the following:

- A color name
- A # followed by the RGB code in hexadecimal
- A % followed by the HSV code in hexadecimal
- A symbolic name
- The string None, indicating that the pixel is transparent and is part of a masking bitmap rather than a pixmap.

The <Pixels> section contains the actual bitmap data. There is <height> number of strings, each containing <width> number of characters. Each character in a pixel string is a character previously defined in the <Colors> section.

The <Extension> section lets additional string information be stored in the XPM file data. If the XPMEXT marker appears in the <Values> section, then an extension block is found after the <Pixels> section. If there is no marker, then the XPM file extension section does not appear and is said to be empty.

An <Extension> section is composed of one or more sub-sections. Each subsection may have one of two possible formats. The first format is a single extension composed of only one string:

```
XPMEXT <extension_name> <extension_data_string>
```


Ex.1007 / Page 1016 of 1158
APPLE INC.

The second format is a single extension sub-section composed of multiple strings:

```
XPMEXT <extension_name> <extension_data_string1>
       <extension_data_string2>
```

The <Extension> section always ends with the XPMENDEXT marker.

The following is an example of an XPM file containing a bitmap, a hotspot, four bitmap character colors, and an extension section with four sub-sections:

```
/* XPM */
static char * plaid[] =
{
/* plaid pixmap */
/* width height ncolors chars_per_pixel */
"22 22 4 2 0 0 XPMEXT",
/* colors */
"   c red       m white  s light_color",
"Y  c green     m black  s ines_in_mix",
"+  c yellow    m white  s lines_in_dark ",
"x              m black  s dark_color ",
/* pixels */
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"Y Y Y Y Y x Y Y Y Y Y + x + x + x + x + x + ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"          x               x x x Y x x x ",
"          x             x   x   Y   x   x   ",
"          x               x x x Y x x x ",
"          x             x   x   Y   x   x   ",
"          x               x x x Y x x x ",
"x x x x x x x x x x x x x x x x x x x x x x ",
"          x               x x x Y x x x ",
"          x             x   x   Y   x   x   ",
"          x               x x x Y x x x ",
"          x             x   x   Y   x   x   ",
"          x               x x x Y x x x ",
"XPMEXT ext1 data1",
"XPMEXT ext2",
"data2_1",
"data2_2",
"XPMEXT ext3",
"data3",
```


Ex.1007 / Page 1017 of 1158
APPLE INC.

```
"XPMEXT",
"data4",
"XPMENDEXT"
};
```

For Further Information

For further information about the XPM format, see the code examples included on the CD-ROM that accompanies this book.

The XPM file format was created by Arnaud Le Hors and Colas Nahaboo of the KOALA Project at Groupe Bull Research. If you have questions or comments about XPM, you can contact:

BULL Research
c/o INRIA
2004 route des Lucoiles
06565 Valbonne Cedex
France
lehors@x.org

You can subscribe to the XPM mailing list by sending an email request to:

xpm-talk-request@sophia.inria.fr

The XPM Library is a collection of Xlib-level functions for the X Window System that read, write, and manipulate XPM data in both files and in memory. The latest version of this library may be obtained by anonymous FTP from:

ftp://avahi.inria.fr/contrib/xpm.tar.Z
ftp://export.lcs.mit.edu

The current version of the XPM library is 3.2g (April 1993).

Other applications capable of generating XPM output include xsnap and pixt, both available via anonymous FTP from:

ftp://avahi.inria.fr/
ftp://ftp.x.org/

A number of software packages included on the CD-ROM also support the conversion of XPM files; see the discussion of FBM, ImageMagick, pbmplus, xli, XLoadimage, and xv in Appendix C, *Installation and Setup.*

A collection of XPM icons also exists in:

ftp://ftp.x.org/contrib/AIcons/

Ex.1007 / Page 1018 of 1158
APPLE INC.

XWD

NAME:	XWD
ALSO KNOWN AS:	X Window Dump
TYPE:	Bitmap
COLORS:	Unlimited
COMPRESSION:	Uncompressed
MAXIMUM IMAGE SIZE:	64K×64K
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Big- and little-endian
ORIGINATOR:	X Consortium
PLATFORM:	UNIX X Windows
SUPPORTING APPLICATIONS:	Many
SPECIFICATION ON CD:	Yes
CODE ON CD:	No
IMAGES ON CD:	Yes
SEE ALSO:	None

USAGE: XWD is used to store images of captured X window displays.

COMMENTS: Many image-processing and display applications and toolkits read and write XWD format image files.

Overview

The XWD (X Window Dump) format is used specifically to store screen dumps created by the X Window System. Under X11, screen dumps are created by the *xwd* client. Using *xwd*, the window or background is selected to dump and an XWD file is produced containing an image of the window. If you issue the following command:

```
% xwd -root > output.xwd
```

the entire contents of the current display are saved to the file *output.xwd*. The *id* of the window to dump may also be specified by using the *–id* command-line flag on versions of *xwd* prior to Release 5.

Ex.1007 / Page 1019 of 1158
APPLE INC.

File Organization

The first version of the X Window System to support window dumps was X10. Only gray-scale and color-mapped dumps were supported, and the bitmapped data was never compressed. The X10 version of XWD contains the following header:

```
typedef struct _X10WindowDump
{
  LONG  HeaderSize;          /* Header size in bytes */
  LONG  FileVersion;         /* X10 XWD file version (always 06h) */
  LONG  DisplayType;         /* Display type */
  LONG  DisplayPlanes;       /* Number of display planes */
  LONG  PixmapFormat;        /* Pixmap format */
  LONG  PixmapWidth;         /* Pixmap width */
  LONG  PixmapHeight;        /* Pixmap height */
  SHORT WindowWidth;         /* Window width */
  SHORT WindowHeight;        /* Window height */
  SHORT WindowX;             /* Window upper left X coordinate */
  SHORT WindowY;             /* Window upper left Y coordinate */
  SHORT WindowBorderWidth;   /* Window border width */
  SHORT WindowNumColors;     /* Number of color entries in window */

} X10WINDOWDUMP;
```

HeaderSize is the size of the header in bytes. This value is always 40.

FileVersion contains the version number of the XWD file. This value is always 06h.

DisplayType is the type of the display from which the image was dumped.

DisplayPlanes is the number of color planes in the image data. This value is typically 01h or 03h.

PixmapFormat indicates the format of the bitmap. A value of 00h indicates a single-plane bitmap (XYFormat), and a value of 01h indicates a bitmap with two or more planes (ZFormat).

PixmapWidth and PixmapHeight represent the size of the image in pixels.

WindowWidth and WindowHeight represent the size of the window to display.

WindowX and WindowY represent the position of the window on the display.

WindowBorderWidth indicates the width of the window border in pixels.

WindowNumColors specifies the number of colors that can be displayed in the window.

Ex.1007 / Page 1020 of 1158
APPLE INC.

If the image is a PseudoColor image, a color map immediately follows the header. The color map contains one entry per color in the image, and each entry has the following format:

```
typedef struct _X10ColorMap
{
  WORD EntryNumber;      /* Number of the color-map entry */
  WORD Red;              /* Red-channel value */
  WORD Green;            /* Green-channel value */
  WORD Blue;             /* Blue-channel value */
} X10COLORMAP[WindowNumColors];
```

EntryNumber is the number of the color-map entry. This value starts at 00h. Color maps typically do not exceed 256 entries in size.

Red, Green, and Blue are the RGB channel values for this entry. The range of each of these values is typically 0 to 65535; often, only the high byte of the value is set (i.e., the value is 0–255 shifted left eight bits.)

The XWD format was revised for Version 11 of the X Window System. The new format can store more types of image data and many fields have been added to the header and to the color map, reflecting the increased graphics capabilities of X11 over X10.

The Version 11 XWD file format contains the following header:

```
typedef struct _X11WindowDump
{
  DWORD HeaderSize;      /* Size of the header in bytes */
  DWORD FileVersion;     /* X11WD file version (always 07h) */
  DWORD PixmapFormat;    /* Pixmap format */
  DWORD PixmapDepth;     /* Pixmap depth in pixels */
  DWORD PixmapWidth;     /* Pixmap width in pixels */ /
  DWORD PixmapHeight;    /* Pixmap height in pixels */
  DWORD XOffset;         /* Bitmap X offset */
  DWORD ByteOrder;       /* Byte order of image data */
  DWORD BitmapUnit;      /* Bitmap base data size */
  DWORD BitmapBitOrder;  /* Bit-order of image data */
  DWORD BitmapPad;       /* Bitmap scan-line pad*/
  DWORD BitsPerPixel;    /* Bits per pixel */
  DWORD BytesPerLine;    /* Bytes per scan-line */
  DWORD VisualClass;     /* Class of the image */
  DWORD RedMask;         /* Red mask */
  DWORD GreenMask;       /* Green mask */
  DWORD BlueMask;        /* Blue mask */
  DWORD BitsPerRgb;      /* Size of each color mask in bits */
  DWORD NumberOfColors;          /* Number of colors in image */
  DWORD ColorMapEntries;         /* Number of entries in color map */
  DWORD WindowWidth;     /* Window width */
  DWORD WindowHeight;    /* Window height */
```


Ex.1007 / Page 1021 of 1158
APPLE INC.

```
    LONG  WindowX;          /* Window upper left X coordinate */
    LONG  WindowY;          /* Window upper left Y coordinate */
    DWORD WindowBorderWidth;        /* Window border width */

} X11WINDOWDUMP;
```

HeaderSize is the size of the header in bytes. This value is always 40.

FileVersion contains the version number of the XWD file. This value is always 07h.

PixmapFormat is the format of the image data. A value of 00h indicates a 1-bit (XYBitmap) format. A value of 01h indicates a single-plane bitmap (XYPixmap). A value of 02h indicates a bitmap with two or more planes (ZPixmap).

PixmapDepth is the depth of the bitmap in pixels. This value is 1 to 32.

PixmapWidth and PixmapHeight represent the size of the image in pixels.

XOffset specifies the number of pixels to ignore at the beginning of each scanline.

ByteOrder indicates the byte order of the image data. Values for this field are 00h for least significant byte first, and 0 for most significant byte first.

BitmapUnit is the size of each data unit in each scan line. This value may be 8, 16, or 32.

BitmapBitOrder indicates the order of the bits within each byte of image data. Values for this field are 00h for least significant byte first, and 0 for most significant byte first.

BitmapPad is the number of bits of padding added to each scan line. This value may be 8, 16, or 32.

BitsPerPixel contains the size of each pixel in bits. For StaticGray and GrayScale images, this value is 1. For StaticColor and PseudoColor images, this value is 2 to 15 (typically 8). For TrueColor and DirectColor images, this value is 16, 24, or 32.

BytesPerLine is the size of each scan line in bytes.

VisualClass indicates the format of the image data:

- Even-numbered values indicate fixed-image data that cannot be changed in memory.
- Odd-numbered values indicate dynamic image data that may be altered.

Ex.1007 / Page 1022 of 1158
APPLE INC.

- The values 00h (StaticGray) and 01h (GrayScale) specify a gray-scale image.
- The values 02h (StaticColor) and 03h (PseudoColor) indicate a color mapped image.
- The values 04h (TrueColor) and 05h (DirectColor) indicate true-color image data.

RedMask, GreenMask, and BlueMask are the RGB mask values used by ZPixmaps.

BitsPerRgb is the size of each RedMask, GreenMask, and BlueMask in bits.

NumberOfColors specifies the number of colors in the image. This value also indicates the number of colors for colormapped images as well.

ColorMapEntries contains the number of entries in the color map. This value is 00h if there is no color map.

WindowWidth and WindowHeight are the size of the window to display.

WindowX and WindowY contain the position of the window on the display.

WindowBorderWidth is the width of the X Window border in pixels. If the border has not been captured in the dump, this value is 00h.

The color map immediately follows the header. Each entry in the color map is 12 bytes in size and has the following format:

```
typedef struct _X11ColorMap
{
  DWORD EntryNumber;     /* Number of the color map entry */
  WORD  Red;             /* Red-channel value */
  WORD  Green;           /* Green-channel value */
  WORD  Blue;            /* Blue-channel value */
  CHAR  Flags;           /* Flag for this entry */
  CHAR  Padding;         /* WORD-align padding */

} X11COLORMAP[ColorMapEntries];
```

EntryNumber is the number of the color map entry. This value starts at 00h. Color maps typically do not exceed 256 entries in size.

Red, Green, and Blue are the RGB channel values for this entry. The range of each of these values is typically 0 to 65535; often, only the high byte of the value is set (i.e., the value is 0–255 shifted left eight bits.)

Ex.1007 / Page 1023 of 1158
APPLE INC.

Flags indicates which of the color channels in the color map are actually used. The value of this field is typically 07h, indicating that all three channels are used.

Padding is a byte set to a value of 00h and used to pad the color map entry out to an even WORD boundary in size.

For Further Information

For further information about the XWD format, see the documentation included on the CD-ROM that accompanies this book.

The XWD format is part of the X Window System created by the X Consortium. Information about the XWD format, and, indeed, all of the file formats associated with the X Window System, is scattered over a wide variety of header files (in */usr/include/X11*) and UNIX manual pages.

The central FTP distribution site for X11 is:

ftp://ftp.x.org/

Many image-processing and display applications and toolkits included on the CD-ROM (e.g., FBM, ImageMagick, pbmplus, xli, xloadimage, and xvread) and write XWD-format image files, and documentation for those tools may contain additional information about XWD.

Ex.1007 / Page 1024 of 1158
APPLE INC.

ZBR

NAME:	ZBR
ALSO KNOWN AS:	Zebra Metafile
TYPE:	Metafile
COLORS:	24-bit
COMPRESSION:	None
MAXIMUM IMAGE SIZE:	1 meter x 1 meter 2560 dpi
MULTIPLE IMAGES PER FILE:	No
NUMERICAL FORMAT:	Little-endian
ORIGINATOR:	Zoner Software
PLATFORM:	Microsoft Windows
SUPPORTING APPLICATIONS:	Zoner Zebra for Windows, Zoner Archive, Zoner Present
SPECIFICATION ON CD:	No
CODE ON CD:	Yes
IMAGES ON CD:	Yes
SEE ALSO:	Microsoft Windows Bitmap, Microsoft Windows Metafile

USAGE: Storage and exchange of vector graphics including bitmaps and multi-layered sheets.

COMMENTS: Usable in DTP applications.

Overview

ZBR (Zebra Metafile) is the native metafile format of the Zebra for Windows vector graphics editor supported by Zoner Software. ZBR v1.0 supports these main types of raster and vector entities:

Polyline (combination of lines and Bezier curves)
Rectangle (including rounded and sharp corners, etc.)
Ellipse
Star
Polygon
Bitmap

Ex.1007 / Page 1025 of 1158
APPLE INC.

Table
Paragraph text
Artistic text
WMF object

All of these entities, excluding bitmaps, can be defined using the following attributes:

Style of pen
- type
- width
- line endings (arrow, circle, . . .)

Style of fill
- solid color
- gradient (multicolor)
  - linear
  - radial
  - conical
  - square
  - cross
- pattern fill (user defined)
- bitmap fill

Shadow
- offset
- color or change of brightness of object

ZBR2.0, due out sometime in 1996, will support multi-page documents and will conform to Microsoft's Structured Storage specification and OLE 2.0. New entities will be added (such as arc and braced), and Microsoft Windows Metafile (WMF) objects will be supported as a main entity.

File Organization

ZBR files contain a header followed by a thumbnail bitmap of the primary graphics data, a variable-size section of configuration data, a color palette, and one or more object entities.

Ex.1007 / Page 1026 of 1158
APPLE INC.

Header
Preview Bitmap
Configuration Data
Color Palette
Object 1
Object 2
Object *n*

File Details

The ZBR file header is 104 bytes in length and has the following format:

```
typedef struct _ZbrHeader
{
    WORD  FileId;           /* File ID value (always 029Ah) */
    WORD  FileVersion;      /* Version of the file */
    CHAR  Comment[100];     /* Text comment */
} ZBRHEADER;
```

FileId is the file identification value and is always 029Ah.

FileVersion is the version of the ZBR format to which the file conforms. This value is 1 for v1.*x*, 2 for v1.1*x* and 1.2*x*, 3 for v1.49, and 4 for v1.50.

Comment is a NULL-terminated text comment string 100 bytes in length.

Following Comment is a preview image of the file's primary graphics data. This preview is always 5264 bytes in size:

```
typedef struct _ZbrPreview
{
    BYTE PreviewDibPalette[64]; /* Preview bitmap color palette */
    BYTE PreviewDib[5200];      /* Preview bitmap of primary image */
} ZBRPREVIEW;
```

PreviewDibPalette is a 64-byte, Windows color palette containing 16 RGBQUAD entries. See the information on Windows color palettes below.

PreviewDib is a 5200-byte, 16-color Microsoft Windows Device Independent Bitmap (DIB) 100×100 pixels in size. Each pixel is four bits in depth, packed two pixels per byte, and padded to the nearest byte boundary. The pixels contain index values pointing into the color palette stored in the PreviewDibPalette field.

Ex.1007 / Page 1027 of 1158
APPLE INC.

Following PreviewDib is a variable-size block of configuration data used by Zoner software applications. The format of the data in this configuration block is not currently published, but we can assume that it is at least four bytes in size:

```
typedef struct _ZbrConfiguration
{
    DWORD ConfigLength;  /* Size of the local configuration data in bytes */
    BYTE  *LocalConfig;  /* Local configuration data (variable size) */
} ZBRCONFIGURATION;
```

ConfigLength is the length of the LocalConfig field in bytes.

LocalConfig is the actual configuration data.

Following LocalConfig is the color palette information for the primary graphics data. We can also assume that this structure is at least four bytes in length:

```
typedef struct _ZbrPalette
{
    DWORD PaletteLength;     /* Length of color palette in bytes */
    BYTE  *Palette;          /* Color palette (variable size) */
} ZBRPALETTE;
```

PaletteLength is the total length of the palette in bytes. Each palette entry is four bytes in size, so a 256-color palette would be 1024 bytes in length.

Palette is an array of Microsoft RGBQUAD palette entry structures. Each palette entry is four bytes in size and has the following structure:

```
typedef struct _RgbQuad
{
    BYTE Blue;      /* Blue component */
    BYTE Green;     /* Green component */
    BYTE Red;       /* Red component */
    BYTE Reserved;  /* Padding (always 0) */
} RGBQUAD;
```

Blue, Green, and Red hold the color component values for a pixel each in the range 0 to 255.

Reserved is used to pad the structure to end on an even-byte boundary and is always zero.

The number of palette entries may be calculated as such:

NumOfEntries = PaletteLength / sizeof(RGBQUAD);

Ex.1007 / Page 1028 of 1158
APPLE INC.

Given the structure of these four sections, we can construct the following structure of a ZBR file:

```
typedef struct _ZbrFile
{
    ZBRHEADER         Header;
    ZBRPREVIEW        Preview;
    ZBRCONFIGURATION  Configuration;
    ZBRPALETTE        Palette;
    VOID              *Objects;
} ZBRFILE;
```

Image Data

The structure and layout of the object entities in the ZBR format is not published. It is, however, possible to read the header and to display the preview bitmap. Example code for a Windows application that displays the preview bitmap from a ZBR file may be found on the CD-ROM.

For Further Information

For further information about the ZBR format, please contact:

Zoner Software Ltd.
Development Department.
Kozeluzska 7
Brno CZ–602 00
Czech Republic
Voice: +42-5-45214788
FAX: +42-5-45214788
Email: *zoner@zoner.anet.cz*

Ex.1007 / Page 1029 of 1158
APPLE INC.

PART THREE

Appendices

APPENDIX A

Graphics Files and Resources on the Internet

Graphics files may be found in a variety of places on the Internet. They are stored as files in FTP archives, used on World Wide Web (WWW) pages as wallpaper and menus, exchanged between people as electronic mail, and distributed around the earth on the USENET global bulletin board system (BBS).

Graphics files are just chunks of data. The Internet was specifically designed to move chunks of data, easily and efficiently, from one computer to another. So you can probably guess that there is more than one way to send, retrieve, store, find, and view graphics files on the Internet.

This section explores a number of ways you can use the information services found on the Internet to collect, transport, and distribute graphics files. These include email, USENET, FTP, Archie, and the World Wide Web. We'll also briefly mention the Internet etiquette, or netiquette, that you should follow when you use these services.

Encoding of Graphics Files

Before discussing specific Internet services, let's look at a legacy of the Internet, the 7-bit data path, and how this affects the transmission and handling of graphics files.

It is reasonable to expect that a byte sent from one computer along a data path to another computer will retain the value it stores. For example, if I send from my computer a byte of data containing the value A and if the byte arrives at your computer still containing the value A, we can say that an error-free transfer of data has occurred. If I send you the byte value A, but when you receive this byte, it contains the value Q instead, we can say that a data transmission error has occurred.

Ex.1007 / Page 1031 of 1158
APPLE INC.

Parity

A very popular method of detecting such data transmission errors is called *single-bit parity*. Parity is used to determine whether the bits in a received byte are the same value they were when the byte was sent. The lower seven bits of each byte contain the data (hence the term *7-bit data path*), and the eighth bit contains the parity information. We refer to these bits as the data bits and the parity bit, respectively.

Parity may use either an even or an odd encoding scheme. If a communications link uses an even parity scheme, the parity bit would be 1 if there were an even number of 1's data bits in the byte, and 0 if the number of 1's data bits were an odd number. If a communications link uses an odd parity scheme, it's just the opposite; a parity value of one indicates an odd number of 1's data bits in the byte. Any byte received that contained a mismatch between the number of 1's data bit values and the parity bit value is said to have a parity error and would therefore be considered corrupt.

It should be obvious from this description that parity is a very limited form of error checking, one that has no built-in form of error recovery, and that it is by no means a foolproof method of detecting erroneous data. However, parity is probably the simplest and most inexpensive form of data transmission error detection yet devised. Although parity has been outdated by error-correcting protocols, many of the Internet's communications links still use parity.

Unfortunately, the use of parity error checking prevents the direct transmission of 8-bit binary data. When parity is used, only seven bits in a byte may contain data. Binary data requires eight data bits per byte for storage and transmission, precluding the use of conventional parity schemes. For this reason, data in binary form cannot be reliably sent to any point on the Internet. This presents a problem for us because most graphics files contain binary data.

How, then, do we exchange binary data across the Internet? The solution is to convert our 8-bit binary data to a 7-bit format. ASCII is the de facto standard for 7-bit data on the Internet (although habitual users of mainframes, where one of the several flavors of EBCDIC presides, may disagree). Another de facto Internet standard is used for binary-to-ASCII data conversion and is called *uucoding*.

Uucoding with uuencode and uudecode

Uucoding (UNIX-to-UNIX coding) is a simple algorithm used to convert three bytes of 8-bit binary data to four bytes of 7-bit ASCII data. The uuencode program converts a binary file to an ASCII equivalent in a process called *uuencoding*.

Ex.1007 / Page 1032 of 1158
APPLE INC.

A uuencoded file is approximately 30 percent larger than the original file. Converting every three bytes into four accounts for 25 percent of the growth, with the other 5 percent being eaten up by control information. Uuencoding is also perfectly lossless; you will decode the exact file that was encoded every time.

The uuencode and uudecode programs originated on the UNIX operating system, but have long since been ported to almost every other operating system (certainly any operating system running on a computer that exchanges information over the Internet).

Let's look at an example. Suppose that we have a graphics file named *toshi.jpg* and we want to email it to someone on the Internet. We would first need to convert the binary graphics file to an ASCII uuencoded file by issuing the following command:

```
uuencode toshi.jpg toshi.jpg > toshi.uue
```

In this command line, uuencode reads the file *toshi.jpg* and encodes it using the file label *toshi.jpg*. (Note that the input filename and the file label need not be the same.) Uuencode always sends its output to the display, but here we've redirected it to the file *toshi.uue*. If we were to look at the file *toshi.uue* via a text editor, we might see:

```
begin 600 toshi.jpg
M```!``$`("`0``````#H`@``%@````@````@````0`````$`!````````@`(`.
M`````````````````````````````"````"`````@(``@````(``@`"`@```6
M@("```,#`P````/\``/\```#__P#_````_P#_`/__``#___\``````````````L
!____\
end
```

All of the uuencoded data is contained between the "begin" and "end" lines. The "600" is the UNIX file mode, and "toshi.jpg" is the file label used by uudecode as the name of the file in which to save the uudecoded data.

To convert a uuencoded file back to its original form, we issue the uudecode command:

```
uudecode toshi.uue
```

This command reads the *toshi.uue* file and recreates the original file. Uudecode is also smart enough to strip away all lines that precede the "begin" line and that follow the "end" line. If you need to change the name of the decoded file, you can use a simple text editor to change the file label on the "begin" line.

In this example, the uuencoded file *toshi.uue* is called a single-part uuencoded file because it is stored in a single file. Large uuencoded files are frequently

Ex.1007 / Page 1033 of 1158
APPLE INC.

spilt into smaller parts and are stored in separate files for purposes of posting and emailing. (See the section on email that follows.) These split files are called *multi-part uuencoded files.*

Uudecoding a multi-part file is an easy job if you have a smart uudecoding program (such as aub, unc, uudo, uuexe, uucat, uuconvert, uulite, or uuxfer) which is able to read the headers of news articles or email messages and to decode the parts in the proper order. But, if you only have a simple uudecoding program that expects all of the data to be in a single file or file stream, then you have a bit of manual work to do.

First, make sure that you have all the parts of the uuencoded file. For example, if the file is separated into three parts, then you should have three files, each with some kind of part designation, such as a "Subject:" or a separator line containing the strings "Part [1/3]", "Part [2/3]", and so on. You must next concatenate these files together in the proper order. With UNIX you would do the following:

```
cat file.01 file.02 file.03 > file.uue
```

With MS-DOS, you would type:

```
copy file.01+file.02+file.03 file.uue
```

Now edit *file.uue* and remove all headers and blank lines, returning the uuencoded data to its original contiguous state. This is how the contents of *file.uue* might look before editing:

```
[ Start of Part 1/3 ]
begin 644 judi.jpg
M```!``$`("`0``````#H`@``%@```"@````@````0`````$`!```````@`(`.
M`````````````````````````````"```"`````@(``@````(``@`"`@```6
M@("```,#`P````/\``/\```#__P#_````_P#_`/__``#___\``````````````L
[ End of Part 1/3 ]

[ Start of Part 2/3 ]
M875D(')A=&5S+@T*#0H@("`@("U(/'-T<FEN9SX-"B`@("`@("`@("!W;W)K>
M<R!J=7-T(&QI:V4@+6@@97AC97!T('1H870@:6YS=&5A9"!O9B`@<V5T=&ENG
M9R`@=&AE#0H@("`@("`@("`@:&ED:6YG("!F;&%G("!F;W(@82!H96%D97(@2
[ End of Part 2/3 ]

[ Start of Part 3/3 ]
M_____@?__^`#___``?__P``E
M__^```___P`/__^`!___@`?__D`#__P``__\``/__``#__Y@`___X`?______S
!____\
``
end

[ End of Part 3/3 ]
```


Ex.1007 / Page 1034 of 1158
APPLE INC.

And after editing:

```
begin 644 judi.jpg
M```!``$`("`0``````#H`@``%@```"@````@````0`````$`!````````@`(`.
M`````````````````````````````"```"`````@(``@````(``@`"`@```6
M@("```,#`P````/\``/\```#__P#_````_P#_`/__``#___\``````````````L
M875D(')A=&5S+@T*#0H@("`@("U(/'-T<FEN9SX-"B`@("`@("`@("!W;W)K>
M<R!J=7-T(&QI:V4@+6@@97AC97!T('1H870@:6YS=&5A9"!O9B`@<V5T=&ENG
M9R`@=&AE#0H@("`@("`@("`@:&ED:6YG("!F;&%G("!F;W(@82!H96%D97(@2
M_____@?__^`#___``?__P``E
M__^```___P`/__^`!___@`?__D`#__P``__\```/__``#__Y@`___X`?_____S
!____\
``
end
```

Now, all you need to do is convert the file using uudecode to retrieve the original file.

You may experience a problem with uudecode arising from the fact that the character sequence used to terminate lines in an ASCII file differs depending upon the operating system. UNIX and the Amiga use a linefeed (ASCII 10); the Macintosh uses a carriage return (ASCII 13); and MS-DOS uses both a carriage return and a linefeed in combination.

Many uudecoders (including the original UNIX program) do not handle uuencoded files with something other than native end-of-line character(s) very well. For example, a uudecode program that expects lines of ASCII text to be terminated using only linefeeds will not be able to handle a uuencoded file whose lines are terminated only with carriage returns. The same program may complain if every linefeed in a file is also followed by a carriage return (the notorious, but harmless, "short file" error under UNIX).

For those EBCDIC people who are wondering "What about me? I don't/can't use ASCII!", there is a program called xxencode. This program converts binary files to an EBCDIC-compatible ASCII format that resembles the output from uuencode but is not readable by uudecode. If you've been having problems with uuencoded files being munged by ASCII-to-EBCDIC converters, then using xxencode instead of uuencode on your files may solve your problems.

The uuencode and uudecode programs are included with every flavor of the UNIX operating system. Implementations of these programs have been ported to almost every other operating system and are freely available in most major software archives. However, not all uuencode programs use the same encoding algorithm as the original UNIX uuencode program, or even the same command-line syntax. As uuencode has been ported to other operating systems, people have changed it to make it more efficient or compatible with other

Ex.1007 / Page 1035 of 1158
APPLE INC.

utilities, sacrificing the backward compatibility with the original program. This unfortunate occurrence has led to a widespread criticism of the de facto uuencode program and has given rise to a movement to officially replace uuencode with a more standard and robust binary-to-ASCII translation program, such as btoa (binary-to-ASCII) and mmencode (also known as mimencode).

Email

Email is the most basic form of data transfer on the Internet. If you have access to the Internet, through either a simple shell account or a full TCP/IP connection, then you most likely have email capability. If you need to send one or more graphics files to only a few people, email is the best way to do so.

If you are going to email graphics files, then you must first convert them from binary to ASCII data. Sophisticated mail programs will transparently perform the binary-to-ASCII translation for you when sending mail and will convert the file back again when it is received. But not everyone has such a mail program, so let's look at how we would manually control this process.

We already know how to use uuencode to create an ASCII file from binary data. A uuencoded file may be emailed as you would any other text file, for example:

```
uuencode toshi.jpg toshi.jpg > toshi.uue
Mail -s "JPEG file of my cat"  user@site.domain < toshi.uue
```

This example shows the two commands used to uuencode and email the graphics file *toshi.jpg* using the Berkeley UNIX mail program. The person receiving the email will see the uuencoded data of a file named *toshi.jpg*. All that person needs to do is save the email to a file and uudecode it.

One other way of sending binary files via email is by using Multipurpose Internet Mail Extensions (MIME). MIME is an extension of the Internet's Simple Mail Transfer Protocol (SMTP) and is a common way to move multiple items of data as a single package across the Internet. MIME has a method of sending binary information in an email message that allows you to avoid the uuencode/uudecode steps. In fact, any MIME-compliant email program does all the work for you. All you need to do is specify that a file (often called an *attachment*) is to be sent using MIME. The other half of the bargain is that the person receiving your email must also be using a MIME-compliant email program to decode your MIME-encoded binary file.

Ex.1007 / Page 1036 of 1158
APPLE INC.

Email Etiquette: Splitting Large Files

Email etiquette for graphics files is mostly about bandwidth. The email that you send may seem to travel directly from you to your friends, but it does so by passing through possibly dozens of computers, which costs the other people who use those computers each a small bit of time and money. Therefore, avoid sending large volumes of email, especially in a short period of time.

If you must email a file larger than 64K in size, split it into several smaller parts and email them separately. Older mailer software frequently cannot handle single email messages longer than 64K. (This arcane limitation includes mailing headers, so be sure that your text is no larger than 60K in size.) Splitting a file offers another benefit as well. If a file transmission error occurs, you will only need to resend the segment of the file that was corrupted, not the entire file.

You can manually split a file into multiple parts using a text editor, or automatically using a text-splitting program. Some uuencoders will automatically split their output into multiple parts and will store each in a separate file. A third alternative is to use a utility such as the UNIX "split" program.

If you are emailing large graphics files, then you should uuencode and split them before sending them. A uuencoded file split into 950-line parts results in files that are 60K or less in size—just perfect for email. You can use the following commands to uuencode and split in two steps:

```
uuencode toshi.jpg toshi.jpg > toshi.uue
split -950 toshi.uue toshi.
```

or in one step using a pipe:

```
uuencode toshi.jpg toshi.jpg | split -950 - toshi.
```

In both cases, the split program will store the uuencoded output into 950-line files named *toshi.aa*, *toshi.ab*, *toshi.ac*, and so on. Each of these files is then emailed separately, with the part numbers in the mail header "Subject:" lines.

USENET News

USENET is a globally distributed bulletin board system that resides on the Internet, providing messaging and conferencing for Internet users. USENET does not have any central location or control. There is no system operator (sysop) who decides who can be a member, when you can have access, or how much information you can download per day. USENET is not a private information network used for the transportation of proprietary or secret data. Instead,

Ex.1007 / Page 1037 of 1158
APPLE INC.

information is placed on USENET for people to read, copy, rebut, and use in just about any way that they see fit.* USENET is a global community; what you and the people in your local community may consider to be "fair use" and "fair play" are not necessarily shared by your fellow USENET brethren on other parts of this great planet. Please practice tolerance and open-mindedness when browsing through the USENET global information village.

Information is "posted" to a USENET newsgroup in the form of an "article." An article is just a piece of email that you send to USENET using a program called a newsreader. A newsreader is a special client program that enables you to read and post articles to USENET newsgroups. Any computer running USENET news hub server software will also have available one or more different newsreader programs, such as rn or tin.

When you post an article to one or more USENET newsgroups, your local news hub will send your article to other computers that are also running news hub software, who in turn will send it on to others. Your article will spread out over the earth—via the regional telephone companies—to be stored and made available to all of the other many thousands of computers linked to USENET.

Newsgroups

The many thousands of newsgroups in USENET are organized as a hierarchy, with the most general grouping appearing first in the newsgroup name, followed by more specific subgroups. For example, graphics files are found in all the newsgroups in the hierarchy *alt.binaries.pictures*. We can be more specific and look for graphics of fine art in the *alt.binaries.pictures.fine-art* newsgroup. In doing so, we would see that this newsgroup contains three subgroups, one for the posting of original artwork, one for posting of scanned artwork, and one for the discussion of files posted on either of these two newsgroups. This branch of the hierarchy appears as follows:

```
alt.binaries.pictures
alt.binaries.pictures.fine-art
alt.binaries.pictures.fine-art.graphics
alt.binaries.pictures.fine-art.digitized
alt.binaries.pictures.fine-art.d
```

Don't think that fine art graphics is all that you'll find on USENET. As of January 1, 1996, no fewer than 124 *alt.binaries.pictures* subgroups were listed on USENET. Most of these groups are frequented by regular readers and some

* Note, though, that an implicit international copyright is assigned to all information posted to USENET by authority of the Berne Convention of 1989.

Ex.1007 / Page 1038 of 1158
APPLE INC.

receive dozens, if not hundreds, of posts a day. To give you an idea of what's available, here is a very abbreviated listing of the *alt.binaries.pictures.** hierarchy:

```
alt.binaries.pictures.animals
alt.binaries.pictures.anime
alt.binaries.pictures.art
alt.binaries.pictures.arts
alt.binaries.pictures.ascii
alt.binaries.pictures.astro
alt.binaries.pictures.babies
alt.binaries.pictures.black
alt.binaries.pictures.bodyart
alt.binaries.pictures.boys
alt.binaries.pictures.cartoons
alt.binaries.pictures.celebrities
alt.binaries.pictures.cemeteries
alt.binaries.pictures.children
alt.binaries.pictures.cops
alt.binaries.pictures.d
alt.binaries.pictures.erotic
alt.binaries.pictures.erotica
alt.binaries.pictures.fine-art
alt.binaries.pictures.fractals
alt.binaries.pictures.furniture
alt.binaries.pictures.furry
alt.binaries.pictures.girlfriends
alt.binaries.pictures.girls
alt.binaries.pictures.lingerie
alt.binaries.pictures.misc
alt.binaries.pictures.nude
alt.binaries.pictures.nudism
alt.binaries.pictures.personal
alt.binaries.pictures.photo-modeling
alt.binaries.pictures.rail
alt.binaries.pictures.supermodels
alt.binaries.pictures.tasteless
alt.binaries.pictures.teen-idols
alt.binaries.pictures.teen-starlets
alt.binaries.pictures.tools
alt.binaries.pictures.utilities
alt.binaries.pictures.vehicles
alt.binaries.pictures.voyeurism
```

The intellectual content of the images posted to these groups includes any person, place, object, or idea that may be captured with a camera, scanner, or graphics art program. You can find or request practically any image you might need or want. Such requests should only be posted to the discussion newsgroups (those ending in ".d").

Before you deem some subjects as "unsuitable" or even "illegal," remember that the Internet is a global information network. It is not owned by any one company that may set restrictions and policies, nor is it governed by only one

Ex.1007 / Page 1039 of 1158
APPLE INC.

social or cultural viewpoint. The Internet is truly the human race's first step at achieving a global community.* For a list of the newsgroups most often frequented by graphics programmers and others interested in graphics files, see "Internet Graphics Resources" later in this appendix.

Frequently Asked Questions (FAQ) Listings

FAQ listings are specialized newsgroup postings that contain a list of answers to questions commonly asked on a particular newsgroup. If you are new to reading a newsgroup, you should obtain and read a copy of the FAQ for that newsgroup before you post any questions to the group. This will give you an immediate answer to most common questions, lower the traffic on USENET, and reduce the need for regular readers of the newsgroup to answer the same basic questions over and over again.

Over the years, FAQs have evolved from simple Q&A listings to entire treatises on a particular subject, or even (unfortunately) infomercials for a particular product. FAQs have grown to the point where they have been published as books, and fierce battles have arisen when the freely available, yet copyrighted, material available in FAQs has been used without permission in for-profit ventures.

Nearly all FAQs may be found on the **.answers* newsgroups. The master newsgroup is *news.answers*, and all FAQs registered with the FAQ administrators at *rtfm.mit.edu* are posted there. Every major newsgroup category also has a **.answers* newsgroup as well (*comp.answers*, *rec.answers*, *alt.answers*, and so on). Each FAQ may also have one or more home newsgroups to which it is also posted. For example, the JPEG FAQ is posted to *news.answers*, *comp.answers*, and *comp.compression*.

FAQs are usually posted at regular intervals, which can be anywhere from once every two weeks to once every few months. The information in a FAQ may change each time the FAQ is posted or may hardly change at all, depending on the discretion of the FAQ's author and its subject matter.

FAQs are authored by people known as "FAQ maintainers." Each maintainer is just another member of the USENET community. Maintainers offer their time and service to the USENET community for free by accepting the responsibility of maintaining and posting one or more FAQs that may be used by tens of thousands of people. Maintainers are regarded as the de facto Internet experts on their FAQ's subject (even if they really are not). Maintainers must answer

* But note that as we write this, issues of censorship on the Net are being hotly debated in the governments and press of many countries, and by both free speech advocates and those who advocate restrictions of various kinds.

Ex.1007 / Page 1040 of 1158
APPLE INC.

questions, gather accurate and up-to-date information, present it in a format that is easily readable, and make their FAQs freely available to the USENET and Internet communities.

To familiarize yourself with USENET FAQs, look through the postings on the *news.answers* newsgroup. If graphics files are your interest, then you will want to check out James Murray's Graphics File Formats FAQ. This FAQ contains detailed information on graphics file formats, including where to find specifications, viewing programs, and source code, and on the problems and legalities of using certain formats. This FAQ is also available on the CD-ROM that accompanies this book (with links to the updated FAQ), and it also contains information on other graphics and file format–related FAQs.

USENET Posting Etiquette

There are a few rules to follow when you post articles to USENET. Because there are no USENET police, per se, no one will break down your door and haul you away in chains if you do not follow these rules. However, if you choose to ignore these rules of etiquette you are likely to receive everything from friendly advice to something close to death threats from your follow USENET newsgroup patrons.

- Do not post an article to a "pictures" newsgroup that is not a graphics file or graphics file archive. Post all questions to the appropriate *alt.binaries.pictures.***.d* discussion newsgroup.
- Do not post articles that do not follow the subject of the newsgroup. For example, if you have a picture of your pet cat that you wish to share, post it to a newsgroup that contains graphics of the same subject, such as *alt.binaries.pictures.animals.* Another newsgroup, such as *alt.binaries.pictures.vehicles*, would not be a good choice.
- When in doubt about where to post a graphics file, post it on the *alt.binaries.pictures.misc* newsgroup. When in doubt about where to post a question, post it to the *alt.binaries.pictures.d* newsgroup.
- Put an accurate description of the image in the post's "Subject:" line. For example, the line:

```
Subject: TOSHI.JPG (01/01) "Picture of my cat, Toshi" [640x480x256]
```

  is more descriptive than, and is preferred to:

```
Subject: my cat
```


Ex.1007 / Page 1041 of 1158
APPLE INC.

- Do not post more than 400K of articles to the *alt.binaries.pictures.** newsgroups a day. This limit may be in the form of eight 50K posts, four 100K posts, one 400K post, or whatever. The *pictures* newsgroups account for nearly 50 percent of the traffic on the Internet and 75 percent of the *alt.** newsgroup traffic; it costs time and money to transmit every byte of data that you post. Dumping many megabytes of data onto a newsgroup is like trying to pour 100 gallons of water down your kitchen sink. The water will only drain so fast, so it's better just to pour a little at a time.
- Email graphics files, rather than post them, if you only need to get them to a few people. Making them available on an FTP site (see the section on FTP below) is also an option.
- Don't attempt to compress GIF or JPG graphics files using a file archiving program, such as *pkzip* or *zoo*. Graphics files that contain compressed data cannot typically be reduced in size by further compression and may actually grow in size. If you need to store graphics files in an archive file format, then turn the compression feature off when you add the graphics files to the archive. For *pkzip*, use the '-e0' option; for *zoo*, use the 'f' option modifier.
- Use uuencode to convert all binary graphics files to ASCII text before posting. The uuencode scheme is the de facto standard used on USENET for converting binary data to ASCII text for posting. Although you can use mmencode, BinHex, btoa, and other encoders, not everyone has the programs required to decode the data created by these programs.
- If you are unsure about how to post an article, try posting a test article on one of the test newsgroups, such as *news.test*, *misc.test*, or *local.test*. You should also change the distribution of your posting to "local" to limit its travel on the Internet. Once you post your test article, you can use a newsreader to check whether your article was posted to the *test* newsgroup to see what it looks like. If you post a uuencoded graphics file, you can save and decode it to see whether your posting method worked. Never post to any newsgroup unless you know what you are doing!

For further information on posting to the *alt.binaries.pictures.** newsgroups, read Jim Howard's *alt.binaries.pictures* FAQ. This FAQ contains the official information on the *alt.binaries.pictures* newsgroups and their netiquette, gives a listing of graphics file viewers for all popular operating systems, and provides detailed troubleshooting information.

Ex.1007 / Page 1042 of 1158
APPLE INC.

The FAQ is available on the *alt.binaries.pictures.d* and *news.answers* newsgroups. You can get a copy via email by sending a message with the body:

```
send usenet/news.answers/pictures-faq/part1
send usenet/news.answers/pictures-faq/part2
send usenet/news.answers/pictures-faq/part3
```

to *mail-server@pit-manager.mit.edu.*

All introductory information on USENET can be found in the *news.announce.newusers* newsgroup. Be sure to read through all of the documents posted to this newsgroup before you attempt to post messages or create a new newsgroup. A list of all the available documents may be found in the "Welcome to USENET!" FAQ, which may be obtained by sending email to *mail-server@rtfm.mit.edu* with the body "send usenet/welcome/part1". All of these documents may also be found on the World Wide Web at:

http://www.smartpages.com/bngfaqs/news/announce/newusers/top.html

Mailing Lists

Mailing lists are a less visible, and often less costly, alternative to USENET newsgroups. Mailing lists are also distributed across the Internet, but via email rather than through a mechanism such as USENET. This access method is less costly in that only basic email service is required to join a mailing list, so users of information services such as CompuServe may join any Internet mailing list. Having access to USENET is not required.

A mailing list is similar to a newsgroup in that it is a discussion composed of contributions made by the mailing list members. Members of the list receive other members' contributions to the list as email and may contribute their own. You can save the email listings you receive and review them later. Most mailing list sites also archive their users' contributions, so you can join a list, request (or FTP) the last few weeks' worth of listings, and catch up on what you've missed.

To subscribe to a mailing list, you normally append "-request" to the name of the list in the Internet address name. For example, to subscribe to the mailing list *cogneuro@ptolemy.arc.nasa.gov,* send an email message to *cogneuro-request@ptolemy.arc.nasa.gov.* In the body of the message put the word "subscribe" followed by the name of the list, such as "subscribe cogneuro" (or "unsubscribe cogneuro" to remove yourself from the mailing list). You may contribute to the list by sending email to the *cogneuro@ptolemy.arc.nasa.gov* address. Whenever any other subscriber contributes to the list, you will receive email of that subscriber's listing.

Ex.1007 / Page 1043 of 1158
APPLE INC.

For a list of the mailing lists of most interest to graphics programmers and others interested in graphics files, see "Internet Graphics Resources" later in this appendix.

FTP Archives

FTP (File Transfer Protocol) is the way that files are moved from one location to another on the Internet. Using a program called an FTP client ("ftp" under UNIX), you can attempt to log on to any other computer on the network that is an FTP server and in this way gain access to its files.

FTP was originally designed to give people private access to their files on remote machines, and also to furnish them with an easy way to transfer files between their local and remote machines. People soon realized that FTP was also a very good method for giving others access to public files without requiring private system accounts. The files, called an FTP archive, can be accessed by anyone who logs into a general public account named "anonymous" with the password "ftp". Thus, the concept of anonymous FTP was born.

Today the Internet contains thousands of anonymous FTP sites, each open to share information and data files with the Internet community. The rules have changed only slightly from early times. Now most sites require you to give your email address as a password rather than "ftp".

Once you have started to explore the Net using FTP, you will quickly find that many of the more popular FTP archive sites can be quite difficult to log in to during what is referred to as "the peak hours." It may be difficult to imagine 300 people all using a single FTP server at the same time, but consider the tens of thousands of people who access the Internet at any time of the day or night, and realize that most of these people have FTP capability. Actually, once you do successfully log in, you will feel the crawl of an overloaded FTP server at peak hours and experience a 10-minute file transfer (one that would take only 10 seconds if you had only done it at 3 a.m. rather than at 3 p.m.).

FTP Etiquette

FTP etiquette is more or less about not being a file hog and behaving yourself on other people's systems. Here is some common sense advice:

- Limit the amount of time you spend logged into an FTP site. Other people with needs greater than your own may be trying to access the site.
- Don't download more than a few files. Each minute you spend online browsing and downloading files is being paid for by the owners of the FTP

Ex.1007 / Page 1044 of 1158
APPLE INC.

site on their telephone bill. Yes, the site is offered as a free service, but you shouldn't abuse it. (We compare it to sitting in a restaurant for hours writing a book on a notebook computer while guzzling gallons of free iced tea refills.)

- Don't upload files to the */incoming* directory that the administrators of the FTP site do not want. This includes commercial software and graphics files containing erotic images.
- When in doubt, read the *README* (or equivalent) file found in the FTP home directory, or displayed when you log in.

FTP servers are an excellent method of distributing graphics files and related information. Listings of FTP archive sites may be found in FAQ lists posted to the **.answers* newsgroups on USENET. You can also ask on relevant newsgroups about any additional FTP sites that may not be listed in a particular FAQ. For example, if you are interested in FTP sites that contain archives of graphics file format specifications, look in the Graphics File Formats FAQ; you'll find an up-to-date list of sites. For a copy of this list, as well as other FTP sites of interest to graphics programmers and others interested in graphics files, see "Internet Graphics Resources" later in this appendix.

Using an FTP client, you can log into any one of these FTP servers and browse and download their files.

Probably the only bad thing about FTP is that you don't get any fancy menus or searching capabilities. You navigate through the directory tree as you would in MS-DOS, VMS, or a UNIX shell account, reading through the occasional outdated *00index.txt*, *ls-lR*, or *README* file. Although there isn't a better way to search for files on a specific FTP server, there is a better way to search for files on all FTP servers, as we describe in the next section.

Archie

Archie search servers maintain file listings, updated monthly, of more than 1000 FTP sites containing nearly three million files. Any FTP server can register to be part of the Archie search service. The FTP site will then generate an *ls-lr* file containing a complete recursive listing of all the files on the archive. Archie will download these listings onto an Archie server and use them as a master index for remotely searching FTP sites.

Ex.1007 / Page 1045 of 1158
APPLE INC.

You can search for a specific filename or regular expression using the archie client command as follows:

```
archie -s cat -h archie.unl.edu
```

This command searches the Archie server at *archie.unl.edu* for any files with a name that contains the string "cat" and outputs a listing (the UNIX "ls -lR" listing, to be exact) of the FTP sites and pathnames of these files. It's always good netiquette to search the archie server geographically closest to you.

You can also perform an Archie search via email. This is often more convenient because the results of the search are then sent to you as email. To perform the Archie search shown in the previous example via email, send an email message to *archie@archie.unl.edu* with the body:

```
set search sub
find cat
```

Sending an email to an Archie server with only the word "help" in the body returns all of the information needed to use their archie client.

The World Wide Web (WWW)

The only medium that is more congenial than FTP and Archie for locating and distributing graphics files is the World Wide Web (WWW). For graphics files, the Web offers the same basic service as FTP archives. In fact, many sites that grant access to graphics file archives using an FTP server also grant access to the same archive using a Web server. In this case, you may browse directories and download graphics files using your Web browser much the same way you would using FTP.

The added beauty of the Web is its graphics capabilities. Web browsers allow you to view files as you download them. And with some extra work, the Web server maintainer can provide thumbnail images of each file embedded in the Web page, so you can see what the image looks like before you decide to download it.

Another wonderful feature available in Web browsers is the Web Search. You can perform a search across the Web on a key word or phrase to help you locate information. Performing a search using the key phrases "file format", "graphics file", or "file archive" will turn up hundreds of pieces of information, most of which will probably be of interest to you.

For a list of some of the Web sites of most interest to graphics programmers and others interested in graphics files, see "Internet Graphics Resources" later in this appendix.

Ex.1007 / Page 1046 of 1158
APPLE INC.

The following sections present some Web basics and describe ways you can use the Web both to retrieve graphics files and to embed them in your own Web pages.

Web Basics

The World Wide Web is a set of software programs called *Web clients* and *Web servers*. Servers contain information stored on disk, which they make available on the Internet. Clients (commonly called *Web browsers*) connect to the servers using a special networking protocol and retrieve the server's information. The networking protocol used by Web clients and servers is called HyperText Transfer Protocol (HTTP). HTTP is a protocol that does the following:

- Enables a client to connect to a server.
- Allows the client to request a piece of data (called a *Web resource*).
- Provides the server with the ability to transfer the requested data to the client.
- Enables the server to close the connection once the transfer has been completed.

Each time you click a new item on a Web page, your Web browser opens a connection to the specified Web server, makes a request for a Web resource (such as an image file), receives the resource (you hope!) from the Web server, and renders, displays, or plays the file on your local workstation.

The data that is delivered to your Web browser across the HTTP link is a MIME file. This file is a wrapper containing all of the data belonging to the requested resource.

A Web browser will immediately begin displaying the resource information as it is received. The information stored first in the MIME file is a textual description of the resource data. This description is known as the Web page and is stored using a format description language known as HyperText Markup Language (HTML). HTML is a standardized language developed specifically for applications-level communications between Web clients and servers. While HTTP negotiates the transfer of data between client and server, it does not care what that data actually is. HTML is the actual hypertext data being transferred between the clients and the servers.

HTML contains the hypertext structure and format of the Web page and the links to other Web resources. HTML is, however, only a framework built from text. Graphics, sound, video, and other non-textual data are not actually part

Ex.1007 / Page 1047 of 1158
APPLE INC.

of the HTML text. They are instead separate files that are referenced as MIME attachments. When the attachments are graphics data, that data is stored and transported using graphics file formats.

Graphics on the Web

For most of its history, the Internet has been a character-based communications channel. But in 1993, the National Center for Supercomputing Applications (NCSA) released Mosaic for X (v0.10) and changed the face of the Internet forever. Mosaic is a Web browser that is not only easy to use, but also supports the display of images embedded in the text. Prior to the release of Mosaic, Web browsers were character-based, able to display hypertext links only as blue or underlined words. Graphics data could be downloaded as files by a text-only Web browser but not displayed in a useful way.

Mosaic was the first graphical Web browser. Like character-based browsers, it was a piece of communications software, capable of interpreting and displaying HTML information. But Mosaic was built as an application under the graphical user interface (GUI) of MIT's X Window System. Mosaic presented an easy-to-use menu and a scrollable and resizeable window in which to display the retrieved text (the Web page), and it supported inline graphics. Now there are a number of other Web browsers, and Netscape Navigator at this point is the most popular. All graphics data on the Web is packaged using standard graphics file formats—the same ones in use throughout the rest of the computer industry. The three formats supported by all Web browsers today are GIF, JPEG, and XBM:

- GIF files are used for storing graphics containing up to 256 colors, such as line drawings, cartoons, and simple color and gray-scale graphics and photographic images.
- JPEG files are used for storing up to 16- and 24-bit photographic and photorealistic images.
- XBM files, a part of the legacy inherited from the original X Window implementation of Mosaic, is an ASCII file format used for storing monochrome data only.

GIF is the format used for most Web graphics. GIF was created by CompuServe specifically as a serial graphics data transmission format, exactly what is used by the Web. GIF uses an efficient, if legally encumbered, method of data compression that results in reasonably small files. (For information about these legal encumbrances, see "Legal Issues" below and also the discussion in the section called "LZW Legal Issues" in Chapter 9, *Data Compression*).

Ex.1007 / Page 1048 of 1158
APPLE INC.

GIF images are always displayed as inline graphics. That is, GIF images appear embedded in the text of the Web page itself. JPEG images, however, are displayed as external images in a separate window. An extension added to HTML 2.0 allows JPEG images to be displayed as inline images as well. This capability has been added to many Web browsers and it is expected to be a standard feature in HTML 3.0.

Graphics images may be an active part of your Web page, serving as the selectable items in a menu, or even as an entire menu itself. Other images may be the actual data you are retrieving, such as images from a database or an image file archive. And graphics images may only serve as pretty wallpaper—something to please the eye rather than to download or actively interact with.

Embedding Graphics Files Using HTML

Only a single line of HTML code is required to embed an inline image in the source code of a Web page. This is accomplished using the <IMG> tag in the HTML language. Here is an example of an IMG tag that displays an inline image stored in a file called *logo.gif*:

```
<IMG SRC="graphics/logo.gif">
```

The image stored in *logo.gif* will appear wherever this statement appears in the HTML text. You can set certain attributes associated with the IMG tag to control how the image is displayed—for example:

```
<IMG SRC="graphics/logo.gif" ALIGN="TOP" WIDTH="150" HEIGHT="75">
```

Looking at this statement from left to right we see the following:

- IMG indicates a tag containing the source file.
- ALIGN indicates the alignment of the image with respect to adjacent text on the page.
- WIDTH and HEIGHT indicate the size of the image, allowing the Web browser to fully display the text of the page before the image data is received.

In the above example, the file *logo.gif* resides on the same server as the Web page source. However, a graphic used in a Web page need not reside on the same Web server as the page that references it. For example, if the *logo.gif* file were on another Web site we could reference it as follows:

```
<IMG SRC="http://www.ora.com/graphics/logo.gif" ALT="Welcome!">
```

This example references an image file located on a machine with the named address *www.ora.com*. When this HTML statement is executed, an HTTP link is

Ex.1007 / Page 1049 of 1158
APPLE INC.

opened to the specified server, and the image resource is requested, transferred, and displayed.

This example also contains the ALT attribute, which specifies a string of text to display if the Web client cannot display graphics images (such as the text-based Lynx Web browser). Without the ALT attribute, the image would be represented by the default string "[IMAGE]". In the above example, the image stored in *logo.gif* would instead be represented by the text "Welcome!".

External images are referenced using the anchor <A> tag, as follows:

```
<A HREF="iaido.jpg">Fast draw and cut</A>
```

This example creates the underlined and highlighted line of text "Fast draw and cut" on the Web page. When you click on this line, the image file is retrieved from the Web server by your browser. This image may then be viewed or saved to disk as a file.

We can replace the text with a reference to an inline image. Now we have a thumbnail image that, if clicked on, will download the image:

```
<A HREF="iaido.jpg"><IMG SRC="iaido-thumbnail.gif"></A>
```

Of course, we can have both an inline image and text together:

```
<A HREF="iaido.jpg"><IMG SRC="iaido-thumbnail.gif">Fast draw and cut</A>
```

So far, we've shown only examples of linking to static images. You can view or save such images, but you don't really interact with them. The most common type of interactive graphics that you will find on a Web page are called *clickable* images.

Clickable graphics allow users to select a region of a graphics image just as they would select a push button or a menu item. This process is described as clickable because the selection is made using a point-and-click device, such as a mouse or another pointing device.

To make a graphics image clickable, you need to construct a pixel coordinate map of the image. This click map specifies the clickable regions of the image in pixel coordinates. Clickable regions may be described as circles, rectangles, polygons, or single points. Each clickable region is also assigned a URL (Uniform Resource Locator) to reference when that region is selected.

Once the map has been created and is in place, you need to add two tags to your HTML document that reference the image and its map:

```
<A HREF="/bin/cgi/maps/mainmenu.map">
<IMG SRC="/graphics/mainmenu.gif" ISMAP>
</A>
```


Ex.1007 / Page 1050 of 1158
APPLE INC.

These HTML statements associate the map and the graphics file and display the graphics. The user is then presented with a graphics image to be used as a point-and-click menu. But what about clients who access the Web site using non-graphical browsers? How can they access this menu? The simple answer is "They can't." But you can display a text menu that both graphical and non-graphical Web browsers can use:

```
<A HREF="/bin/cgi/maps/mainmenu.map">
<IMG SRC="/graphics/mainmenu.gif" ISMAP>
</A>
<P>
<A HREF="back.html">Back</A>|
<A HREF="home.html">Home</A>|
<A HREF="search.html">Search</A>
```

These additional statements will display a three-item menu under the graphic that will be visible and clickable by both graphical and non-graphical Web browsers. These statements are not physically attached to the IMG tag. They are placed to appear just after the image, so the eye naturally associates them with the menu image.

If you have ever wondered what the HTML code for a particular Web page looks like, you can select your Web browser's View/Source menu item and see for yourself. You can also save the text of any Web page by using the browser's File/Save As command. These are very useful tools for anyone who is learning HTML.

Web Browser Helper Applications

All Web browsers have the ability to display HTML text, inline GIF images, and external XBM and JPEG images. (Many browsers also support inline JPEG images.) In fact, a browser will display or play back any type of data that it is configured to recognize. Because many of the resources on the Web are made of data such as audio, video, and PostScript, it is therefore important to know how to configure your Web browser to recognize as many types of data as possible.

A Web browser may be initially configured in such a way that it does not display or play back any unrecognized information that it receives. You will, however, want to configure your browser to ask you what you want to do with any unrecognized data. That way, you will at least have the option of saving the data to a file. You should also have the option of having the data sent to an external program that is capable of displaying it or playing it back. Such external programs are called *helper applications.*

Ex.1007 / Page 1051 of 1158
APPLE INC.

When you configure a browser to use a helper application, you are creating an association between an external application and a type of data. Whenever the browser receives data of a recognized type, it automatically launches the associated helper application to display or play back the data. This transparent operation allows uninterrupted viewing of any HTML document that contains links to many different types of data.

The browser identifies all of the data it receives by the data's MIME type. Remember that all data received by your Web browser is in the form of a MIME mail message. Each section of data in the message has a tag called a *MIME type*, which describes the type of data stored in the message. The Web browser reads the MIME message and compares the MIME type tag in each section of data with a list of associated helper applications in a file. If the MIME type association is found, the data is sent to the associated helper application.

The MIME type of a file is assigned to the message by the Web server to which your client has connected. If you are accessing local files, or files via FTP, your browser will attempt to determine the data type from the file's extension. The number of data types supported by a Web server can be very extensive. For example, Table A-1 lists the MIME types supported by the Netscape client for X Window:

TABLE A-1: ***Netscape MIME Types (for X Windows)***

MIME Type/Subtype	Extension(s)	Description
application/fractals	fif	Fractal Image Format
application/mac-binhex40	hqx	Macintosh BinHex Archive
application/octet-stream	exe bin	Binary Executable
application/postscript	ai eps ps	PostScript Program
application/rtf	rtf	Rich Text Format
application/x-cpio	cpio	UNIX CPIO Archive
application/x-csh	csh	C Shell Program
application/x-dvi	dvi	TeX DVI Data
application/x-gtar	gtar	GNU Tape Archive
application/x-latex	latex	LaTeX Document
application/x-sh	sh	Bourne Shell Program
application/x-shar	shar	UNIX Shell Archive
application/x-stuffit	sit	Macintosh Archive
application/x-tar	tar	UNIX Tape Archive
application/x-tcl	tcl	TCL Program
application/x-tex	tex	TeX Document
application/x-texinfo	texi texinfo	GNU TeXinfo Document
application/x-troff	t tr roff	TROFF Document

Ex.1007 / Page 1052 of 1158
APPLE INC.

MIME Type/Subtype	Extension(s)	Description
application/x-troff-me	me	TROFF Document
application/x-troff-ms	ms	TROFF Document
application/x-troff-man	man	UNIX Manual Page
application/x-zip-compressed	zip	Zip Compressed Data
audio/basic	au snd	ULAW Audio Data
audio/x-aiff	aif aiff aifc	AIFF Audio
audio/x-wav	wav	WAV Audio
encoding/x-compress	Z	Compressed Data
encoding/x-gzip	gz	GNU Zip Compressed Data
image/gif	gif	CompuServe Image Format
image/ief	ief	IEF Image
image/jpeg	jpeg jpg jpe	JPEG Image
image/tiff	tiff tif	TIFF Image
image/x-cmu-raster	ras	CMU Raster Image
image/x-portable-anymap	pnm	PBM Image
image/x-portable-bitmap	pbm	PBM Image
image/x-portable-graymap	pgm	PGM Image
image/x-portable-pixmap	ppm	PPM Image
image/x-rgb	rgb	RGB Image
image/x-xbitmap	xbm	X Bitmap
image/x-xpixmap	xpm	X Pixmap
image/x-xwindowdump	xwd	X Window Dump Image
text/html	html htm	Hypertext Markup Language
text/plain	txt text	Plain Text
video/mpeg	mpeg mpg mpe	MPEG Video
video/quicktime	qt mov	Quicktime Video
video/x-msvideo	avi	Microsoft Video
video/x-sgi-movie	movie	SGI Video

Most books on HTTP and the World Wide Web contain extensive information on helper applications. A great deal of information is also available on the Web itself on this subject. (Perform a Web search on the key phrase "helper applications" and see for yourself.) We advise you to take a look at a Web page containing links to software archives that collect helper applications, such as:

http://home.mcom.com/assist/helper_apps/

Tips for Including Graphics on Web Pages

Here are some tips for using graphics on Web pages:

- Use graphics sparingly on each page. Too many images may make it difficult for users to navigate easily through your Web pages.

Ex.1007 / Page 1053 of 1158
APPLE INC.

- Keep your graphics files small to reduce the amount of time it takes to transfer them across a network. A maximum size of 100K is an acceptable limit.
- Make graphics files smaller by using compressed data formats, such as GIF and JPEG. Avoid uncompressed formats, such as XBM and PostScript Level 1 unless necessary.
- Remember that solid colors compress better than dithered or gradient fill patterns.
- Keep the size of the images as small as is convenient. Use 500×400 as a suggested maximum image size. Don't force the user to scroll or resize his or her browser window to see your graphics.
- Display the text on the page first to give the user something to read while the graphics are loading.
- Always specify the WIDTH and HEIGHT attributes of the IMG tag to allow a browser to format and display the page's text before all the inline image data has loaded.
- If a page's menu selections are graphical, store them in an interlaced format so it may be possible for the user to select menu items before the graphic has completely loaded.
- Access your Web page using a 9600bps link to get a reasonable worst-case access feel of the page. Trim or remove graphics to speed things up.

Here are some additional tips for increasing the performance of Web browsers:

- Use the fastest possible data link. A single user with a 28.8kbps modem will experience much quicker access than a user with a 14.4 or 9.6kbps modem, or many users jammed on to a 56kbps DS0 line.
- On slow data links, disable the loading of inline images to speed up your apparent access.
- Enable text and image caching. It is faster for your browser to read previously loaded text and images from memory or disk than to constantly request data from a server. Use a large cache size.
- Remember that you can never have too much memory installed in your computer or workstation.

Ex.1007 / Page 1054 of 1158
APPLE INC.

Legal Issues

Under U.S. copyright law, and in other nations through the authority of the Berne Convention, all graphics and images are implicitly copyrighted upon creation. This means that the graphics images you see on a Web page are not freely available, unless explicitly permitted by the owners of the graphics files. Remember this the next time you fire up your screen capture application to get a sample of the wallpaper or the logo on somebody else's Web page. (See the section called "Trademarks, Patents, and Copyrights" in Chapter 8, *Working With Graphics Files*, for additional information.)

Note also that the LZW compression algorithm used to create all GIF files currently requires a licensing fee for its use. See the section called "LZW Legal Issues" in Chapter 9, for more details.

Internet Graphics Resources

This section contains references to a variety of newsgroups, mailing lists, FTP archives, and World Wide Web sites that will be helpful to graphics programmers and others who need information and graphics resources.*

General Resources

The prime source of graphics information on USENET is the FAQ for the *comp.graphics* newsgroup. Information on file formats, image processing and analysis, books and journal articles, and graphics and imaging software packages is updated monthly in this FAQ.

The image-processing FAQ found in the *sci.image.processing* newsgroup gives a list of Macintosh image-processing information available via FTP, USENET, email, telephone, and the postal service.

The Pilot European Image Processing Archive (PEIPA) is a repository and distribution service for software, digests, and newsgroup archives concerned with image processing, analysis, manipulation, generation, and the display of graphics. The PEIPA FTP site (*peipa.essex.ac.uk*) contains the British Machine Vision Association (BMVA) and Society for Pattern Recognition newsletter (directory *ipa/digest/bmva*), the International Association for Pattern Recognition newsletter (directory *ipa/digest/IAPR*), and the archives for the pixel (*ipa/digests/pixel*) and Vision-List (directory *ipa/digests/vision-list*) mailing lists.

* Our thanks to Shari L.S. Worthington for her contribution to a similar summary that appeared in the first edition of this book. Many of the Internet resources listed here were included in her article, "Imaging on the Internet: Scientific/Industrial Resources," which appeared in *Advanced Imaging*, February 1994.

Ex.1007 / Page 1055 of 1158
APPLE INC.

Newsgroups

alt.3d
3D imaging

alt.binaries.multimedia
An excellent source of multimedia files

alt.binaries.pictures.misc
An excellent source of graphics files

alt.comp.compression
An alternative to *comp.compression*

alt.graphics
An alternative to *comp.graphics.misc*

alt.graphics.pixutils
Discussion of image manipulation software

alt.image.medical
Medical image processing and analysis

comp.compression
Data compression algorithms and theory

comp.compression.research
Discussions about data compression research, including JPEG and MPEG

comp.dsp
Digital signal processing and audio formats

comp.fonts
Font file formats

comp.graphics.misc
Computer graphics, art, animation, image processing, and file formats

comp.graphics.animation
Technical aspects of computer animation

comp.graphics.research
Technical discussion on latest graphics research

comp.graphics.visualization
Information on scientific visualization

Ex.1007 / Page 1056 of 1158
APPLE INC.

comp.infosystems.gis
Geographic information systems

comp.infosystems.www.authoring.images
Information on using image files on Web pages

comp.multimedia
Interactive multimedia technologies

comp.speech
Speech processing topics

comp.sys.mac.scitech
Macintosh scientific and engineering applications

comp.sys.sgi.graphics
Graphics software and issues on SGI systems

sci.data.formats
Modeling, storage, and retrieval of scientific data

sci.image.processing
Scientific image processing and analysis

Mailing lists

arachnet@uottowa
Association of electronic lists and journals on electronic publishing

To subscribe: send email to *dkovacs@kentvm.kent.edu*

nih-image@soils.umn.edu
Discussion of the NIH Image software package for the Macintosh

To subscribe: send email to *listserv@soils.umn.edu* with the body "subscribe nih-image *yourfirstname yourlastname*"

pixel@essex.ac.uk
British Machine Vision Association newsletter for all image processing, machine vision, pattern recognition, remote sensing, and related topics

To subscribe: send email to *pixel-request@essex.ac.uk*

vision-list@ads.com
Computer vision discussion of algorithms and techniques

To subscribe: send email to *vision-list-request@ads.com* with the body "subscribe vision-list"

Ex.1007 / Page 1057 of 1158
APPLE INC.

ximage@expo.lcs.mit.edu
Image processing using the X Window System

To subscribe: send email to *ximage-request@expo.lcs.mit.edu* with "subscribe ximage" in the body of the message

FTP Archives

ftp://avalon.chinalake.navy.mil/pub/format_specs/
3D object repository; archive of graphics file format specifications

ftp://avalon.viewpoint.com/pub/format_specs/
File format archive

ftp://ftp.cc.monash.edu.au/pub/graphics.formats/
File format archive

ftp://ftp.cica.indiana.edu
Clearinghouse for Microsoft Windows applications, tips, utilities, drivers, and bitmaps

ftp://ftp.ncsa.uiuc.edu/misc/file.formats/graphics.formats/
National Center for Supercomputing Applications. Publicly available software for image processing, data analysis, and visualization for the Macintosh, PC, and UNIX platforms. Archive of graphics file specifications.

ftp://ftp.std.com/obi/Standards/Graphics/Formats/
File format archive

ftp://ftp.sdsc.edu/
San Diego Supercomputer Center (SDSC). Image Tools, ImageTyper, and Interactive Color Tutorial. Sound and image file archive.

ftp://ftp.switch.ch/mirror/simtel/msdos/graphics/
File format archive

ftp://ftp.uu.net/graphics/
ftp://ftp.uu.net/doc/literary/obi/Standards/Graphics/Formats/
UUNET archive. Large collection of graphics and imaging software including the *comp.graphics.misc* archive.

ftp://ftp.wust1.edu/doc/graphic-formats/
File format archive

ftp://mirrors.aol.com/pub/pc_games/programming/formats/
File format archive

Ex.1007 / Page 1058 of 1158
APPLE INC.

ftp://mom.spie.org/
International Society for Optical Engineering. Proceedings, programs, and information on Technical Working Groups, including electronic imaging.

ftp://peipa.essex.ac.uk/ipa/info/file.formats/
ftp://peipa.essex.ac.uk/ipa/khoros/
Pilot European Image Processing Archive. Contains many images and software packages, including the Khoros GUI development environment. Also contains a modest collection of graphics file formats specifications.

ftp://ra.nrl.navy.mil/
Naval Research Laboratory Research Computation Division Visualization Laboratory. Mostly Macintosh programs for chemistry, biology, math, imaging, AI, data acquisition, etc.

ftp://photo1.si.edu/images/
Smithsonian Institution photoimage archives

ftp://sumex-aim.stanford.edu/info-mac/
Large repository of Macintosh software

ftp://sunsite.unc.edu/pub/multimedia/
University of North Carolina. Information on multimedia images, video, and sound. Graphical image collection.

ftp://telva.ccu.uniovi.es/pub/graphics/file.formats/
Archive of graphics file format specifications

ftp://titan.cs.rice.edu/public/graphics/graphics.formats/
Large archive of graphics software and graphics file format specifications and information

ftp://wuarchive.wustl.edu/graphics/graphics/packages/
ftp://wuarchive.wustl.edu/graphics/graphics/mirrors/avalon/format_specs/
Mirror site for most major FTP archive sites. Also contains a large archive of graphics and images for math and life science educators, as well as file formats.

ftp://x2ftp.oulu.fi/pub/msdos/programming/formats/
File format archives

ftp://zippy.nimh.nih.gov/pub/nih-image/
National Institutes of Health archive for the Macintosh NIH Image and related publicly available programs.

Ex.1007 / Page 1059 of 1158
APPLE INC.

Web sites

http://www.w3.org/hypertext/DataSources/Journals.html

Electronic journals and newsletters

Chemical and Biomedical Imaging Resources

The following resources contain information on the sciences of chemistry, biology, biomedicine, and nuclear medicine. The Biomedical Computer Laboratory (BCL) is supported by the National Institute of Health's (NIH) National Center for Research Resources (NCRR). The BCL promotes the application of advances in computer science and engineering, mathematics, and the physical sciences to research problems in the biological and medical fields by supporting the development of advanced research technologies. Emphasis is on quantitative imaging, including PET image reconstruction, computational optical-sectioning microscopy, shape modeling and segmentation, electron-microscopic autoradiography (EMA), image acquisition and quantitative analysis of DNA electrophoretic gels and autoradiograms, and parallel processing.

Mailing lists

cogneuro@ptolemy.arc.nasa.gov

Cognitive science and neuroscience discussion

To subscribe: send email to *cogneuro-request@ptolemy.arc.nasa.gov* with the body "subscribe cogneuro"

medimage@polygraf

Medical imaging discussion

To subscribe: send email to *listserv%polygraf.bitnet@mitvma.\%mit.edu** with the body "subscribe medimage"

nucmed@uwovax.uwo.ca

A discussion of nuclear medicine and related issues, including the format of digital images

To subscribe: send email to *listserv@largnet.uwo.ca* with the body "subscribe nucmed *yourfirstname yourlastname*"

listproc@u.washington.edu

Radiology Special Interest Group

To subscribe: send email to *listserv@u.washington.edu* with the body "subscribe radsig *yourfirstname yourlastname*"

Ex.1007 / Page 1060 of 1158
APPLE INC.

listserv@listserv@ksuvm.ksu.edu

Veterinary medicine computer assisted instruction

Topics include imaging, expert systems, and LIMs.

To subscribe: send email to *listserv@ksuvm.ksu.edu* with body "subscribe vet-cai-l *yourfirstname yourlastname*"

FTP archives

ftp://ftp.sdsc.edu/pub/sdsc/

Computational chemistry and biology information

ftp://sunsite.unc.edu/pub/academic/

University of North Carolina. Information on astronomy, biology, chemistry, molecular modeling, geology, and GIS.

ftp://wubcl.wustl.edu/pub/

Biomedical Computer Laboratory. Quantitative imaging data.

Telnet archives

130.199.112.132

Nuclear Data Center

Login: nndc

Web Sites

http://www.ozemail.com.au/~mmajor/cci/maillist.html

Agricultural-related mailing lists

http://www.seas.gwu.edu/seas/institutes/medimage/

Medical imaging, George Washington University

http://www.ashe.miami.edu/ab/medweb.html

MedWeb: Adam's Guide to medical resources on the Internet images

http://http2.sils.umich.edu/Public/nirg/nirg1.html

Neurosciences Internet resource guide

http://ivory.lm.com/~nab/neurolist.html

Neuroscience mailing lists

http://www.cs.cmu.edu/Web/Groups/CNBC/other/other-neuro.html

Other Internet neuroscience resources

Ex.1007 / Page 1061 of 1158
APPLE INC.

http://johns.largnet.uwo.ca/nucmed/index.html
Nuclear medicine resources

http://bpass.dentistry.dal.ca/nucmed.html
Nuclear medicine resources on the Internet

http://netvet.wustl.edu/vetmed.htm
Veterinary medicine

Meteorological, Oceanographic, and Geophysical Imaging Resources

The latest weather data sources are located in the FAQ for the newsgroup *sci.geo.meteorology*. The FAQ contains current information on weather satellite data and images, and on meteorological, oceanographic, and geophysical research data. Information on Geographical Information Systems (GIS) may be found in the GIS-L mailing list, the newsgroup *comp.infosystems.gis*, or on the FTP site *csn.org*.

Newsgroups

comp.infosystems.gis
Information on all aspects of Geographical Information Systems (GIS)

sci.geo.meteorology
Discussion of meteorology

Mailing lists

GIS-L@ubvm.cc.buffalo.edu
Forum for the discussion of Geographical Information Systems

To subscribe: send email to *listserv@ubvm.cc.buffalo.edu* with the body "subscribe GIS-L *yourfirstname yourlastname*"

FTP archives

ftp://ftp.csn.org/COGS/
Geologic, GIS, mapping, earth science software and resources for the PC and Macintosh

ftp://ics.uci.edu/honig/
Synthetic stereo satellite images of earth

ftp://liasun3.epfl.ch/pub/weather/
Weather map of England, Europe, and the earth in GIF format

Ex.1007 / Page 1062 of 1158
APPLE INC.

ftp://vmd.cso.uiuc.edu/wx/
Weather satellite images of North America and Surface Analysis weather maps in GIF format

Telnet archives
128.175.24.1
Ocean Information Center. Data sets related to oceanography.

Login: `info`

Web Sites
http://www.lib.berkeley.edu/EART/listservers.html
Earth sciences listservers

http://www.hdm.com/gis3.htm
GIS net sites

http://rigel.csuchico.edu/ores.html
Online resources for earth sciences

Astronomy and Space Exploration Imaging Resources

The Internet abounds with information on astronomy, astrophysics, and space exploration. Archives contain thousands of images collected from telescopic, satellite, and spacecraft data.

The Space Telescope Electronic Information System (*stsci.edu*) is a very comprehensive resource for all astronomical information, accessible both via the Web and FTP. It includes many spacecraft and Hubble Space Telescope images.

The NASA archive *ftp.gsfc.nasa.gov* contains not only a huge collection of images but also references to other resources, such as UUNET, the Washington University archives, and the Lawrence Berkeley Labs.

Newsgroups
alt.sci.astro
An astronomy discussion

alt.sci.astro.figaro
Figaro data-reduction package discussion

Ex.1007 / Page 1063 of 1158
APPLE INC.

alt.sci.astro.aips
Discussions on the Astronomical Image Processing System (AIPS)

sci.astro
An astronomy discussion

sci.astro.fits
Issues related to the Flexible Image Transport System (FITS)

sci.astro.hubble
Hubble Space Telescope data

sci.astro.planetarium
A planetarium-oriented discussion

Electronic journal
COSMIC UPDATE
Information on new NASA software for astronomy and space exploration

To subscribe: send email to *service@cossack.cosmic.uga.edu.*

FTP archives
ftp://ames.arc.nasa.gov/pub/SPACE/
NASA/Ames Archives. Data files, GIF images, and NASA press releases and indexes.

ftp://suncub.bbso.caltech.edu
Big Bear Solar Observatory. Solar full-disk and high-resolution images.

ftp://ftp.gsfc.nasa.gov/pub/images/
NASA images in GIF, JPEG, PostScript, Sun Raster, and X Bitmap formats. Links to LBL, Washington University, and UUNET.

ftp://wuarchive.wustl.edu/graphics/magellan/
Planetary Data System (PDS) Geosciences Node and Magellan spacecraft images

Telnet archives
envnet.gsfc.nasa.gov
EnviroNET (Space Environment Information Service).. Space data from NASA and the European Space Agency.

Username: `envnet`
Password: `henniker`

Ex.1007 / Page 1064 of 1158
APPLE INC.

nssdc.gsfc.nasa.gov
nssdca.gsfc.nasa.gov
National Space Sciences Data Center. Archive for space and earth science researchers.

Login: `nodis`

For Further Information

For information about the Internet and the various types of Internet services, see:

Harrison, Mark, *The USENET Handbook*, O'Reilly & Associates, Sebastopol, CA, 1995.

Krol, Ed, *The Whole Internet User's Guide & Catalog*, second edition, O'Reilly & Associates, Sebastopol, CA, 1994.

Liu, Cricket, Jerry Peek, Russ Jones, Bryan Buss, and Adrian Nye, *Managing Internet Information Services*, O'Reilly & Associates, Sebastopol, CA, 1994.

Musciano, Chuck and Bill Kennedy, *HTML: The Definitive Guide*, O'Reilly & Associates, Sebastopol, CA, 1996.

Niederst, Jennifer with Edie Freedman, *Designing for the Web: Getting Started in a New Medium*, O'Reilly & Associates, Sebastopol, CA, 1996.

The Web itself is crammed full of information about creating and using inline graphics (try performing a Web search using the key phrase "Web graphics"). The following Web resources contain very good information on Web page graphics, but they are by no means the only available sources:

http://hakatai.mcli.dist.maricopa.edu/tut/
Writing HTML: A Tutorial for Creating Web Pages
A complete Web page construction tutorial courtesy of the Maricopa Center for Learning and Instruction. Tutorial 7 contains the inline graphics information.

http://www.w3.org/CERN/Divisions/SL/off-comp/links.html
Jaap's Link Collection
Links to many useful Web pages for HTML development.

Ex.1007 / Page 1065 of 1158
APPLE INC.

http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html
A Beginner's Guide to HTML

Contains a nice section on inline images.

http://home.netscape.com/assist/net_sites/html_extensions.html
Extensions to HTML 2.0

Information on advanced features of Web graphics.

http://www.cs.indiana.edu/docproject/mail/mime.html
What is MIME?

An introduction to MIME.

http://www.yahoo.com/Computers/World_Wide_Web/Programming/Icons/
Yahoo Icons and Bitmaps

Links to numerous icon and bitmap archives. All suitable for use as Web graphics.

http://dragon.jpl.nasa.gov/~adam/transparent.html
The Transparent/Interlace GIF Resource Page

A terrific source of information for using GIF files to store Web page graphics.

http://www.galcit.caltech.edu/~ta/tgif/transgifnotes.html
The Creative Internet: Transparent GIF Info

More info on transparent GIFs and GIF software tools.

http://www.softlab.ntua.gr/www/inline-images-faq.html
WWW Inline Images FAQ

Frequently asked questions about inline images on the Web.

http://sunsite.unc.edu/boutell/faq/www_faq.html
WWW FAQ

Tom Boutell's excellent FAQ on the World Wide Web.

http://melmac.corp.harris.com/transparent_images.html
Transparent Background Images

More info on transparent GIFs and how to create them.

http://sd-www.jsc.nasa.gov/web_formats.html
Recommended File Formats for WWW Documents

Information on file formats used on the Web.

Ex.1007 / Page 1066 of 1158
APPLE INC.

Graphics Files and Resources on the Commercial Services

This appendix provides some pointers to information on graphics files and resources on the major commercial online information services, CompuServe and America Online, as well as on a variety of bulletin board systems (BBSs). The other services—Prodigy, Delphi, BIX, GEnie, and others—also offer graphics information, though to a lesser extent.

Some of the graphics files and programs available on the commercial services and the BBSs eventually make their way to the Internet file archive sites, but little of the message traffic does. Worldwide, traffic through CompuServe and the private BBSs has been estimated to be at least as large as the volume of traffic through the entire Internet. The number of files on these services is many times greater than on the Internet.

Despite the plethora of valuable information on the commercial services, it is unfortunately often difficult to find what you want; CompuServe, for example, hosts more than one thousand forums. Another disadvantage of the commercial services is that it's not possible to redistribute materials provided by vendors for their own customers' use.

CompuServe

As we've mentioned, CompuServe contains an enormous amount of information, images, and discussion forums of interest to graphics programmers and others who need information about graphics files and formats. In particular, CompuServe is the place to go for system-specific programming information in the PC and Macintosh worlds. Most major companies provide support through CompuServe, so if you know what graphics information you want, CompuServe is the first place to look.

Ex.1007 / Page 1067 of 1158
APPLE INC.

We won't try to tell you how to use CompuServe in any detail but will simply point you to the most helpful resources now available. Because things change so rapidly, initially you should find out what graphics information is currently available. Do a search on "graphics" or "computer graphics" or "Macintosh graphics" or "3D rendering" or whatever your area of interest might be. CompuServe will return a list of the forums that contain information that might potentially be of interest to you. A recent search on the word "graphics" turned up 189 separate forums. A search on "graphics programming" turned up 18.

Here are just a few forums that might be of interest. Remember that the forums and areas of interest change rapidly, and new forums are always being added, so check frequently for new resources.

Amiga Tech Forum
Animation Vendor A Forum
Atari ST Prod. Forum
CADD/CAM/CAE Vendor Forum
Computer Animation Forum
Desktop/Electronic Publ.
Graphics B Vendor Forum
Graphics Corner Forum
Graphics Developers Forum
Graphics Forums
Graphics Support Forum
Graphics Vendor Forum
IBM OS/2 Vendor Forum
Imaging Vendor A Forum
MIDI A Vendor Forum
MS Windows AV Forum
Mac A Vendor Forum
Macintosh Multimedia Forum
Multimedia A Vendor Forum
PC Vendor G Forum
Windows 3rd Party A Forum
World of Graphics Forum

Each graphics forum has its own associated keyword. To speed things up or to access a forum from a command-line-oriented system or terminal, you can type the keyword (e.g., GO GRAPHDEV gets you to the Graphics Developers Forum and GO GRAPHSUPPORT gets you to the Graphics Support Forum).

Within each forum, you'll find all kinds of resources. There are libraries containing GIF and JPEG images that you can view and download, as well as libraries containing free software and shareware, also available for downloading. There are discussion forums where you can exchange mail or post news or opinions about current events in the graphics world, such as the LZW/GIF patent issue and the development of the new PNG file format. There are also other resources, such as conferences with particular people or vendors, special news stories, and more.

The GRAPHSUPPORT forum is a particularly important source of graphics information on CompuServe. It is the central distribution point of GIF-related materials (recall that GIF was designed, and continues to be maintained, by CompuServe engineers). Because of the widespread popularity of GIF, the

Ex.1007 / Page 1068 of 1158
APPLE INC.

GRAPHSUPPORT forum has become an attractive place to post more general graphics-related information and queries and to upload files, including images and program-related materials.

Here is what GRAPHSUPPORT offered at the time we went to press. Remember that this menu is subject to change at any time.

Forum Information	JPEG Tools
Online Viewing	Developers' Den
Graphic Viewers	Misc. Util & GIF Tools
Format Conversion	Non-GIF Software
Paint/Draw Programs	Standards and Specs
Digitizing Hardware	Copyright & More!
Video Adapters	'Go Graphics' Lab!
Printing Graphics	GIF/LZW Discussion
Publishing Projects	PNG Developments
GIF Tools	Ghostscript
Animation Players	Last Chance!
Compression/UU	

You'll find a similar depth of information in many of the other graphics forums.

America Online

If you are using America Online (AOL), you will also find many forums containing graphics information. Some include images, applications, and file format specifications available for downloading. The focus of graphics information on AOL is the Graphic Arts Program (keyword GRAPHICS), a repository of graphics information. There are two versions of the Graphic Arts Program—one for the PC and one for the Macintosh. You can easily navigate from one to the other. A recent look at the PC Graphic Arts Program showed the following categories of resources and a total of 92,000 messages in 20 categories:

About the PC Graphics Forum	Suggestion Box
Weekly Forum News	Software Libraries
Meet the Graphics Staff	Message Boards
Graphics Help and Info	Conference Center
Forum Conference Schedule	Special Interest Groups
Artists' Spotlight	World Wide Web

Ex.1007 / Page 1069 of 1158
APPLE INC.

Special Interest Groups	Feature
Recommended Viewers	Resource Center
Free Uploading	Help Desk
PC Vendor Database	

Special Interest Groups branches out to many different topics, vendors, and products (e.g., Autodesk, CorelDraw!, Virtual Reality, 3D Rendering). *Software Libraries* includes such categories as Clip Art, GIF Images, and Animations). Within each category there are further subcategories (e.g., *3D Rendering* contains 3D Models, Free Uploading, and Rendered Images). *Graphics and Presentation Companies* includes categories for Adobe, Apple, Disney, Island Graphics, SGI, and many more vendors.

Bulletin Board Systems

In addition to the Internet and the large commercial services, there are private bulletin board systems (BBSs) dedicated to discussions and distributions of information for specific hardware platforms (e.g., PC, Macintosh, or Amiga), corporations (e.g., Adobe, Apple), and software products (e.g., CorelDraw!). Some estimates put the total number of BBSs at as many as 60,000. The largest amount of traffic in these BBSs is in files, but the FidoNet, RIME, and ILink networks provide conferences of services akin to the Internet newsgroups described in Appendix A, *Graphics Files and Resources on the Internet*, or the CompuServe and AOL conferences described earlier in this appendix.

Here is a sampling of private BBSs of special interest to graphics programmers; for each, the network technology (e.g., Fido) is specified.

- AtariNet (uses Fido and includes discussions about Atari home computers and related topics). Conference topics include Atari graphics (ATARI.GRAPHICS).
- FidoNet (uses Fido and includes a host network and many subnets). Conference topics include Amiga video graphics and desktop video (AMIGA_VIDEO).
- Inter-Comm Network (ICN) (uses Fido and includes many echomail conference areas). Conference topics include image processing and graphics (IMAGEPRO) and CD-ROM technology and multimedia (CDROM).
- ILink (uses QWK and includes many professional forums, including those from AT&T, Novell, and Hayes). Conference topics include graphics (GRAPHICS) and CD-ROM technology (CD-ROM).

Ex.1007 / Page 1070 of 1158
APPLE INC.

- Magnet (uses Fido and is aimed at the computer hobbyist). Conference topics include graphics (GRAFIC).
- RIME (uses Postlink). Conference topics include graphics (GRAPHICS).
- SourceNet (uses Fido/QWK and is aimed particularly at programmers). Conference topics include compression software (COMPRESSION) and the various hardware platforms and operating systems (e.g., MAC, WINDOWS, UNIX, HARDWARE).
- WildNet (uses Fido/QWK and contains a broad range of topics, both technical and personal). Conference topics include application areas (e.g., COREL DRAW), CD-ROM technology (CD-ROM), and graphics software and files (GRAPHICS).

Ex.1007 / Page 1071 of 1158
APPLE INC.

Installation and Setup

This book includes a CD-ROM (Compact Disc–Read-Only Memory) containing a great deal of information that will help you understand, use, convert, manipulate, and otherwise make sense of the more than 100 graphics file formats described in the book. The CD-ROM contains the full text of the book, file format specifications from vendors, contributed software, sample code and images, and our own GFF product software.

Our GFF software provides browser support for searching, image maps, and forms. Usually, these are functions that are performed by a *server* (for example, if you are viewing a page at *www.ora.com*, that's the server). Since you view the *Encylcopedia* on your local machine, without installing it on a server, there is no way to provide searching, forms, or other sorts of dynamic behavior, without a helper application for the browser. GFF runs silently in the background; you'll probably forget it's even there.

This appendix describes how to install and set up the GFF software so you'll be able to browse the book, obtain the files you need, run demos, and get up-to-date information about graphics file formats.

The information provided on the CD-ROM—voluminous as it is—is only the beginning. If you have an Internet connection, you can link to the GFF Web Center to find updates to specs, software, and the book itself. At that site, you'll also be able to view relevant Frequently Asked Questions (FAQ) listings and graphics news, and request additional information.

For the most up-to-date and accurate information about GFF, check out GFF Web Center. The information that follows was written during the final testing stages of the GFF application; it is always possible that some last-minute changes may have been required. In addition to reading this appendix, be sure to read the informational files on the CD-ROM for specific, updated instructions.

Ex.1007 / Page 1072 of 1158
APPLE INC.

Using the CD-ROM

CD-ROMs provide a durable, cost-effective distribution medium that is becoming the standard way to distribute operating systems, third-party software, and other types of information. If you are new to CD-ROMs, you will need to learn about some specific CD-ROM issues before you can use the information on the CD-ROM that accompanies this book.

The CD-ROM Format

Because graphics file formats are of interest to users of many different platforms—by their very nature, these formats are meant for interchange between platforms—we have chosen to develop a multi-platform CD-ROM for inclusion in this book. This CD-ROM is in a hybrid ISO 9600/Macintosh HFS format. It conforms to the ISO 9660 and Apple HFS standards. Virtually all CD-ROM drives support the ISO 9660 standard, although there are some differences in how the files are read and presented, as we'll describe below. (All Macintosh drives support the HFS standard with no significant variation.)

ISO 9660 is the standard approved by the International Standards Organization (ISO) in 1987. This standard is adapted slightly from the original standard proposed by CD-ROM application developers and computer vendors. That original standard was known as the High Sierra format. You will sometimes see the terms High Sierra and ISO 9660 used interchangeably, but they are actually slightly different. ISO 9660 is the standard that will be supported from now on, although ISO 9660-compliant CD-ROM drives will continue to be able to read disks created in the older High Sierra format.

The ISO 9660 standard has the major advantage that it is relatively consistent across platforms. It does, however, impose a few limitations on files and directories:

- Directories may not be more than eight levels deep.
- Directory names may contain up to eight characters with no extensions. The name may consist only of the characters A-Z (or a-z, but cases may not be mixed) and the digits 0-9.
- All file and directory names are monocase. Depending on the driver program associated with your particular CD-ROM drive, they will appear as either all uppercase or all lowercase. For example, the *README.TXT* filename may be displayed as either *README.TXT* or *readme.txt*.

Ex.1007 / Page 1073 of 1158
APPLE INC.

- A filename may contain up to eight characters, with an extension of up to three characters. Filenames (both the name and the extension) may consist only of the characters A-Z (or a-z, but cases may not be mixed), the digits 0-9, and the underscore (_).

With some CD-ROM driver programs, you will notice that a period is appended to the filename (if the filename does not have an extension). You will also notice that a semicolon, followed by a version number, is appended. For example, the *README.TXT* filename might be displayed as any of the following, depending on which system and CD-ROM driver you use:

README.TXT
README.TXT;1
readme.txt
readme.txt;1

For PC users, ISO 9660 will have familiar characteristics because it is basically an MS-DOS format (for example, the familiar 8-character filenames with 3-character extensions).

For UNIX users, ISO 9660 will look quite different. Lengthy UNIX filenames and multiple extensions have had to be changed to conform to the ISO 9660 standard.

For Macintosh users, this CD-ROM will look like a normal Macintosh file system.

Organization of the CD-ROM

The following directories are on the CD-ROM:

/gff

Contains the full text of the book, sample images, code and vendor specifications. The book is available in two separate versions, a graphics version, in */gff/graphics*, and a text-only version in */gff/textonly*.

Note that using the HTML files in these directories requires the GFF application; if you simply point your browser at these files, using your browser's option for opening local documents, for example, some links will not work. See below for instructions on customizing your view of GFF.

/Mac

Contains the Macintosh implementation of GFF and the Spyglass Enhanced Mosaic browser.

Ex.1007 / Page 1074 of 1158
APPLE INC.

/pc

Contains the Windows implementations of GFF and the Spyglass Enhanced Mosaic browser. Use the Windows 95 implementation (under */pc/win95*) of GFF for Windows 95 and Windows NT. Use the Windows 3.1 implementation (under */pc/win31*) for Windows 3.1.

/unix

Contains the UNIX implementations of GFF. The directories under */unix* are divided by platform. Each directory contains the appropriate GFF implementation and the Spyglass Enhanced Mosaic browser. Consult the *README* and *INSTALL* files in these directories for more information.

/software

Contains the third-party contributed software. There are separate directories below */software* for each platform. The UNIX applications are provided in source form and are not pre-compiled. You can navigate to these directories through GFF or directly as you wish.

Any late-breaking information about GFF is stored in the *README* file in the appropriate directory. Please read this file before you install GFF.

Installing GFF

Installing GFF is straightforward. Follow the rules for your platform, as described in the following sections.

Installing GFF on a Windows System

In the descriptions that follow, wherever you see a reference to the drive `D:`, substitute the drive letter of the CD-ROM drive on your system containing GFF.

If you do not have Spyglass Enhanced Mosaic version 2.11 or later installed on your system, begin by installing it. Run:

```
D:\PC\MOSAIC\DISK1\SETUP.EXE
```

To install GFF under Windows 95 or Windows NT, run the setup program:

```
D:\PC\WIN95\SETUP.EXE
```

To install GFF under Windows 3.1, run the setup program:

```
D:\PC\WIN31\SETUP.EXE
```

Installing GFF will create an *O'Reilly Online Books* program group; select the *GFF* icon within that group to start GFF.

Ex.1007 / Page 1075 of 1158
APPLE INC.

Installing GFF on a Macintosh

If you do not have Spyglass Enhanced Mosaic version 2.11 or later installed on your system, begin by installing it. Double-click the self-extracting archive icon in the *Spyglas Enhanced Mosaic* folder in the *Mac* folder on the CD-ROM.

On the Macintosh, you can run GFF directly from the CD-ROM. Simply double-click the *GFF* icon in the *Mac* folder on the CD-ROM.

If you want to install GFF on your hard drive, simply drag the *GFF* and *GFF Init* icons into a folder on your hard disk. *GFF* and *GFF Init* must remain in the same folder.

Installing GFF on UNIX

In the descriptions that follow, the GFF CD-ROM is assumed to be mounted on */cdrom*. If you have mounted it somewhere else, subsitute its real mount point everywhere you see */cdrom*.

Many different UNIX implementations are stored on the CD-ROM; find yours in a subdirectory under */cdrom/unix*. For example, if you are on a SunOS 4.*x* machine, the subdirectory is */cdrom/unix/sunos*. For a complete list of the UNIX platforms supported, read */cdrom/unix/README*.

If you do not have Spyglass Enhanced Mosaic version 2.11 or later installed on your system, begin by installing it. Unpack the distribution in the *mosaic* directory in the directory for your platform and follow the instructions from Spyglass.

To complete the installation of GFF, copy the *gff* and *gffx* programs from the CD-ROM into */usr/local/bin** on your system. Copy *gff.ini* into */usr/local/lib*.

Look at *gff* and *gff.ini*. Make sure that the default values and configuration options in these files match the installation directories and CD-ROM mount point that you actually selected.

Removing GFF

If you should decide to remove GFF from your system temporarily, following the instructions below will ensure that GFF is completely and safely removed.

---

* If you prefer, you may install the files in another directory; */usr/local/bin* is merely a suggestion. The same is true of */usr/local/lib* in the following sentence.

Ex.1007 / Page 1076 of 1158
APPLE INC.

Removing GFF under Windows

Use the uninstall program that is created as part of the Windows installation process to remove GFF from your system.

Removing GFF under UNIX

To remove GFF under UNIX, delete the following files:

- *gff* and *gffx*, probably from */usr/local/bin*.
- *gff.ini*, probably from */usr/local/lib*.
- Individual users may have created personal customization files, *.gff.ini* or *gff.ini* in their home directories. Remove them at your discretion.

Removing GFF on the Macintosh

Simply delete *GFF* and *GFF Init* from the folder on your hard disk where you installed them.

Which Browsers Can I Use?

The state-of-the-art in browser technology changes very fast, almost daily. In order to be used with GFF, a browser must meet two requirements: it must support tables and it must support the Software Development Interface (SDI) developed by Spyglass. GFF communicates with the browser using the Spyglass SDI to provide support for searching and other dynamic behavior.

In GFF version 1.0, the SDI is the only communication mechanism supported. In future versions of GFF, other mechanisms may be supported. At the time of this writing (March, 1996), the following browsers are known to work:

Windows
: Spyglass Enhanced Mosaic version 2.11 and Netscape Navigator 2.0 (the real release, not the beta releases). Microsoft's Internet Explorer 2.0 seems to work, although it works a little strangely.

Macintosh
: Spyglass Enhanced Mosaic version 2.11. Check the GFF Web Center for other supported browsers.

UNIX
: Spyglass Enhanced Mosaic version 2.11.

Ex.1007 / Page 1077 of 1158
APPLE INC.

What if the Browser Crashes?

Some browsers, at least under Windows, crash when GFF exists. In versions of Spyglass Mosaic prior to 2.11, this was caused by a bug in the browser. Other browsers, built on the Spyglass code base, may exhibit similar problems.

If your browser crashes, stick with Spyglass Enhanced Mosaic, included on the CD-ROM.

Accessing Software on the CD-ROM

The contributed freeware and shareware on the CD-ROM are stored in directories below */software.*

You can view these directories by following links from the software pages in GFF, but you cannot install the software directly from your browser.

To install a software package, use whatever commands you would normally use to access the appropriate directory on the CD-ROM and run the install program or follow the instructions provided.

The source code and UNIX directories contain only source code, no compiled binaries. These packages are provided by their authors in this form; instructions for building these programs is beyond the scope of this book. For help building UNIX binaries, you may want to consider contacting Ready-to-Run software (see the accompanying sidebar) the company that helped us build the CD-ROM for this book.

How Does GFF Work?

GFF runs in the background and communicates with the browser, using an appropriate operating system mechanism. GFF has no user interface to speak of. In the normal course of events, GFF behaves like this:

During initialization, GFF attempts to contact a running browser. If it fails, GFF starts a browser for you. After contacting a browser, GFF sets up the communication mechanism and recedes into the background. From this point on, you can ignore GFF; it will run silently in the background handling requests from the browser.

When the browser exits, GFF notices this fact and silently exits as well.

You can end GFF explicitly, without closing the browser:

- On Windows, open the GFF window (it was automatically minimized after initialization) and choose *Exit* off the system menu (the pull-down menu under the icon in the upper-left corner of the window).

Ex.1007 / Page 1078 of 1158
APPLE INC.

Need Help Building Binaries?

If you need help with the process of building UNIX binaries, you might want to contact Ready-to-Run Software. RTR has precompiled versions of many of the packages described in this appendix for common UNIX platforms, including Sun 3, Sparc (Sun 4), DECSTATION, IBM RS6000 (AIX), HP 700 Series (HP/UX), SCO UNIX, and SCO Xenix.

Contact them at:

Ready-to-Run Software, Inc.
Software Porting Specialists.
4 Pleasant Street
Forge Village, MA. 01886 USA
Voice: 508-692-9922
FAX: 508-692-9990
Email: *info@rtr.com*

If you have a copy of *UNIX Power Tools* (O'Reilly & Associates, 1993) you will find on the accompanying CD-ROM prebuilt binaries for two of these packages (pbmplus and xloadimage) for a variety of UNIX systems.

- On the Macintosh, select GFF from the Finder menu in the upper-right hand corner and choose *Quit* from the *File* menu.
- On UNIX, run *gff -quit* or manually kill the background process by sending it the *TERM*. In order to send it a signal, you must know its process id. That id is stored in the file *.gff.pid* in your home directory. The following command will quit GFF on most versions of UNIX:

```
kill -TERM `cat $HOME/.gff.pid`
```

On some systems, if GFF started the browser, GFF can be configured to end the browser if you exit GFF directly.

Customizing GFF

Most aspects of GFF are configured by modifying the *gff.ini* file in the same directory as the executable.* This is a plain text file that can be edited with any text editor you wish. If you edit this file using a word processor, make sure that

* On UNIX systems, where there may be multiple users, you can create a copy of the global configuration in your home directory and modify that file. Ask your system administrator where the global configuration file is on your system.

Ex.1007 / Page 1079 of 1158
APPLE INC.

you save the file in plain text mode so that additional control codes are not inserted by your application.

WARNING

Modifying the configuration file has a direct impact on the performance of GFF. Inappropriate changes to the configuration file will completely cripple GFF. *Always make a backup copy before you make any changes.*

Please visit the GFF Web Center for late-breaking news and the most up-to-date customization information.

The install process creates a default *gff.ini*, and you should not need to make any changes in order to get GFF to run.

Format of the gff.ini File

The following rules define the format of the *gff.ini* file:

- *gff.ini* is a plain text file divided into named sections. Each section begins with its name in square brackets. The order of sections within the file is irrelevant.
- Within each section, variable-value pairs are stored by assignment in the form `variable=value`.
- Section and variable names are case-insensitive.
- The names of variables must be unique within each section. The order of variables within a section is mostly irrelevant. However, in the [format list] and [software list] sections, the order is relevant. The variables in each of these sections are used to construct lists; items are placed in the lists in the order that they appear in these sections.
- The names of sections must be unique within the file.
- Lines beginning with a semicolon are comments; they are ignored.

Note: changes to the initialization file while GFF is running have no effect. In the 1.0 release of GFF, GFF never makes any changes to the *gff.ini* file.

Sections in the gff.ini File

Unless otherwise noted, a section is used by all implementations of GFF (Windows, Macintosh, and UNIX). The following sections occur within the file:

Ex.1007 / Page 1080 of 1158
APPLE INC.

[config]
This section defines general configuration options for GFF.

CDROMRoot
: Defines the root directory of the CD-ROM drive. On a PC running Windows, this should be the drive letter of the CD-ROM drive that contains the GFF CD-ROM followed by a slash (for example, *d:/*). On the Macintosh, this should be the volume name of the GFF CD-ROM followed by a slash (for example, *OReilly_GFF_1_0/*). On UNIX, this should be the name of the CD-ROM mount point (for example, */cdrom*).

GFFDir
: Defines the root directory of the *Encyclopedia of Graphics File Formats*. It is set to */gff/graphics* during installation. If you prefer the text-only version of GFF, set it to */gff/textonly* (also change the "substitution" sections of the *gff.ini* file, described below, if you do this).

HomePage
: Defines the initial file displayed by GFF, relative to the root directory of the Encyclopedia. When GFF is installed, this is set to *../index.htm*, which displays the GFF "splash screen." You can bypass this screen by changing HomePage to *main.htm*.

LogFile
: Under UNIX, setting this value to the name of a file will cause GFF to write a log of transactions and some additional status information to that file. This variable is not supported on any other platforms in GFF 1.0.

CloseBrowser
: Normally, GFF does not close the browser if you exit GFF directly (as opposed to closing the browser yourself, which automatically ends GFF). If this variable is set to 1, GFF will close the browser if it started it.

[DDE]
This section identifies browsers that GFF can communicate with. It is only used by the Windows implementations of GFF.

In this section, each variable should be the DDE (Dynamic Data Exchange) name registered by an SDI-compliant browser. The value of that variable should be the full path and filename of that browser.

When GFF starts, it searches for a running instance of each of these browsers, in order. If it fails to find one, it attempts to start each of them, in order.

Ex.1007 / Page 1081 of 1158
APPLE INC.

[UnixSockets]
This section identifies browsers that GFF can communicate with. It is only used by the UNIX implementations of GFF.

At present, this section is unused. GFF determines which browser it can communicate with by examining environment variables at run time. Consult the *INSTALL* file in the directory containing your UNIX implementation of GFF for more information.

[AppleEvents]
This section identifies browsers that GFF can communicate with. It is only used by Macintosh implementation of GFF.

In this section, each variable should be the name of an SDI-compliant browser, for example "Mosaic" or "Netscape". The name is not really relevant, it is simply a mnemonic for the user. The value of that variable should be the creator code of the browser.

When GFF starts, it searches for a running instance of each of these browsers, in order. If it fails to find one, it attempts to start each of them, in order.

[imagemap mapname]
This section of the file is used internally by GFF. Do not change any values in this section.

[pages]
This section of the file is used internally by GFF. Do not change any values in this section.

[substitute]
This section of the file is used internally by GFF. Do not change any values in this section. If you plan to use the text-only version of GFF, rename this section to [graphics substitute].

[textonly substitute]
This section of the file is used internally by GFF. Do not not change any values in this section. If you plan to use the text-only version of GFF, rename this section to [substitute].

[format list]
Each of the variables in this section is a short name, a key, for a file format. The value of each key is the full name of that format. When additional formats

Ex.1007 / Page 1082 of 1158
APPLE INC.

become available on the GFF Web Center, you will be able to incorporate them into your installed version of GFF by adding a new section to the *gff.ini* file and updating this list. For now, don't make any changes to this section.

[formatkey]
There is one section for each key variable defined in the [format list], described above. Within this section, the following variables are defined:

Names
: A comma-separated list of alternate names for this format. All of the names of the format should be listed here. This is the list that GFF searches when you search for a particular format by name.

File
: If you set the file to the complete path name of a file, GFF will display that file when you view this format, instead of trying to display the version on the CD-ROM.

Several other variables are defined for each format in the *gff.ini* file that is created during installation. In GFF 1.0, these variables are not used.

[software]
The variables in this section determine how software on the CD-ROM is displayed when you view a particular format. By default, all packages are displayed.

Message
: Defines the text of the message displayed before the list of software.

Platforms
: Defines the platforms that should be displayed. This is a simple space-delimited list of values. In GFF 1.0, only the following values are defined: mac, mswin, msdos, os2, and unix.

  If a platform occurs in this list, software for that platform will be displayed when you view a format. Regardless of the value that you use here, you can always look at all of the software packages from the Software page.

platform-value
: Defines the full name of each platform.

Ex.1007 / Page 1083 of 1158
APPLE INC.

[software list]
Each of the variables in this section is a short name, a key, for a software package. The value of each key is the full name of that package. Additional packages can be incorporated into your installed version of GFF by adding a new section to the *gff.ini* file and updating this list. For more complete instructions about how to do this, please check the GFF Web Center.

[softwarekey]
There is one section for each key variable defined in the [software list], described above. Within this section, the following variables are defined:

Name
: Defines the full name of the software package.

Platform
: Defines the platform that the software package runs on. In GFF 1.0, only the following values are defined: mac, mswin, msdos, os2, and unix. Also, in GFF 1.0, only a *single* platform may be defined.

Formats
: A simple space-delimited list of the formats which this software package understands. These values must be key variables in the [format list] section of *gff.ini* in order to be of any use.

Additional Customization for the UNIX Version

While the Macintosh and Windows versions of GFF use information in the *gff.ini* file to find the browser, the UNIX version uses environment variables. This is the more common means of customization under UNIX. Four variables may be set:

`GFF_HOST`
: Defines the name of the host where the browser is running. The default is localhost. This variable is of dubious utility since GFF and the browser must each have access to the CD-ROM. In most cases, GFF and the browser run on the same host, so the default value, localhost, is correct.

`GFF_BROWSER`
: Defines the command line which must be executed in order to start your browser. This command line should include the full pathname of the browser executable and any arguments required to start the SDI. The default value is */path/*emosaic -sdi.

Ex.1007 / Page 1084 of 1158
APPLE INC.

`GFF_SDIFILE`

In order to communicate with the browser, GFF must know what port the browser is "listening" to for SDI requests. Spyglass Mosaic, the only browser known to support the SDI under UNIX at this time, stores the port number in a file in the users home directory. The default value for the name of this file is stored in GFF. GFF_SDIFILE is provided so that you can specify an alternate name, in case some other browser supports the SDI specification under UNIX in the future.

`GFF_DEBUG`

If this value is set to 1, additional messages are displayed while GFF initializes.

Ex.1007 / Page 1085 of 1158
APPLE INC.

GLOSSARY

4-bit color
Refers to a way of representing bitmap or other data that can handle up to 16 (2^4) colors.

8-bit color
Refers to a way of representing bitmap or other data that can handle up to 256 (2^8) colors.

8-mm
Tape backup format and medium. Video recording format and medium, usually used for home use due to the compact size of the physical medium, and not considered of broadcast quality. See *VHS* and *Beta.*

15-bit color
Refers to a way of representing bitmap or other data that can handle up to 32,768 (2^{15}) colors.

16-bit color
Refers to a way of representing bitmap or other data that can handle up to 65,536 (2^{16}) colors.

24-bit color
Refers to a way of representing bitmap or other data that can handle up to 16,777,216 (2^{24}) colors.

Ex.1007 / Page 1086 of 1158
APPLE INC.

accelerator card

A video display card with active components designed to enhance or speed up the display of image data sent to it by a rendering application. Contrast with dumb frame buffer.

active information device

An electronic device with which the user must constantly interact in order to obtain information. Video arcade games and most multimedia applications are active information devices. Contrast with *passive information device.*

adaptive encoding

An algorithm that has no certain prior knowledge about the format of the data it is encoding. It must adapt to the format of the data as it encodes it. LZW is an adaptive encoding algorithm.

additive system

A color system in which colors are created by adding colors to black. The more color that is added, the more the resulting color tends toward white.

aliasing

Artifact produced on a pixel-based output device where diagonal or curved edges appear jagged.

alpha channel

An additional channel of bitmap data used to store transparency data for an image, which can be on a per-pixel, per-block, or per-image basis. The degree of pixel transparency for an 8-bit alpha value ranges from 0 (the pixel is completely invisible or transparent) to 255 (the pixel is completely visible or opaque). See also *overlay bit.*

animation

A sequence of two or more images displayed in a rapid sequence so as to provide the illusion of continuous motion. Animations are typically played back at a rate of 12 to 15 frames per second.

anti-aliasing

The process of reducing artifacts by interpolating intermediate colors perpendicular to an aliased edge.

Ex.1007 / Page 1087 of 1158
APPLE INC.

aperture mask
A shadow mask with vertical strips instead of round holes. See *Trinitron, shadow mask,* and *dot pitch.*

array of pixels
An ordered set of colored display elements on an output device. This term is used loosely to refer to an array of numerical values used by an application program to specify colored elements on an output device.

ART
A method of compression developed by the Johnson-Grace company.

artifact
A detectable change in an image produced by a rendering application, such as a filter, or an editing tool, such as a paint program. Such changes are said to be introduced by human intervention and are therefore artifactual influences upon natural, ecofactual data.

aspect ratio
The proportional measurement of an image or pixel based on its horizontal and vertical size. For example, an image with an aspect ratio of 4:3 has a horizontal width that is 4/3 of the vertical height.

band
See *strip.*

banding
Horizontal, vertical, or, more rarely, diagonal bands of discoloration inadvertently placed in an image during creation or rendering.

BBS
Bulletin Board System. A telecommunications program running on a computer that allows other computers with modems to dial in and access files. BBSs are a prime source of image files and file format information. Older names for BBSs include Computer Bulletin Board System (CBBS) and Electronic Bulletin Board System (EBBS).

Beta
Video recording format and medium, considered to be of superior quality to VHS but not widely used. Not considered broadcast quality. See *VHS.*

Ex.1007 / Page 1088 of 1158
APPLE INC.

Betacam
Broadcast quality video recording format and medium. See *Beta, VHS, U-Matic, M-II,* and *D-2.*

Bezier curve
A smooth curve specified by a small set of values, including tangent and control point information.

bi-level
An image that contains only two colors: a background color and a foreground color. See *monochrome* and *halftone.*

big-endian
Refers to systems or machines that store the most significant byte (MSB) at the lowest address in a word, usually referred to as byte 0. Contrast with *little-endian.*

bit depth
The size of a value used to represent a pixel in bitmap graphics data. This is usually stated as the number of bits making up the individual data value, or sometimes the number of bytes. The number 2 raised to the power *bit depth* specifies the maximum number of values the pixel can assume. Same as *pixel depth.*

bit order
The order of the bits within a byte. The first bit in a byte may be either the most significant or the least significant bit. See also *LSB* and *MSB.*

bit plane
A 2D array of bits one bit deep. A bitmap containing pixels with a depth of eight bits may be said to contain eight bit planes. A monochrome image (one bit per pixel) is usually stored as a single bit plane.

bit sex
The state of a bit (0 or 1).

bitmap
A set of numerical values specifying the colors of pixels on an output device. In older usage, the term referred to data intended for display on an output device capable of displaying only two levels. It is used in this book as a synonym for *raster.*

Ex.1007 / Page 1089 of 1158
APPLE INC.

bitmap data
The portion of a bitmap file containing information associated with the actual image.

bitmap image
A representation of a graphics work on a raster device or in a bitmap file. Redundant in our terminology.

bitonal
See *bi-level.*

bits per pixel
See *bit depth.*

block
See *chunk.*

bpp
Same as bits per pixel.

broadcast quality
Video recording medium retaining sufficient quality after multiple edit and copy operations to be broadcast on commercial television.

byte order
The order of bytes within a word of data. The first byte in a word may be either the most significant or least significant byte. See also *big-endian, little-endian, LSB,* and *MSB.*

bytes per pixel
Bit depth expressed in bytes.

CAD
See *Computer Aided Design.*

CCD
Charge coupled device. An array of electronic components which convert light into electrical signals. Used in scanners and video cameras. See *scanner.*

Ex.1007 / Page 1090 of 1158
APPLE INC.

CCITT
International Telegraph and Telephone Consultative Committee. See *ITU.*

CD
See *Compact Disc.*

CD-DA
Compact Disc–Digital Audio. The standard used for encoding audio data onto a compact disc.

CD-I
Compact Disc–Interactive. The standard used for encoding audio and video information onto compact discs for use in interactive multimedia systems.

CD-R
Compact Disc–Recordable. The standard for creating write-once compact discs that may be mastered on a standard PC.

CD-ROM
Compact Disc–Read-Only Memory. A compact disc containing data encoded using the CD-XA standard. See also *CD-XA* and *ISO-9660.*

CD-XA
Compact Disc–Extended Architecture. The standard used for encoding data onto what we know as a CD-ROM. See also *ISO-9660.*

chroma
Term used when referring to color. Same as *chrominance.*

chromakeying
The process of creating an image, a portion of which is placed on a background of uniform color, usually blue, so that another image can later be added by placing it in the area of uniform color.

chrominance
The color portion of an image. It is the mixture of hue and saturation, or the combination of three primary colors, such as red, green, and blue.

Ex.1007 / Page 1091 of 1158
APPLE INC.

chunk
A collection of data with a known format within a graphics file. Chunks are also called blocks in some graphics file format specifications. See also *packet.*

chunking
The breaking up of a block of data into two or more smaller pieces, usually to accommodate memory limitations or to avoid hardware dependencies.

CIE
International Commission on Illumination (Commission Internationale de l'E'clairage). The CIE established an international standard for primary colors in 1931. This standard allows all colors to be defined as a weighted sum of three primary colors.

CLUT
Color Look-Up Table. See *look-up table.*

CMY
Acronym for Cyan/Magenta/Yellow. A subtractive color system based on the primary colors cyan, magenta, and yellow.

CMYK
Acronym for Cyan/Magenta/Yellow/Key. A subtractive color system based on the primary colors cyan, magenta, and yellow. Key color is the color black, which is not reproducible using the CMY model alone.

codec
Shortened version of encoder/decoder (similar to modem for modulator/demodulator). A codec is any hardware device or set of software algorithms that can encode data and decode it back to it original (lossless) or reasonably original (lossy) state.

color calibration
The process of determining and adjusting the properties of a display device or the colors in an image to ensure that the rendered image is accurate to some standard, usually the human eye.

Ex.1007 / Page 1092 of 1158
APPLE INC.

color channel
One of the numerical elements used to specify a color in a particular color model when that color is specified using an ordered n-tuple. Green is one channel in the RGB color model, which is specified using the ordered triplet (R,G,B).

color correction table
A section of a file for the storage of information designed to help a rendering application in displaying an image on a particular output device or class of output devices different from that assumed by the creator application.

color definition scheme
A system by which colors are specified, usually by numerical values or ordered sets of numbers.

color gamut
The range of colors which can be displayed using a particular color model or output device.

color map
See *look-up table.*

color-mapped image
Image data whose colors are not stored in the bitmap itself, but in a separate data array.

color models
The way colors are broken down and specified in a particular application or system.

color plane
A section of a file holding information about one color component of the color model currently in use.

color space
When a particular color scheme uses an ordered n-tuple to specify color, all the possible values corresponding to colors can be plotted on an n-dimensional graph. All the points plotted, which correspond to colors in the color model, constitute the color space.

Ex.1007 / Page 1093 of 1158
APPLE INC.

color table
See *look-up table.*

color values
Same as *pixel values.*

Compact Disc
A circular plastic disc used for the storage of audio, video, textual, and other data that can be represented in a digital form, and from which data can be retrieved using an optical process. Although there are various formats, the one in most common use is 4.75 inches (12 centimeters) in diameter. See also *CD-ROM.*

component video
Color video information transmitted using three separate signal channels. RGB, YIQ, and YUV are examples of component video signals.

composite color
A color specified in a color model where that color is specified using an ordered n-tuple. A system where more than one color channel value exists, and where more than one channel value is needed to specify the color.

composite image
An image formed of two or more subimages stored separately in a file. Sometimes refers to a bitmap image with a lot of color variation per unit area.

composite video
Color video information transmitted using a single signal channel. NTSC, PAL, and SECAM are examples of composite video signals.

compression artifact
Spurious addition to or degradation of an image due to the process used to compress or decompress it.

compression type
The algorithm or family of algorithms used by the creator application in producing the file.

Ex.1007 / Page 1094 of 1158
APPLE INC.

Computer Aided Design
The use of applications, usually vector-based, for the design and rendering of graphical data of architectural and mechanical drawings, electronic schematics, and 3D models. Commonly referred to as *CAD.*

contiguous data
Image data stored as a continuous block of pixel values without scan-line or block delimiters.

continuous tone
An image consisting of smooth gradations of color between adjacent elements, requiring an output device capable of displaying thousands or millions of colors at high resolution in order to prevent image artifacts.

convenience revision
A file format version created by an application vendor to accommodate a bug or quirk in a program. This is sometimes caused by ignorance or honest error, but in many cases is intentional. There is ample evidence that at least one vendor, the custodian (but not the originator) of a file format specification, knowingly released format revisions so as to avoid shipping delays caused by bugs introduced by junior programmers.

convolution
The process of transforming the value of a pixel, or a field of pixels, based on a mathematical formula. Convolution is used to alter the color of an image (filtering), or to re-encode the data (compression).

copyright
The exclusive rights to the production, publication, and sale of a work of authorship, such as a photograph or a captured image.

CRC
Cyclic Redundancy Check. An algorithm that recursively generates a 16- or 32-bit numerical value based on a stream of data. The value can be used to verify whether the data has changed. See *digital signature.*

D-2
Broadcast quality video recording format and medium. See *Beta, VHS, U-Matic, M-II,* and *Betacam.*

Ex.1007 / Page 1095 of 1158
APPLE INC.

data compression
The process of converting data from one format to another format that is physically smaller in size. The same logical information is stored using less physical information.

data element
Typically the smallest units of readable data with a collection of data. Bits, bytes, WORDs, and DWORDs are all data elements.

data encoding
A generic term for the process of converting data from one format to another. Data compression and data encryption are both forms of data encoding.

data encryption
The process of converting data from an intelligible format to an unintelligible, but decryptable, format.

DCT
Discrete Cosine Transform. A mathematical transform used to convert data from a 3D to a 2D form. Used by lossy compression methods such as JPEG and MPEG.

DDB
Device Dependent Bitmap. A bitmap format designed to support the capabilities of a specific type of display hardware. A bitmap format not designed with portability in mind.

decimation
The process of throwing away portions of a bitmap image when reducing it in size.

decoder
An algorithm that converts encoded data to a raw format.

DIB
Device Independent Bitmap. A bitmap format that is designed not to be limited by the capabilities of a specific type of display hardware. A portable or interchange data format.

Ex.1007 / Page 1096 of 1158
APPLE INC.

dichroic filter
A light producing mechanism which concentrates and directs visible light but not infrared radiation (heat). Used in movie and multimedia production, and in some scanners. See *scanner.*

digital camera
An input device in the form of a camera, capable of delivering bitmap image data of real-world scenes in digital form to a creator application.

digital signature
An electronic method of verifying the authenticity of a message or file. A block of data is attached to the message that can be used to validate who sent the message and when it was sent. If a change is made to the file, the signature will not verify.

digitizing
The process of converting an analog signal to a digital signal. See *sampling.*

digitizing device
A device that creates a version of a physical graphical representation by creating a digital version. Common digitizing devices are scanners, image capture boards that work with video cameras, and digital cameras.

digitizing tablet
An input device incorporating a pen-like component and a flat surface, and meant to provide a simulation of the interaction between the hand, a pen or pencil, and paper. Can be pressure-sensitive.

display adapter
See *video display card.*

display surface
The portion of an output device where an image appears. The screen of a monitor, or printed paper.

dithering
The process of displaying colors not available on an output device. Patterns of other colors are created by intermixing monochrome pixel values with color pixels to produce shading and highlighting that appears to the eye as differing colors. See *monochrome* and contrast with *halftone.*

Ex.1007 / Page 1097 of 1158
APPLE INC.

dot pitch
The measure of the spacing between the centers of physical pixels on an output device, usually a monitor. On monitors, dot pitch is expressed in millimeters (mm) between the centers of like-colored pixels. See *shadow mask.*

down-bit ordering
Where the most significant bit (MSB) is the first bit read in a byte. See *MSB.*

drawing surface
The area on an output device where a rendered image appears.

drum scanner
A high-resolution scanning device used in the most demanding professional applications. They can often be the source of extremely large files. See *scanner* and *film scanner.*

dumb frame buffer
A video display card consisting mainly of a frame buffer with few enhancements. Contrast with *accelerator card.*

EDIP
See *Electronic Document Image Processing.*

Electronic Document Image Processing
A subfield of image processing specializing in the creation, storage, and manipulation of black-and-white images derived from printed documents. At least 75 percent of the image-processing market today is based on EDIP systems and applications.

element attributes
Information, such as color, line width, pen style, and fill color, stored in a file for use by a rendering application in reconstructing an image element.

encoder
An algorithm that converts raw data to an encoded form, usually to physically compress the data.

Ex.1007 / Page 1098 of 1158
APPLE INC.

entropy encoder
See *lossless encoding.*

FAX files
Graphics files produced by a program that manages FAX-modem hardware. These are generally bitmap files and may be compressed. They are often in a proprietary format, although versions of TIFF and PCX are popular.

field
A fixed-size data structure in a file.

fields
Images stored in a video or animation file designed to support various display technologies. NTSC video consists of two sets of images meant to be displayed alternately, each of which is a field.

file element
The smallest unit of logical information within a file. Examples include fields within graphics file headers and color triples used to store RGB pixel data.

file header
A data structure containing information on the data stored within a file. Graphics file headers contain information such as the height and width of an image and the number of bits per pixel.

file ID
See *file identifier value.*

file identifier value
A specific value, or set of values, used to positively identify a file as being of a particular file format. File ID values may be an integer, such as 59A66A95h, or a string of ASCII characters, such as BITMAP, and they usually appear in the first field of a file header. Also called *magic number.*

fill attributes
Fill color and/or other information associated with an image element, and used by the rendering application to reconstruct an image.

Ex.1007 / Page 1099 of 1158
APPLE INC.

fill color
A color meant to be used by a rendering application when filling a closed area, usually polygonal, when reconstructing an image element.

filler
See *padding.*

film scanner
See *slide scanner.*

filter
A section of code or program which operates on an entire block of data. Contrast with scanner.

fixed
Refers to an element in a file that has a known position, usually identified by an offset from a landmark in a file.

fixed frequency
A monitor or other output device with fixed frame rate and horizontal frequency, only capable of displaying a small set of resolutions. See horizontal frequency and frame rate.

footer
A data structure similar to a header but appended to the end of a file.

format creator
The person or organization responsible for the definition of the physical structure of, and conventions associated with, a file format. Often this person is a programmer called on to produce a file format in association with an application. In some cases, the format creator is a standards committee.

fractal
Repetitious patterns that naturally occur in the texture of all surfaces. Mathematics is used to described the properties of fractals.

fractal compression
The use of fractal encoding to reduce the amount of physical data required to store an image. See *fractal encoding.*

Ex.1007 / Page 1100 of 1158
APPLE INC.

fractal encoding
The process of describing a bitmapped image as a sequence of fractals and by its fractal properties. See *fractal.*

frame
A single image. Multiple frames of slightly differing images displayed in rapid sequence are used to create animations.

frame buffer
Older term for video display card. Technically, the portion of a video display adapter containing memory in which digital image data is assembled prior to sending it to a monitor. See *video display card.*

frame rate
Number of full images which can be displayed by an output device, usually a monitor. Number of fields a monitor can display in a given time. Usually expressed in hertz (Hz). See *fields, interlaced, non-interlaced,* and *vertical retrace.*

frames
Series of single images stored in a video, animation, or multimedia format file, meant to give the illusion of motion when rendered in rapid succession.

frequency
Factor that determines the quality of the image a monitor or other output device can display. Higher frequency monitors update the screen faster and theoretically can display more information in a given amount of time. See *vertical retrace, interlaced,* and *non-interlaced.*

FTP
File Transfer Protocol. A low-level protocol used to transfer files between computers over computer networks. FTP is the primary means by which binary files are transferred between machines on the Internet.

fullcolor display
A term sometimes used to imply that a device is capable of displaying 2^{15} (32,768) or 2^{16} (65,536) colors; however, this actually describes *hicolor; fullcolor* tends to be a marketing term, rather than a technical one.

Ex.1007 / Page 1101 of 1158
APPLE INC.

full-motion video
Video image frames displayed at a rate of 30 frames per second for NTSC and 25 frames per second for PAL.

fuzzy logic
A sub-discipline of mathematics used to quantify subjective linguistic concepts, such as bright, dark, very far, quite close, most usually, almost impossible, etc.

G3
Abbreviation for CCITT Group 3 encoding.

G4
Abbreviation for CCITT Group 4 encoding.

gamma
A numerical value used to indicate the non-linear response curve of an output device to light intensity. Used to correct the intensity of an image on a display device (gamma correction). Also called gamma value.

gradient fill
An elaboration on a fill color consisting of two colors placed at opposite ends of a closed area of an image element. The area is filled with a continuous blend of color intermediate to the two colors and between the two ends. See *fill color.*

graphic work
The end result of effort by a graphic artist. A drawing or other artifact.

graphics adapter
See *video display card.*

graphics card
See *video display card.*

graphics data
Data which may or may not have a physical representation, intended for display on an output device.

Ex.1007 / Page 1102 of 1158
APPLE INC.

graphics file
A file containing graphics data.

graphics file format
The definition of, and conventions associated with, a file structure used for the storage of graphics data.

graymap
In older terminology, raster data composed of values with more than two levels, intended for an output device capable of displaying only shades of gray.

gray-scale
A term used when referring to an image. A gray shade is any color whose three primary colors are the same value. Gray shades only have intensity (luminance) and no color (chrominance).

halftone
The use of bi-level pixels or dots to create the appearance of shades of gray by grouping the pixels in patterns that produce the desired shades. Used in printing and liquid crystal displays. See *bi-level* and contrast with *dithering*.

HBL
Acronym for Hue/Brightness/Luminosity. See *HSI*.

heuristics
A set of rules derived from experimentation or experience.

Hi8
Video recording format offering resolution of over 400 lines. See *VHS*, *S-VHS*, *8-mm*, and *Beta*.

hicolor display
A term used to imply that a device is capable of displaying 2^{15} (32,768) or 2^{16} (65,536) colors.

HLS
Acronym for Hue/Lightness/Saturation. See *HSI*.

Ex.1007 / Page 1103 of 1158
APPLE INC.

horizontal frequency
Measure of the speed at which the electron beam in a monitor sweeps across the width of the active area of the screen. Usually expressed in kilohertz (kHz).

HSB
Acronym for Hue/Saturation/Brightness. See *HSI.*

HSI
Acronym for Hue/Saturation/Intensity. An additive color system based on the attributes of color (hue), percentage of white (saturation), and brightness (intensity). Similar or identical color systems include HBL, HLS, HSB, HSL, and HSV.

HSL
Acronym for Hue/Saturation/Luminosity. See *HSI.*

HSV
Acronym for Hue/Saturation/Value. See *HSI.*

hue
Any color, such as red, violet, orange, and so on.

hybrid text
The storage and display of bitmap and textual data using a single graphics file format. GIF89A is an example of a format with a hybrid text capability.

hybrid database
The ability to store complex and highly organized database information in conjunction with graphical data. See also *hybrid text.*

hypertext
A collection of graphical and textual data organized in such a way as to facilitate easy access to all of the information it contains. Hypertext may be thought of as a precursor to multimedia, or simply as an extension of it. Certain extensions of hypertext are becoming known as *hypermedia.*

icon
A small bitmap image used as a placeholder to represent an object. An icon may also be though of as a graphical file name. Contrast with *thumbnail image.*

Ex.1007 / Page 1104 of 1158
APPLE INC.

ID value
See *file identifier value.*

IEC
International Electrotechnical Commission. See *ISO.*

image
A visual representation of graphics data displayed on the display surface of an output device. Output of a rendering application. One end of the graphics production pipeline. A single frame from an animation or video sequence.

image bitmap
See *bitmap image.*

image data
A term used loosely to refer to bitmap data, or the portion of a bitmap file containing bitmap data.

image elements
Portions of an image, often repeated, from which the image is composed by duplication, rotation, scaling, and translation.

image file index
Image offset table. An area of a file designed to hold descriptions of subimages, image components, or individual images in a multi-image file.

image height
The vertical size of an image, usually expressed in pixels or scan-lines or other non-device-dependent units.

image length
The horizontal size of an image, usually expressed in pixels or other non-device-dependent units.

image offset table
A portion of a file designed to hold offsets, usually measured in bytes, where subimages, image components, or individual images in a multi-image file can be found.

Ex.1007 / Page 1105 of 1158
APPLE INC.

index map
See *look-up table.*

index values
Pairs of numbers arranged in a table so that an application can match numbers it knows about to numbers representing colors that an output device knows about.

indexed-color image
See *color-mapped image.*

indirect color
The specification of colors through the use of a palette or look-up table.

input
Generic term in computer technology referring to any data which is processed or transformed.

interframe encoding
The creation of encoded data from two or more image frames. MPEG encoding is an interframe encoding method.

interlaced
Refers to a strategy used by televisions and some older monitors, where the electron gun draws every other line during a single sweep across the active area of the screen. The alternate lines are filled in during a second pass. See *vertical retrace, non-interlaced,* and *fields.*

interlaced encoding
See *interleaved encoding.*

interleaved encoding
The storage of bitmap scan-lines, or pixels within scan lines, in a non-sequential pattern. Contrast with *sequential encoding.*

interleaving
In reference to a single image, the storage of two or more subimages which are combined to create a final image by displaying alternate scan-lines from each subimage. In reference to video, multimedia, or animation formats, the process of storing or displaying information other than that used to reconstruct the video portion between video frames.

Ex.1007 / Page 1106 of 1158
APPLE INC.

interpolation
The addition of pixels between pairs of others. Usually made necessary when enlarging a bitmap.

intraframe encoding
The creation of encoded data from a single image frame. JPEG encoding is an intraframe encoding method.

ISO
International Standards Orgainzation. The primary organization for creating world-wide technical standards. Along with the IEC and the ITU, the ISO authors and maintains standards for everything from nuts and bolts to computer languages.

ISO-9660
A file system standard developed for CD-ROMs using the CD-XA encoding standard. An ISO-9660 file system is readable by many operating systems, including MS-DOS, Macintosh OS, and UNIX.

ITU
International Telecommunications Union. An agency of the United Nations responsible for telecommunications. The ITU replaced the CCITT on Februrary 28, 1993.

ITU-T
ITU Telecommunications Standardization Sector. The body within the ITU responsible for setting world telecommunications standards (Recommendations).

jaggies
Term denoting the presence of aliasing in an image.

JBIG
Joint Bi-level Image Experts Group. The ISO/IEC JTC1/SC29/WG9 and CCITT SGVIII group formed in 1988 to establish a standard for the progressive encoding of bi-level image data. JBIG is also the name given to the codec created by this group.

Ex.1007 / Page 1107 of 1158
APPLE INC.

JPEG
Joint Photographic Experts Group. The group of the ISO responsible for the creation and maintenance of the JPEG still-image compression standard.

JTC1
Joint Technical Committee 1. A group within the ISO/IEC that handles information technology.

key points
Points necessary for the reconstruction of a graphics object from vector data. These are usually the minimum needed to specify the object. Two points at the corners of a rectangle are the key points.

landmark
Refers to an element in a file from which other positions and offsets are calculated. The canonical landmarks are the beginning, end, and current position. Other features, such as prominent data structures, may at times act as landmarks.

laser disc
Recording medium used for video, similar to a large CD or CD-ROM, offering the advantage of random access and high quality.

leak
Image artifact, usually generated from vector data, produced by a rendering application from an image element incorrectly designated as closed. Usually consists of horizontal lines of the fill color in an inappropriate area of the image.

little-endian
Refers to systems or machines which store the least significant byte (LSB) at the lowest address in a word, usually referred to as byte 0. Contrast with *big-endian.*

logical pixels
Idealized pixels having perfectly defined characteristics and occupying no physical extent. The graphics equivalent of a mathematical point. Contrast with *physical pixels.*

Ex.1007 / Page 1108 of 1158
APPLE INC.

look-up table
A series of pairs of numerical values whereby a program can match a meaningful value to one which specifies a color on an output device.

lossless encoding
A data compression or encoding algorithm that does not lose or discard any input data during the encoding process.

lossy encoding
A data compression or encoding algorithm that loses, or purposely throws away, input data during the encoding process to gain a better compression ratio. JPEG is an example of a lossy encoding method.

LPI
Lines per inch, usually used to refer to screen size or the resolution of an output device. See *screen.*

luminance
The brightness or intensity of a color. The pixels in a monochrome image have a luminance of either 100 percent or 0 percent.

LSB
Depending on context, either the least significant byte (of more than one juxtaposed bytes) or the least significant bit (of the bits in a byte or word of data). Contrast with *MSB.*

LUT
See *look-up table.*

M-II
Broadcast quality video recording format and medium. See *Beta, VHS, U-Matic, Betacam,* and *D-2.*

magic number
See *file identifier value.*

magic values
Arbitrary numbers or text strings, often picked "out of the air" by a format creator for the purpose of identifying the format.

Ex.1007 / Page 1109 of 1158
APPLE INC.

message digest function
A family of algorithms used to create digital signatures.

metadata
Data comprised of attributes, parameters, notebooks, and other types of miscellaneous complex data aggregates associated with primary scientific data.

metafile
A file format capable of storing two or more types of image data, usually vector and bitmap, in the same file.

MIDI
Acronym for Musical Instrument Digital Interface. A standard for digital signals used to control electronic musical instruments. MIDI information may be stored as a data file and is found in many multimedia file formats.

MMR
Modified Modified READ, The compression algorithm used in CCITT Group 4 facsimile compression.

monochrome
An image composed of a single color and black. Most monochrome images are black and white, although any color might be substituted for white. Also called 1-bit images. Although the term monochrome, of course, means single-colored, in computer graphics it is used to denote a system where two colors can be specified: the foreground color and the background.

MPEG
Motion Picture Experts Group. The group of the ISO responsible for the creation and maintenance of the MPEG video compression standard.

MR
Modified READ, The compression algorithm used in CCITT Group 3 facsimile compression.

MSB
Depending on context, either the most significant byte (of more than one juxtaposed bytes) or the most significant bit (of the bits in a byte or word of data). Contrast with *LSB*.

Ex.1007 / Page 1110 of 1158
APPLE INC.

multi-channel palette
A palette with two or more individual color values per color element. Contrast with *single-color palette.*

multi-sync
A monitor or other output device with frame rate and horizontal frequency adaptable to demand. See *horizontal frequency, fixed frequency,* and *frame rate.*

multimedia
The concept of creating, storing, and playing back two or more forms of electronic information simultaneously. Such information includes still-images, motion-video, animations, digitized sound, and control information such as MIDI codes.

non-interlaced
Refers to a strategy used by higher-frequency monitors where the electron gun draws all lines of the active area of the screen during a single sweep. See *interlaced, vertical retrace,* and *fields.*

NTSC
Acronym for National Television Standards Committee. The standards committee responsible for, among other things, the creation of the color television signal used in the United States (NTSC video).

objects
Image elements, particularly in vector files. Descriptions of complex image elements. Image element information stored along with code for use by the rendering application.

output
Generic term in computer technology meaning the result of any process or transformation of data.

output device
Physical mechanism used to create a display.

output device language
A computer language or set of commands created by a vendor to communicate with a particular output device, such as a printer. Hewlett Packard's PCL is one well-known output device language and is understood by Hewlett Packard printers and HP-compatible printers. It may or may not be easily human-readable. See *page description language.*

Ex.1007 / Page 1111 of 1158
APPLE INC.

overflow
A condition which results when data of a certain size is placed in a storage cell, such as location in memory or a register, which is too small to hold it. This is usually a problem resulting in loss or corruption of data.

overlay bit
An additional bit found in a pixel or pixel plane that indicates whether the pixel is displayed as visible (opaque) or transparent (overlayed). See *alpha channel.*

packet
A block of data with a known structure, usually used to denote elements of a stream.

padding
Portion of a file usually included to accommodate machine dependencies or to increase reading or writing speed.

page description language
A computer language created by a vendor to communicate with output devices. It may be a fully functional language and is always human-readable. It is generally more sophisticated than an output device language and is not tailored to any particular output device. The most popular page description language in use today is Adobe's PostScript.

page table
An array of offset values used to index the location of multiple bitmaps within a single graphics file. Each offset value indicates the starting position of each bitmap.

PAL
Acronym for Phase Alternation Line. PAL is a standard of color television and video signals developed in West Germany and used throughout Europe (PAL video).

palette
The gamut of colors which a device can display; a software data structure used to match numbers that are meaningful to a software program to numbers that cause colors to appear on an output device.

Ex.1007 / Page 1112 of 1158
APPLE INC.

passive information device
An electronic device with which the user need not interact in order to obtain data. Television and newspapers are examples of passive information devices. Contrast with *active information device.*

PDL
Page Description Language. A computer language used for describing the layout, font information, and graphics of printed and displayed pages.

pel
See *pixel.*

pen
A logical device used by creator applications to draw lines or curves or objects composed of them, having the properties of width, color, and possibly line style. An area in a file holding information used by a rendering application in reconstructing lines.

persistence
A term often used in object-oriented technology to describe data that is stored in a static medium, such as a disk file or database. The data is said to "persist" even after the application that created it is no longer in memory. Spreadsheet, word processing, and graphics files are examples of persistent data.

PGP
Pretty Good Privacy. A powerful public-key encryption system authored by Phil Zimmermann that is freely available on the Internet.

physical pixels
The actual pixels which appear on the display surface of a raster output device. Contrast with *logical pixels.*

picture element
See *pixel.*

pixel
In traditional usage, short for "picture elements." These are irreducible elements of color created by an output device on its display surface. The term is sometimes used loosely to refer to the values of bitmap data elements used by an application to order the display of color elements on an output device.

Ex.1007 / Page 1113 of 1158
APPLE INC.

pixel depth
See *bit depth.*

pixelmap
In older terminology, bitmap data composed of values with more than two levels, intended for an output device capable of displaying color.

pixel values
Numerical data items in a graphics file indicating the color or other information associated with an individual pixel.

pixmap
See *pixelmap.*

planar data
Image data stored as separate color planes, and meant to be assembled into the final image by the rendering application. Contrast with *scan-line data.*

planar files
Graphics files with image data stored as bit planes or color planes rather than as pixels.

plotter
A bitmap rendering of vector or 3D graphical data used to display an approximation of the graphical data. See *thumbnail.*

predictive encoding
An algorithm that has certain prior knowledge about the format of the data it is encoding. Huffman is a predictive encoding algorithm.

preview
A bitmap rendering of vector or 3D graphical data used to display an approximation of the graphical data. See *thumbnail.*

primary colors
Colors in a particular color model from which other colors can be constructed. In the RGB color model, red, green, and blue are the primary colors because other colors can be produced by mixing them.

Ex.1007 / Page 1114 of 1158
APPLE INC.

production pipeline
The series of operations involved in defining, creating, and displaying an image, from conception to its realization or recording on an output device.

progressive encoding
The storage of a single bitmap as several different images, each at a different level of resolution.

pseudo-color
A color specified through the use of a palette or look-up table.

quantization
The process of reducing the number of colors defined in the source data to match the number available on an output device.

quantization artifacts
Generally refers to features introduced in an image when the data used to render that image is converted to a data format capable of displaying fewer colors than the original. Banding and false color are two examples of possible quantization artifacts. Usually considered undesirable.

raster
Refers to graphics data represented by color values at points, which taken together describe the display on an output device. *Bitmap* is used in preference to *raster* in this book.

raw
Image data without header information. Sometimes refers to image data, especially bitmap data, which is uncompressed or otherwise unencoded.

READ
Relative Element Address Differentiation code. A compression method used by CCITT Group 3 and 4 facsimile transmission.

realization
The representation of an image on an output device. Sometimes meant to signify the current rendered version of some particular graphics data.

Ex.1007 / Page 1115 of 1158
APPLE INC.

render
To produce a visual representation of graphics data on an output device.

rendering
The actual representation of an image on an output device.

representation
The actual artifact produced as the end result of the computer graphics production process, which may be an image on a monitor or on paper.

reserved fields
Fields in a file designated in the format specification as reserved space.

reserved space
Portion of a file designated in the format specification as space for additional information should it become necessary in the future.

resolution
The measure of detail within an image. The resolution of an image is its physical size (number of pixels wide by number of scan lines long). The resolution of a display is the number of scan lines it may display (800×600 is a higher resolution than 320×200).

RGB
Acronym for Red/Green/Blue. An additive color system based on the primary colors red, green, and blue. The RGB model is loosely patterned after human eyes, which have a peak sensitivity to the colors red, green, and blue light.

RLE
Run-Length Encoding. A simple method of compressing runs of identical byte sequence values into a code only a few bytes in length.

S-VHS
Video recording format and medium offering horizontal resolution of over 400 lines. See *VHS* and *Beta.*

sample rate
The number of digital samples recorded per second. The sample rate increases with the number of samples recorded per second. Same as *sample resolution.*

Ex.1007 / Page 1116 of 1158
APPLE INC.

sample resolution
See *sample rate.*

sampling
The process of reading an analog signal at specific increments in time (sample rate) and storing the data as digital values. Sampling is the basic process used to create digital audio and video.

saturation
The percentage of white in a color. Zero percent saturation is full white (no color). 100 percent saturation is no white (pure color).

scalable
An image, such as that stored as vector data, which can be scaled without introducing artifacts.

scaling
The process of enlarging an image in one or more directions.

scan line
A row of pixels. The term comes from the scanning action of raster CRT output devices, which produces successive lines of output on the display surface.

scan-line data
Image data stored as scan lines, and meant to be displayed a line at a time by the rendering application. Contrast with *planar data.*

scan-line table
An array of offset values used to index the location of each scan line or tile within a collection of bitmap data, which may or may not be compressed. Each offset value indicates the starting position of each scan line or tile.

scanner
An input device generating a bitmap image of a surface. A section of code or program allowing random access to a block of data and which treats different portions of the block differently in accord with its informational content. Contrast with *filter.*

Ex.1007 / Page 1117 of 1158
APPLE INC.

scene description language
A computer language used to describe the position and attributes of objects within a 2D or 3D image. A file produced by such a language is called a scene format or scene description file.

screen
Term borrowed from traditional graphics denoting a device or process designed to turn a continuous tone image into an array of dots, usually for display on a low-resolution output device. See *LPI.*

SECAM
Acronym for Sequential Coleur Avec Memoire (sequential color with memory). SECAM is a standard of color television and video signals used in France and several other European countries (SECAM video).

segment
An independent section of a data stream. For example, a JPEG data stream is composed of many different types of information, each stored in a separate segment.

sequential encoding
The storage of a bitmap using the natural order of its scan lines from the top-to-bottom or bottom-to-top of image. Contrast with *interleaved encoding.*

shadow mask
A plate with tiny holes, usually in a monitor, which helps locate and focus the electron beam. The spacing of the holes in monitors with a shadow mask determines the dot pitch. See dot pitch.

single-channel palette
A palette with one color value per element. Contrast with *multi-channel palette.*

single pass scanner
A scanner which images an object in a single pass of its mechanism. See *scanner, three-light method,* and *three-filter method.*

slide scanner
A device allowing the digitization of slides taken with conventional film cameras. Usually much higher resolution devices than scanners designed with paper in mind. See *scanner.*

Ex.1007 / Page 1118 of 1158
APPLE INC.

stream
Data with no fixed position in a file, composed of sub-elements with a known structure.

strip
A collection of one or more contiguous scan lines in a bitmap. Scan lines are often grouped in strips to buffer them in memory more efficiently. Also called *bands* in some file format specifications.

stripe mask
See *aperture mask.*

stroke font
A file of character information meant to be rendered by drawing single lines, usually by a plotter or other device responding only to pen up, pen down, and move to commands.

subsampling
See *decimation.*

subtractive system
A color system in which colors are created by subtracting colors from white. The more color that is added, the more the resulting color tends toward black.

tag
A data structure in a file which can vary in both size and position.

three-filter method
Process whereby color images are produced, usually in scanners, by illuminating the object to be imaged with white light which is made to pass through three successive colored filters. Contrast with *three-light method.* See *color model* and *scanner.*

three-light method
Process whereby color images are produced, usually in scanners, by successively illuminating the object to be imaged with three colors, usually red, green, and blue. Contrast with *three-filter method.* See *color model* and *scanner.*

Ex.1007 / Page 1119 of 1158
APPLE INC.

thumbnail
A small image derived from a larger image used to quickly display an approximation of the contents of an image. See *preview.*

tile
A 2D sub-section of a bitmap. For example, a bitmap 100×100 pixels in size may be divided into four 25×25 pixel tiles. Pixels are often grouped as tiles rather than scan lines to achieve a more efficient use of memory.

trailer
See *footer.*

transcode
Convert from one encoded data format to another encoded data format—for example, converting CCITT Group 3-encoded data to RLE-encoded data.

transform
See *convolution.*

transparency
The degree of visibility of a pixel against a fixed background. A totally transparent pixel is invisible. See also *alpha channel.*

trichromatic colorimetric
Color models that use three color channels to specify a color. The RGB color model is a trichromatic colorimetric system.

Trinitron
Trademark of Sony Corporation, referring to picture tubes used in monitors and televisions using aperture mask technology, and where dot pitch is an expression of the horizontal distance between strips in the aperture mask. See *aperture mask, shadow mask,* and *dot pitch.*

triple sensor scanner
A scanner which images an object using three recording devices at once, usually CCDs. See *scanner* and *CCD.*

Ex.1007 / Page 1120 of 1158
APPLE INC.

truecolor display
A term used to imply that a device is capable of displaying 2^{24} (16,777,216) colors or more (said to match or exceed the color-resolving power of the human eye). *Truecolor* formerly referred to any device capable of displaying 2^{15} (32,768) colors or more, but *hicolor* more accurately describes the display of 2^{15} (32,768) or 2^{16} (65,536) colors.

U-Matic
Broadcast quality video recording format and medium. See *Beta, VHS, Betacam, M-II,* and *D-2.*

up-bit ordering
Where the least significant bit (LSB) is the first bit read in a byte. See *LSB.*

vector
Refers to graphics data composed mainly of representations of lines and outlines of objects, which can be compactly represented by specifying sets of key points. A program displaying vector data must know how to draw lines by interpolating points between the key points.

vertical refresh rate
See *frame rate.*

vertical retrace
The interval between when an electron beam in a monitor reaches the end of its sweep across the active area of the screen and when it returns to the start.

vertical scanning frequency
See *frame rate.*

VHS
Video recording format and medium in wide use in conjunction with television technology, offering horizontal resolution of 240 lines. Not considered broadcast quality. See *Beta.*

video adapter
See *video display card.* Also, a device allowing output in NTSC, PAL, or other video format.

Ex.1007 / Page 1121 of 1158
APPLE INC.

video adapter card
See *video display card.*

video controller
See *video display card.* Also, a video adapter which may have circuitry for controlling a video recording or playback deck.

video display card
Device which takes digital information from a rendering application and converts it to an analog format suitable for output on a monitor.

virtual output
Data or an image that is produced, but that can't be seen—in other words, for which no physical representation yet exists. Data in a file.

virus
A computer program that mimics a biological virus in its characteristics and actions, including hiding, replicating, and possibly causing the death of its host.

voxel
A 3D pixel. Voxels contain all of the components of a pixel (such as color values) and include an extra component that specifies the distance of the voxel from the point of observation.

VRML
Virtual Reality Modeling Language. An interpreted language used to render both still and animated 3D objects.

VTR
Video tape recorder used as an output device for video, animation, and multimedia creator applications.

x origin of image
The point in an image from which pixels are numbered in the horizontal direction. Usually at a corner or at the center of the image.

y origin of image
The point in an image from which to start counting scan-lines. Usually at the top, bottom, or center of the rendered image.

Ex.1007 / Page 1122 of 1158
APPLE INC.

YIQ

The color model used by NTSC video signals. See NTSC.

YUV

Acronym for Y-signal, U-signal, and V-signal, which is based on early color television terminology. A luminance/chrominance-base color model (Y specifies gray-scale or luminance, U and V chrominance) used by many video compression algorithms, such as MPEG.

Ex.1007 / Page 1123 of 1158
APPLE INC.

INDEX

1-bit displays, 29
3D file formats, 15
  3D Studio, 282-309
  Sense8 NFF, 796-800
3D Interchange File (see SGI)
3D Studio, 15, 282-309
3DS (see 3D Studio)
8-, 15-, 16-, 24-bit color, 1061
8-mm recording format, 1061
8SVX file format, 229

absolute format, 35
accelerator card, 1062
A/D conversion, 223
Adaptive Differential Pulse Code Modulation (ADPCM), 226
adaptive encoding, 157, 1062
  for audio information, 226
additive color systems, 47, 1062
ADI (see AutoCAD DXF)
Adobe
  Illustrator, 235-249
  LZW compression and, 177
  Photoshop, 250-260
  Streamline, 105
ADPCM (Adaptive Differential Pulse Code Modulation), 226
Advanced Programming Interface (see API)
AI (see Adobe Illustrator)
AIFF (Audio Interchange File Format), 229
  and QuickTime, 749
aliasing, 104, 1062
alignment of elements within structure, 110
alpha channel, 52, 1062
alphanumeric fonts, 17-20
America Online (AOL), 1043-1044
  LZW agreement with, 177
American Standards National Institute (ANSI), 330
Amiga IFF, 122
Amiga Paint (see IFF)
AMSC's SAF format (see SAF)
Andrew Formats (see CMU file formats)
Andrew Toolkit API, 336-341
ANI (see Atari ST file formats)
Animatic Film, 262
animation, 13, 222, 1062
  frames, 224, 402, 1076
Animation Compressor method, 749
animation file formats, 3, 372, 604
  Atari ST, 262-275
  DEGAS (see Atari ST file formats)
  DVM Movie, 372-375
  FLI, 401-416
  GRASP, 451-456
  MIFF, 599
  MPEG, 604
  NEOchrome (.ANI), 270
  POV, 342, 720-724
  QuickTime, 229, 498
  SGI, 806-811, 818
  TTDDD, 909-913
  Wavefront RLA, 953-963
  (see also DPX; multimedia file formats; video file formats)
ANM (see Atari ST file formats)
ANSI (American Standards National Institute), 330
anti-aliasing, 1062
AOL (America Online), 1043-1044

Ex.1007 / Page 1124 of 1158
APPLE INC.

AOL (America Online) (cont'd)
LZW agreement with, 177
aperture mask, 1063
API (Advanced Programming Interface), 345
Andrew Toolkit, 336-341
and GEM VDI, 423
Archie, 1019-1020
archivers, error checking, 126
archiving programs
arj, 170
lzh, 217
pkzip, 170, 217, 228, 437
zoo, 170, 217, 228, 437
arithmetic encoding, 199
arj program, 170
array of pixels, 1063
ART compression, 23, 210-212, 1063
artifact (in image), 1063
ASC (see 3D Studio)
ASCII data, 17
(see also fonts)
ASCII readers, 108
aspect ratio, 1063
astronomical file formats
FITS, 392-400, 926
PDS, 678-682
VICAR2, 921-924
astronomy imaging resources, 1037-1039
asymmetric compression, 156
Atari ST file formats, 262-274
.AU file format, 229
audio file formats, 16, 225, 229
8SVX, 229
AIFF, 229, 749
.AU, 229
AVSS (see Intel DVI)
.HCOM, 229
Microsoft RIFF, 229, 554-563
.SND, 229
.VOC, 229
audio information, 221
binary representation of, 226
compression, 226
digital, 225-229
Audio Interchange File Format (AIFF), 229
and QuickTime, 749
Audio/Video Support System (see Intel DVI)
authorship (see copyrights; patents; trademarks)
AutoCAD DXB (see AutoCAD DXF)
AutoCAD DXF (Data Exchange Format), 13, 77, 82, 108, 276-281
conversion to another format, 103
Autodesk
3D Studio (see 3D Studio)
Animator Pro, 402
DXF (see AutoCAD DXF)
AVI (see Microsoft RIFF)
AVS (see Dore Raster File Format; Intel DVI)
AVSS (see Intel DVI)

Ballistic Research Laboratory (BRL-) CAD, 315-317
banding, 1063
bands (see strips)
baseline JPEG, 194-198
BBS (Bulletin Board System), 1044-1045
BBS (Bulletin Board Systems), 1063
BDF (Bitmap Distribution Format), 310-314
Beta video format, 1063
Bezier splines, 82, 235, 1064
BIG file format, and VIFF, 926
big-endian byte ordering, 92, 1064
and header reading, 115
on a little-endian machine, 112
bi-level displays, 29, 1064
binary file formats
DXB (see AutoCAD DXF)
Macintosh PICT, 103, 544-548
.MOD, 947
SGO, 818
XBM, 926
binary readers, 108
binary representation, 120
of sound, 226
Binary Universal Form for the Representation of Meteorological Data (BUFR), 318-320
biomedical imaging resources, 1034-1036
bit
depth, 9, 1064
order, 94-95, 1064
per pixel (bpp), 1064
plane, 1064

Ex.1007 / Page 1125 of 1158
APPLE INC.

bit (cont'd)
sex, 95, 1064
bit-level RLE, 164
bitmap data, 8
Bitmap Distribution Format (BDF), 310-314
bitmap file formats, 3, 430
BMP (see Microsoft Windows BMP)
DPX, 350-367
EPSI (see Encapsulated PostScript)
GEM Raster, 169, 417-422
Lumena Paint (BPX, PIX), 532-536
Microsoft RIFF, 229, 554-563
MIFF, 599
OS/2 BMP, 631-650
PNG, 150, 700-719
Presentation Manager Metafile, 725
SPIFF, 202, 822-837
TIFF (see TIFF)
Utah RLE, 917
XBM, 122, 983
bitmap files, 9, 55-74, 1065
compressing, 154
conversion
to bitmap, 102
to metafile, 105
to vector, 105
organization of, 55-57
and pixel data, 74
scaling, 74
strips and tiles, 69
bitmap fonts, 18
bitmap headers
Kofax Raster format, 63
Microsoft Windows BMP, 62
reading speed, 65
Sun Raster format, 63
bitmap images, 7-24, 1065
converting to fractal data, 213
monochrome, 73
bit-ordering schemes, 94
Black, White, and Gray color model, 50
blocks, 1067
BMP (see Microsoft Windows BMP; OS/2 Bitmap)
BND (see Microsoft RIFF)
Bob (see Vivid and Bob)
bpp (bits per pixel), 1064
BPX (see Lumena Paint)
BRL-CAD (Ballistic Research Laboratory CAD), 315-317
broadcast quality, 1065
browsers, web (see World Wide Web)
BSAVE (see Pictor PC Paint)
BUFR (Binary Universal Form for the Representation of Meteorological Data), 318-320
bugs, corrupting graphics files, 123
Bulletin Board System (BBS), 1044-1045, 1063
BW (see SGI)
byte ordering, 91-93, 1065
big-endian, 92, 112, 115, 1064
little-endian, 92, 112, 1083
LSB (least significant byte), 92
middle-endian, 92
MSB (most significant byte), 92
byte-level RLE, 164

C16 (see Intel DVI)
CAD (Computer Aided Design), 77, 1070
BRL-CAD, 315-317
CALS (see CALS raster format)
DXF (see AutoCAD DXF)
graphics, 222
and SGI, 806
CAL (see CALS raster format)
CALS raster format, 321-329
and bit order, 95
and PostScript PDL, 322
Type II, 71
case sensitivity and filenames, 98
CCDs (charge coupled devices), 1065
CCIR-601 digital television standards, 605, 610
CCITT encoding schemes, 16, 23, 179-191
Group 3, 179
Group 4, 179
JBIG versus, 206
as non-adaptive, 180
tips for designing, 188
CD-DA (Compact Disc–Digital Audio) standard, 221, 228, 1066
CD-I (Compact Disc–Interactive) standard, 221, 1066
CD-R (Compact Disc–Recordable or Compact Disc–Write Once) standard, 221, 1066

Ex.1007 / Page 1126 of 1158
APPLE INC.

CD-ROMs (Compact Disc–Read Only Memory), 1066
Kodak Photo CD, 516-519
and QuickTime, 747
of spacecraft data, 679
for storing multimedia information, 220
CDs (compact discs), 221, 1069
and MPEG, 605
CD-XA (Compact Disc–Extended Architecture) standard, 221, 1066
CE1, CE2, CE3 (see Atari ST file formats)
CGM (Computer Graphics Metafile), 13, 77, 81-83, 330-335
converting to another format, 104
and GKS standard, 331
and Lotus PIC, 529
character data, 17-20
chemical imaging resources, 1034-1036
chroma, 1066
chrominance components, 196, 1066
CHT (see Harvard Graphics)
chunks, 1067
3DS file (see 3D Studio)
in Microsoft RIFF, 555
CIE (Commission Internationale de l'E'clairage), 1067
CLP (see GRASP; Pictor PC Paint)
CLUT (Color Look-Up Table), 1084
CMI, CMQ (see Intel DVI)
CMU file formats, 336-341
CMY (Cyan/Magenta/Yellow), 49, 1067
CMY file format (see Intel DVI)
CMYK (Cyan/Magenta/Yellow/Key), 1067
codec, 1067
color, 45-51
attributes, 80
books about, 52
calibration, 1067
channels, 35, 1068
correction tables, 56, 1068
definition scheme, 32, 1068
few colored-data on many-colored device, 30
fullcolor, 29, 1076
gamut, 47, 1068
hicolor, 29, 1076
how we see, 45
map, 32, 1084
palettes, 32, 42
plane, 60
representation, 47-51
sex problems, 95
spaces, 35, 1068
specification, 31-45
subtraction, 47
systems, 47, 1062
tables, 32, 1084
truecolor (see truecolor)
values, 1089
Color Look-Up Table (CLUT), 1084
color models, 37, 1068
Black, White, and Gray, 50
CMY, 49
HSB, 1079
HSI, 1079
HSL, 1079
HSV, 49, 1079
RGB, 35
YUV, 50
ColoRIX VGA Paint (see RIX)
compact discs (CDs), 221, 1069
and MPEG, 605
standards (see CD-DA; CD-I; CR-R; CD-XA)
Compact Video Compressor method, 748
Compatible Bitmap (see Microsoft Windows BMP)
comp.compression newsgroup, 217, 611
comp.graphics newsgroup, 611, 1029
and MTV, 613
comp.graphics.misc newsgroup, 141
comp.multimedia newsgroup, 611
component selective refinement, JPEG, 201
component video, 1069
composite
colors, 35, 1069
image tiling, 202
video, 1069
compress program, 170
compression, 22, 153-218, 1071
adaptive, semi-adaptive, and non-adaptive, 156
algorithms, 23
ART, 23, 154, 210-212, 1063
artifact, 1069
CCITT (see CCITT)

Ex.1007 / Page 1127 of 1158
APPLE INC.

compression (cont'd)
Discrete Cosine Transform, 154, 194, 225, 1071
fractal image, 154, 212-217
JBIG, 154, 205-210
lossless (see lossless compression)
lossy (see lossy compression)
LZW (see LZW)
MR and MMR, 180, 205
negative, 166
physical and logical, 155
proprietary methods of, 128
ratio, 155
RLE (see RLE)
symmetric and asymmetric, 156
type, 60
CompuServe, 1041-1043
Unisys and, 174
Computer Aided Acquisition and Logistics Support (see CALS Raster)
computer graphics, 27-54
books about, 52
Computer Graphics Metafile (see CGM)
computer-aided design (see CAD)
ComputerEyes Raw Data Format, 263
conforming software, 120
Constrained Parameters Bitstreams (CPBs), 606
contiguous data, 69-70, 1070
convenience revisions, 59, 1070
conversion applications, 102
converted file, 102
convolution, 30, 1070
copy protection, 131
copyrights, 145-151, 1070
on graphics files, 145, 150
object code, 147
rights of ownership, 149
Corel Trace, 105
corruption of graphics files, 121-126
count value, 166
CPBs (Constrained Parameters Bitstreams), 606
CRC (Cyclic Redundancy Check), 1070
cross-coding, 162
CRTs, 75
cryptography, 127-136
misconceptions and benefits, 131
private versus public key, 132
CUT (see Dr. Halo)
Cyber Paint Sequence, 264
Cyclic Redundancy Check (CRC), 1070

D-2 video recording format, 1070
DAT (Digital Audio Tape) systems, 226
data
compression (see compression)
contiguous, 69-70
corrupt, in graphics files, 123
dictionary, 170
elements of, 20, 1071
encoding, 22, 60, 153, 1071, 1084
encryption, 1071
Data Interchange Format (see Lotus DIF)
DataBeam DBX and HiJaak Conversion Utility, 487
DBW_uRay (see uRay)
DCT (Discrete Cosine Transform), 154, 194, 1071
and digital video, 225
DCX (see PCX)
DDB (Device Dependent Bitmap), 1071
DeBabelizer, 22
decimation, 1071
decompressor (data), 153
decryption, 127-136
DEGAS (see Atari ST file formats)
delta compression, 222, 225
DEM file format and VIFF, 926
DIB (see Microsoft Windows Bitmap; OS/2 Bitmap)
dichroic filter, 1072
dictionary (data), 170
dictionary-based encoding algorithm, 170
DIF (see Lotus DIF)
differencing, 172
Differential Pulse Code Modulation (DPCM), 199, 226
digital audio, 225-229
(see also audio file formats; audio information)
Digital Audio Tapes (DAT) and MPEG, 605
digital signatures, 126, 131, 135, 1072
digital video, 223-225, 1072
Digital Video Interface (see Intel DVI)
digitizing, 1072
analog-to-digital, 223
audio-to-digital, 226

Ex.1007 / Page 1128 of 1158
APPLE INC.

digitizing (cont'd)
device, 1072
(see also video; video file formats)
Discrete Cosine Transform (see DCT)
display
bi-level, 29
environment, 122
hicolor, 1078
rights, 149
surface, 1072
truecolor, 1096
Display PostScript, 377
distribution rights, 149
dithering, 102, 1072
DKB ray trace application, 342
DLG file format and VIFF, 926
Dore Raster File Format (Dore RFF), 344-349
dot pitch, 1073
down bit ordering, 94, 1073
downsampling pixel groups in JPEG, 196
DPCM (Differential Pulse Code Modulation), 199, 226
DPX, 350-367
copyright notices in, 150
Dr. Halo, 368-371
drawing surface, 61
drum scanner, 1073
DVI (see Intel DVI)
DVM Movie, 372-375
DXB (see AutoCAD DXF)
DXF (see AutoCAD DXF)

edge detection, 105
EDIP (Electronic Document Image Processing), 1073
EGA (Enhanced Graphics Adapter) and PCX, 663
ELAS file format and VIFF, 926
elecronic mail (see email)
Electronic Arts Interchange File Format (see IFF)
Electronic Document Image Processing (EDIP), 1073
element attributes, 80
email (electronic mail), 1010-1011
mailing lists, 1017-1018
Encapsulated PostScript (EPSF), 108, 376-385
and CALS, 322
and Lumena Paint, 532
and WMF, 592
enclosed elements, 80
encoding, 155, 1005-1009
adaptive, 157, 226, 1062
arithmetic, 199
dictionary-based, 170
Huffman, 16, 179
progressive, 208
sequential, 208
encryption, 127-136
Enhanced Graphics Adapter (EGA) and PCX, 663
EOF marker, 124
EOL (End Of Line) for CCITT Group 3, 184
.EPI (see Encapsulated PostScript)
EPICenter (Electronic Photography and Imaging Center), 860
EPS, EPSF, EPSI (see Encapsulated PostScript)
errors, detecting (see corruption)
etiquette (see netiquette)
Excel, 77
explicit copyrights, 146
extended vector formats, 15
extensions, file, 122

FaceSaver, 386-388
FAQ (Frequently Asked Questions) listings, 1014
FAX file formats, 7, 389, 1074
and CCITT encoding, 179
field (in file), 21, 1074
FII (see FLI Animation)
file format
error-detecting reader, 125
proprietary, 128
writers, 126
writing specifications for, 141-144
file identifier value (ID), 58, 1074
File Transfer Protocol (see FTP)
filenames, 97
files
detecting corruption of, 124-125
elements of, 20, 1074
extensions to, 122
fully conforming, 120
reader for, 122
test, 120
unexpected characters in, 124

Ex.1007 / Page 1129 of 1158
APPLE INC.

fill attributes, 80, 1074
fill colors, 80
filler, 61
filter, 107
fingerprint (see digital signatures)
FITS, 392-400
  and VIFF, 926
flag value, 166
FLC (see FLI Animation)
Flexible Image Transport System (see FITS)
FLI Animation, 401-416
Flic (see FLI Animation)
FLM (see Atari ST file formats)
floating-point formats, 94
FM 92-VIII Ext. GRIB (see GRIB)
FM 94-IX Ext. BUFR (see BUFR1)
FNT (see GRASP)
fonts, 17-20
  Hershey character sets, 19, 84
footers, 72, 81
formal copyrights, 146
format
  absolute, 35
  conversion, 101-106
  creator, 1075
  graphics file (see graphics file formats)
  image (see image file formats)
Fractal, 24
fractal image compression, 212-217, 1075
Fractal Transform, 213
frames (animation), 224, 402, 1076
Frequently Asked Questions (FAQ) listings, 1014
FTI (see FITS)
FTP (File Transfer Protocol), 1076
  graphics archives, 1018, 1032, 1040
    (see also graphics files, resources for)
fullcolor, 29, 1076
full-motion video, 1077
fully conforming software, 120
fuzzy logic, 1077

G31D encoding, 181
G32D encoding, 186
G42D encoding, 188
GDI (see GEM VDI)
GEM Raster (IMG), 169, 417-422
GEM VDI (vector) metafile format, 103, 423
GEMDOS format, 269
geometrical shapes (see vector files)
  three-dimensional (see 3D formats)
geophysical imaging resources, 1036-1037
GIF (Graphics Interchange Format), 429-450, 1022
  compressing with archive programs, 1016
  copyright notices in, 150
  file identifier, 59
  for test purposes, 120
  LZW compression of (see LZW)
  versus JPEG, 191
GKS (Graphical Kernel System), 331
GL (see GRASP)
government file format (see CALS raster)
graphic work, 1077
Graphical Kernel System (GKS) standard, 331
Graphical System for Presentation (see GRASP)
Graphical User Interface (see GUI)
graphics, 4, 27-54, 1077
  adapter/card (see video display card)
  CAD (see CAD)
  general resources for, 1029-1040
  logos and trademarks, 144
Graphics Compressor method, 749
graphics file formats, 3, 55, 1078
  animation (see animation file formats)
  bitmap (see bitmap file formats)
  designing your own, 139
  immortality of, xvii
  metafile (see metafile formats)
  multimedia (see multimedia file formats)
  scene description (see scene description file formats)
  vector (see vector file formats)
graphics files, 102, 1078
  copyrights, patents, and trademarks, 144-151
  corruption of, 121-126
  embedding using HTML, 1023
  encryption of, 127-136
  physical protection of, 127
  protecting with PGP, 134-136
  reading and writing, 125
  resources for, 1005-1040

Ex.1007 / Page 1130 of 1158
APPLE INC.

graphics files (cont'd)
testing, 120
viruses in, 137-138
over the Web, 1022
Graphics Interchange Format (see GIF)
GRASP (GRAphical System for Presentation), 451-456
graymap, 9, 1078
gray-scale, 1078
on Kodak XL7700 printer, 520
Green Book, 221
GRIB (GRIdded Binary), 457-459
Grid File Format (see VIS-5D)
Gridded Binary (see GRIB)
Group 3 One-Dimensional encoding, 181-185
Group 3 Two-Dimensional encoding, 186
Group 4 Two-Dimensional encoding, 188
GUI (Graphical User Interface)
applications, 3
Microsoft Windows, 592, 881
and RIFF, 554
Open Look, 838
SunView, 838

Haeberli (see SGI)
HALO Image File Format Library, 368
Harvard Graphics, 460
.HCOM file format, 229
HDF (Hierarchical Data Format), 462
headers
of vector files, 78
reading, 108-116
Hershey character sets, 19, 84
hicolor, 29, 1076
display, 1078
hierarchical selective refinement, JPEG, 201
HiJaak Graphics File Conversion Utility, 22, 487
hints (see fonts)
HLS (Hue/Lightness/Saturation), 1078
HPGL and HiJaak Conversion Utility, 487
HPPCL and HiJaak Conversion Utility, 487
HSB (Hue/Saturation/Brightness), 1079
HSI (Hue/Saturation/Intensity), 1079
HSL (Hue/Saturation/Luminosity), 1079
HSV (Hue/Saturation/Value), 49, 1079
HTML (Hypertext Markup Language), 12, 138, 1021
embedding graphics files with, 1023
(see also page description languages)
HTTP (Hypertext Transfer Protocol), 16
hue, 1079
Huffman encoding, 16, 179
hybrid database, 14, 1079
hybrid text, 14, 1079
hypermedia, 15
hypertext, 14, 1079
Hypertext Markup Language (HTML), 12, 138, 1021
embedding graphics files with, 1023
Hypertext Transfer Protocol (HTTP), 16

I8, I16 (see Intel DVI)
IBM Multimedia Tool Series, 402
IC1, IC2, IC3 (see Atari ST file formats)
ICB (see TGA)
ICC (see Kodak YCC)
ICO (see Sun Icon)
ID value, file, 58
IDL terminal communication protocol, 699
IFF (Interchange File Format), 465-483
Amiga IFF, 122
and byte order, 93
copyright notices in, 150
and Microsoft RIFF, 555
IGES (Initial Graphics Exchange Specification), 485
ILBM (see IFF)
ILM (see IFF)
IMA, IMB, IMC (see Intel DVI)
image bitmap, 10, 1080
image data, 1080
image elements, 75
image file formats, 35, 860
Atari ST (see Atari ST file formats)
CALS (see CALS raster)
Dore RFF, 344-349
EPSF (see Encapsulated PostScript)
FaceSaver, 386-388
FITS, 392-400
GIF (see GIF)
.ICBA, 861
Lumena Paint, 532-536
MSP, 549
NEOchrome (.NEO), 269
P3D, 651-657
PDS, 678-682
Pictor PC Paint, 683-694

Ex.1007 / Page 1131 of 1158
APPLE INC.

image file formats (cont'd)
raw, 55
SGI (see SGI)
STAD (.PAC), 271
stream-based, 430
SunRaster (see SunRaster)
TGA, 51
VICAR2, 393, 921-924
Wavefront RLA, 953-963
XPM (X PixMap), 987-991
XWD (X Window Dump), 992-997
(see also animation file formats; multimedia file formats; video file formats)
image file index, 73
image height, 60
image length, 60
image offset table, 73
image portion, 10
image tiling, 201
image width, 60
ImageMagick toolkit
and MIFF, 599
images, 5, 10, 1080
examples of, xx
interlacing, 438
x origin of, 60
Imagic Film/Picture Format, 268
imaging resources
astronomy, 1037-1039
biomedical, 1034-1036
chemical, 1034-1036
geophysical, 1036-1037
meteorological, 1036-1037
oceanographic, 1036-1037
IMG (see GEM Raster; Intel DVI)
IMI, IMI, IMQ, IMR, IMY (see Intel DVI)
implicit copyrights, 146
Independent JPEG Group, 193
index map, 32, 1081
index values, 32, 1081
indirect color, 1081
indirect storage, 32
information devices
active, 219, 1062
passive, 219, 1088
Initial Graphics Exchange Specification (see IGES)
input (of graphics process), 4, 1081
Inset IGF and HiJaak Conversion Utility, 487
Inset PIX, 487-496
Integrated Services Digital Network (IDSN), 191
Intel DVI (Digital Video Interactive), 497-509
AVSS (.AVS), 499
Intel Real-Time Video (see Intel DVI)
Interchange File Format (see IFF)
interframe encoding, 1081
interlaced images, 438
International Telecommunications Union (ITU), 1082
interpolation, 1082
intraframe encoding, 1082
Inventor (see SGI)
IRIS (see SGI)
IRIS Inventor 3d Toolkit, 806
ISDN (Integrated Services Digital Network) lines, 191
ISO (International Standards Organization), 191, 330
and MPEG, 604
ISO-9660 standard, 1082
ITU (International Telecommuncations Union), 1082

jaggies, 104
Java, 12
JBIG (Joint Bi-level Image Experts Group) compression, 23, 154, 205-210
JBIG-KIT, 209
JFI, JFIF (see JPEG File Interchange Format; SPIFF)
JPEG (Joint Photographic Experts Group), 23, 823
baseline, 194-198
component selective refinement, 201
File Interchange Format (JFIF), 72, 191-205, 510-515
extensions to, 198-202
versus fractal compression, 214
Tiled Image Pyramid (JTIP), 201
(see also SPIFF)
JPG (see JPEG File Interchange Format; SPIFF)
JTIP (JPEG Tiled Image Pyramid), 201

Ex.1007 / Page 1132 of 1158
APPLE INC.

key points, 7, 1083
Khoros Visualization/Image File Format (VIFF), 925-934
Kodak Cineon file format (see DPX)
Kodak Photo CD, 516-519
Kodak XL7700 printer, 520
Kodak YCC format, 520
Kofax Raster format, 63

landmark, 1083
LBM (see IFF)
legal issues
  copyrights, patents, and trademarks, 144-151
  graphics on Web, 1029
  LZW compression, 174-177
Lempel-Zev-Welch compression (see LZW)
lha program, 170
license notice, 148
licensing, Unisys LZW, 174-177
literal format, 35
little-endian byte ordering, 92, 1083
  and header reading, 115
  on a big-endian machine, 112
logical compression, 155
logical pixels, 1083
logos and trademarks, 144
look-up table, 32, 1084
lossless compression, 157, 1084
  of audio information, 228
  with JPEG, 199
lossy compression, 157, 1084
  of audio information, 228
  with JPEG, 192
Lotus 1-2-3, 77
  and Lotus DIF, 523
  and Lotus PIC, 529
Lotus DIF (Data Interchange Format), 77, 522
  conversion to another format, 103
Lotus PIC, 529
Lotus Print Graph, 529
LSB (least significant byte), 92, 1084
Lumena Paint, 532-536
luminance, 1084
LUT (Look-Up Table), 1084
LZ77 and LZ78 compression algorithms, 170
lzh program, 217
LZMW data compression, 173
LZW (Lempel-Ziv-Welch) compression, 23, 170-179, 431, 437
  GIF format licensing, 139
  legal issues, 174-177
  variations on, 173

MAC (see Macintosh Paint)
MacBinary standards, 538
Machine Independent File Format (MIFF), 599
Macintosh Paint, 537-543
  and byte order, 93
Macintosh PICT (Macintosh Picture), 13, 103, 544-548
  and previewing EPSF files, 381
MacPaint (see Macintosh Paint)
magic numbers, 1074
magic values, 58, 1084
  errors in, 125
mail (see email)
mailing lists, 1017-1018
  resources via, 1031
makeup code words, 182
MathCAD and HiJaak Conversion Utility, 487
MATLAB and VIFF, 926
McIDAS (see VIS-5D)
MET (Presentation Manager Metafile), 725-729
metadata, 463, 1085
metafile formats, 3
  CGM (see CGM)
  GEM VDI (see GEM VDI)
  Microsoft RTF, 564-568
  Microsoft Windows Metafile, 592-598
  WPG, 169, 964-982
metafiles, 13, 87, 1085
  ASCII, 88
  compression of, 154
  conversion to bitmap, 104
  conversion to metafile, 104
  conversion to vector, 105
meteorological file formats
  BUFR, 318-320
  GRIB, 457-459
  VIS-5D, 935-942
meteorological imaging resources, 1036-1037
Microray (see uRay)

Ex.1007 / Page 1133 of 1158
APPLE INC.

Microsoft Paint (MSP), 549
Microsoft RIFF (Resource Interchange File Format), 229, 554-563
Microsoft RTF (Rich Text Format), 564-568
Microsoft SYLK (Symbolic Link), 13, 77, 569
Microsoft Video (for Windows), 402
Microsoft Waveform, 229
Microsoft Windows BMP, 130, 572-591
  compression of, 154
  copyright notices in, 150
  and Run-Length Encoding, 160
  Version 1.x, 62
  versus JPEG, 191
Microsoft Windows Metafile (WMF), 105, 592-598
  converting to another format, 104
  and previewing EPSF files, 381
middle-endian byte ordering, 92
MIDI (Musical Instrument Digital Interface), 220, 230, 1085
  and .RMI files, 555
MIFF (Machine Independent File Format), 599
MIME (Multipurpose Internet Mail Extensions), 1010, 1021, 1026, 1040
MJPEG (Motion JPEG), 606
MMR (Modified Modified Read) compression algorithm, 180, 205
  (see also JBIG compression)
.MOD file format, 947
modification rights, 149
monochrome, 1085
  bitmaps, 73
motion compensation, 225
motion-video data, 605
movie resource, 749
Movie Toolbox, 747
MPEG (Motion Picture Experts Group), 604
MPEG-1, MPEG-2 (see MPEG)
MPG (see MPEG)
MR (Modified Read) compression algorithm, 205
  (see also JBIG compression)
MSB (most significant byte), 92, 1085
MS-DOS, 19
MSP (see Microsoft Paint)
MTV, 613-616
multi-channel palette, 1086
multimedia, 14, 26, 219-231, 1086
  application files, viruses in, 138
  and CD-ROMs, 220
  (see also hypermedia)
multimedia file formats, 3, 372
  Intel DVI, 497-509
  Microsoft RIFF, 229, 554-563
  MPEG, 605
  (see also animation file formats; video file formats)
multi-platform file formats
  Andrew Toolkit (CMU file formats), 336
  CALS (see CALS raster)
  CGM (see CGM)
  EPSF (see Encapsulated PostScript))
  FITS, 392-400, 926
  GIF (see GIF)
  HDF, 463
  Microsoft RTF, 565-568
  MIFF, 599
  MTV, 613-616
  P3D, 651-657
  PCX (see PCX)
  Pixar RIB, 695-697
  QuickTime (see QuickTime)
  TIFF (see TIFF)
  YAODL, 812-817
multiple-channel, 38
Multipurpose Internet Mail Extensions (see MIME)
Musical Instrucment Digital Interface (see MIDI)

NAPLPS (North American Presentation Layer Protocol Syntax), 617
National Center for Supercomputer Applications (NCSA), 462
National Television Standards Committee (see NTSC)
n-bit color (n=4,8,15,16,24), 1061
NCGA (National Computer Graphics Association) and IGES, 486
negative compression, 166
NEOchrome animation, 26, 269-270
  (see also Atari ST file formats)
netiquette
  Archie, 1020
  email, 1011
  FTP, 1018
  USENET posting, 1015

Ex.1007 / Page 1134 of 1158
APPLE INC.

Neutral File Format (see NFF)
newsgroups (general), 1012
newsgroups (graphics-related)
comp.compression, 217, 611
comp.graphics, 611, 613, 1029
comp.graphics.misc, 141
comp.multimedia, 611
list of, 1030
sci.astro.fits, 399
sci.data.formats, 399
sci.image.processing, 1029
NewTek's Lightwave, 15
NFF (Neutral File Format), 620-626
and PRT, 730
and RTrace, 776
noise removal, 172
non-adaptive compression, 156
for audio information, 226
North American Presentation Layer Protocol Syntax (NAPLPS), 617
NTSC (National Television Standards Committee), 196, 224, 1086
and TGA, 860
numerical format: Lotus DIF, 522

OBJ (see Wavefront OBJ)
object code copyright, 147
object data, 10, 75
Object Description Language (ODL) and PDS, 678
Object File Format (OFF), 627
object-based file systems, 3
object-oriented databases, 807
object-oriented files, viruses in, 138
oceanographic imaging resources, 1036-1037
ODA, ODIF, 321
ODL (Object Description Language) and PDS, 678
OFF (Object File Format), 627
Office Document formats (ORA, ODIF), 321
offset values, out of range, 125
Open Look GUI and Sun Icon, 838
order, bit, 94-95
OS/2 BMP, 631-650
output device, 4, 102, 1086
language, 7, 1086
output (of graphics process), 4, 1086
overlay bit, 51, 1087
overlays, 51-52
OVR (see Pictor PC Paint)
ownership (see copyrights; patents; trademarks)

P10 (see Plot-10)
P3D, 651-657
PAC (see Atari ST file formats)
packets, 21, 1087
padding, 61
page description languages (PDLs), 20, 1087
virusus and, 138
page table, 73, 1087
PAL file format (see Dr. Halo)
PAL (Phase Alternation Line) digital television standards, 224, 605, 1087
and TGA, 860
palettes, 63, 1087
color, 32, 42
in vector files, 80
multi-channel, 38, 1086
single-channel, 38, 1093
types of, 39
VGA color, 669
Parallel Ray Trace (see PRT)
parity, 1006
parser, 107
patents on graphics files, 144
PBM utilities, 22, 26, 659
and FaceSaver, 386
and VIFF, 926
(see also PGM; PNM; PPM)
pbmplus (see PBM; PGM; PNM; PPM)
PC Paint (see Pictor PC Paint)
PC Paintbrush File Format (see PCX)
PC1, PC2, PC3 (see Atari ST file formats)
PCC (see PCX)
PCT (see Macintosh PICT)
PCX, 108, 122, 662-677
copyright notices in, 150
and Run-Length Encoding, 160
and testing, 120
PDL, 235
PDS (Planetary Data System), 678-682
pels, 8, 1088
pen styles, 80
Persistance of Vision (see POV)
persistence, 1088
PGM (Portable GreyMap Utilities), 26, 658
(see also PBM; PNM; PPM)

Ex.1007 / Page 1135 of 1158
APPLE INC.

PGP (Pretty Good Privacy), 134-136
Photo CD (see Kodak Photo CD)
Photo Compressor method, 748
Photoshop (see Adobe Photoshop)
PhotoYCC file format (Kodak), 518
physical compression, 155
physical pixels, 27, 1088
PI1, PI2, PI3 (see Atari ST file formats)
PIC (see GRASP; Lotus PIC; Pictor PC Paint)
PICT (see Macintosh PICT)
Pictor PC Paint, 683-694
  and GRASP animation, 455
picture elements, 8
Pittsburgh Supercomputer Center 3D Metafile (see P3D)
PIX (see Inset PIX; Lumena Paint)
Pixar RIB (RenderMan Interface Bytestream), 695-697
pixel-level RLE, 165
pixelmap, 9, 1089
pixels, 8, 1083, 1088
  array of, 1063
  depth of, 60, 1064
  packing, 23, 154, 158-160
  physical versus logical, 27
  three-dimensional, 345
  values of, 1089
pixmap, 9, 1089
pkzip program, 126, 170, 217, 228
  and GIF files, 437
planar data, 67-69
planar files, 1089
Planetary Data System Format (see PDS)
Planetary File Format (see VICAR2)
platform dependencies, 91-99
platform-independent file formats (see multi-platform file formats)
plot file formats: .ADI, 276
Plot-10 file format, 698
PM Bitmap (see OS/2 BMP)
PM BMP, PM DIB (see OS/2 Bitmap)
PNG, 126, 700-719
  copyright notices in, 150
PNM (Portable ANyMap Utilities), 658
PNTG (see Macintosh Paint)
polygon elements (see vector files)
Portable Bitmap Utilities (see PBM; PGM; PNM; PPM)
Portable Bitmap Utilities (PBM), 658-661
Portable Network Graphic format (see PNG)
POS (see SAF)
postage-stamp images, 381
  and Lumena Paint, 534
PostScript, 138, 235, 377-378
  Encapsulated (see Encapsulated PostScript)
  files, 20
  for storing text (in multimedia format), 220
  LZW compression and, 170, 177
  Page Description Language (PDL)
    and BDF, 310
    and CALS, 322
    and EPSF, 376
  (see also page description languages)
POV (Persistence of Vision)
  file format, 720-724
  ray trace application, 342
POV-Ray (see POV)
Powerflip Format (see YAODL)
PPM file format (see POV)
PPM (Portable PixMap Utilities), 26, 658
  (see also PBM; PGM; PNM)
predictive encoding, 1089
Presentation Manager Bitmap (see OS/2 Bitmap)
Presentation Manager Metafile (MET), 725-729
previewing
  Encapsulated PostScript files, 381
  Lumena files, 534
  QuickTime files, 748
primary colors, 47, 1089
primary three-color systems, 47
private key cryptography, 132
Prodigy
  and NAPLPS, 617
  Unisys agreement for LZW, 177
production pipeline, 5, 1090
program code, corrupting graphics files, 123
progressing selective refinement, JPEG, 201
progressive encoding/decoding, 208
progressive image buildup, 198
proprietary information, 128-129
PRT, 730-735
PSD (see Adobe Photoshop)
pseudo-color, 1090
pseudo-color storage, 32

Ex.1007 / Page 1136 of 1158
APPLE INC.

public display rights, 149
public domain, 148
public key cryptography, 132
Pulse Code Modulation (PCM), 226
pyramidal image tiling, 201

Q factor and lossless compression, 199
Q-coder algorithm, 199
  and MIFF, 602
QRT (Quick Ray Trace), 736
QTM (see QuickTime)
quantization, 30, 102, 1090
  artifacts, 1090
  coefficient, 197
  JPEG (see variable quantization, JPEG)
  table, 197
Quattro Pro, 77
  and Lotus DIF, 523
Quick Ray Trace (see QRT)
QuickDraw Picture Format (see Macintosh PICT)
QuickTime, 229, 747-759
  and AIFF, 749
  CD-ROMs and, 747
  and DVI, 498

Radiance, 760
radix-64, 134
RAS (see CALS Raster; Sun Raster)
raster file formats (see bitmap file formats)
rasters, 7, 9, 1064, 1090
Raw Compressor method, 749
raw data, 60, 155
raw formats, 55
ray tracing and MTV, 613
ray tracing file formats
  POV, 720-724
  PRT, 730-735
  QRT, 736
  Radiance, 760
  Rayshade, 764-771
  uRay, 914
  Vivid and Bob, 943-945
Rayshade, 764-771
RDI (see Microsoft RIFF)
READ (Relative Element Address Designate) codes, 187
readers
  ASCII, 105
  binary, 105, 108
  FITS, 396
  FLI, 401, 411
  RIFF, 556
  TIFF, 882, 895
reading graphics data, 107-117, 125
realization, 5, 1090
registered trademarks (see trademarks)
rendering, 5, 1091
  applications, 760
RenderMan Interface Bytestream (see Pixar RIB)
repeat scan line packet, 168
replicating files (see viruses)
representation, 4, 1091
reproduction rights, 149
reserved fields, 61
reserved space, 61
resolution, 1091
Resource Interchange File Format (see Microsoft RIFF)
RFF (see Dore Raster File Format)
RGB format (see Atari ST file formats; SGI)
RGB (Red-Green-Blue) color model, 35, 1091
  and digital video, 224
RGBA (see SGI)
RIB (see Pixar RIB)
Rich Text Format (see Microsoft RTF)
RIFF, RIFX (see Microsoft RIFF)
RIX, 772
RLA, RLB (see Wavefront RLA)
RLE (see run-length encoding)
RMI (see Microsoft RIFF)
RTC (Return To Control) code, 185
RTF (see Microsoft RTF)
RTrace, 776-785
RTV (see Intel DVI)
run count, 160
run lengths, code words for, 182
run value, 160, 166
Run-length Encoded Versions A, B (see Wavefront RLA)
run-length encoding (RLE), 23, 26, 153, 160-170
  and animation, 222
  (see also SGI; Utah RLE; Utah RLE)
runs (strings), 160, 182

Ex.1007 / Page 1137 of 1158
APPLE INC.

SAF, 786-795
sample rate, 223, 227, 1091
sample resolution, 227, 1092
sampling, 1092
saturation, 1092
scaling vector files, 83
scan lines, 9, 1092
scan-line data, 66-67
scan-line tables, 56, 73, 1092
scan-line width, 60
Scanmaster, 180
scanning, 107, 224
scene description file formats, 3, 13, 1093
  NFF, 620-626
  OFF, 627
  P3D, 651-657
  Pixar RIB, 695-697
  POV, 720-724
  QRT, 736
  Radiance, 760
  Rayshade, 764
  RTrace, 776
  SGI Inventor, 807
  uRay, 914
  Vivid and Bob, 943-945
  YAODL, 812-817
scenes, 13
scientific data management and HDF, 462
sci.image.processing newsgroup, 1029
SCN (see RTrace)
screen dumps and XWD, 992
SECAM (Sequential Coleur Avec Memoire) digital television standards, 224, 605, 1093
security, file (see cryptography)
selective refinement, JPEG, 201
semi-adaptive encoder, 157
Sense8 NFF, 796-800
SEQ (see Atari ST file formats)
sequential encoding/decoding, 208
SET (see GRASP)
sex, bit, 95, 1064
sex, color, 95
SFF, 776-785
SGI (Silicon Graphics Inventor), 26, 801-811
  RLE, bit/byte-level, 164
  and SGO, 818
  (see also YAODL)
SGO (Silicon Graphics Object), 818
Showcase (see SGO)
SIGGRAPH, 87
Silicon Graphics Object (see SGO)
simple image tiling, 201
single-channel palette, 38, 1093
size of structure, 111
SLD (see AutoCAD DXF)
slides, format for, 276
SLK (see Microsoft SYLK)
SMPTE DPX (see DPX)
.SND file format, 229
Society of Motion Picture and Television Engineers (SMPTE) DPX (see DPX)
software
  driver, 122
  fully conforming, 120
  public domain, 148
sound (see audio formats)
specifications, file format, 141-144
Sperry Corporation (see Unisys)
SPF (see SPIFF)
SPIFF (Still Picture Interchange File Format), 202, 822-837
  copyright notices in files, 150
spline-based outline fonts, 19
splines, Bezier, 82, 235, 1064
spreadsheet file formats
  Microsoft SYLK, 569
  vector formats, 77
spreadsheet file formats and Lotus DIF, 522
STAD image, 271
Standard Archive Format (see SAF)
stereo sampling, 227
STI terminal communication protocol, 699
Still Picture Interchange File Format (see SPIFF)
storage, 32
stream, 21, 1094
stream-based image format, 430
Streamline (Adobe), 105
string table, 170
strips, 70, 1094
stroke fonts, 19
structure, determining size of, 111
substitutional encoding algorithm, 170
substrings, 170
subtractive color systems, 47, 1094
Sun Icon, 838

Ex.1007 / Page 1138 of 1158
APPLE INC.

SunRaster image, 63, 841-845
and byte order, 93
and VIFF, 926
SunView GUI and Sun Icon, 838
SYLK (see Microsoft SYLK)
Symbolic Link Format (see Microsoft SYLK)
symbols (see fonts)
symmetric compression, 156

T3D (see TDDD)
T3DLIB shareware library, 909
tag, 21, 1094
Tag Image File Format (see TIFF)
Targa Image File (see TGA)
Targa market, 532
TCS terminal communication protocol, 699
TDDD (Turbo Silver 3D Data Description) format, 846-859
technical writing, suggestions for, 143
Tek Plot-10 (see Plot-10)
television standards (digital), 605
terminating code words, 182
test files, 120
text in vector files, 83-84
Textual 3D Data Description (see TTDDD)
TGA (Truevision Targa Image File), 122, 860-879
copyright notices in, 150
and Lumena Paint, 532
and VIFF, 926
three-dimensional formats (see 3D file formats)
TIFF (Tag Image File Format), 72, 108, 120, 122, 880-908
and bit order, 95
and byte order, 93
and CCITT encoding, 179
Class F (FAX) encoding, 186, 390
compression of, 154
copyright notices in files, 150
and Lumena Paint, 532
LZW compression and, 171, 173
LZW compression and, 170, 177
and previewing EPSF files, 381
and Run-Length Encoding, 160
TIFF Developers Toolkit, 174
Type 2 Compression, 185
versus JPEG, 191
and VIFF, 926
tiles, 71, 1095
tiling, image, 201
Tiny image file format, 273
TN1, TN2, TN3 (see Atari ST file formats)
TNY (see Atari ST file formats)
TPIC (see TGA)
trademarks on graphics files, 144
trailers, 72
transform, 1070
translation table, 170
transparency, 51-52, 1095
trichromatic colorimetric, 47, 1095
truecolor, 29
bitmap formats, 35
TGA, 51, 861-879
display, 1096
display adapters, 532
on Kodak XL7700 printer, 520
TTDDD (Textual Three-Dimensional Data Description), 909-913
Turbo Silver 3D Data Description (see TDDD)
tutorials, advice on writing, 143
twip (twentieth of a point), 592
TXT (see GRASP)
Type II CALS raster format, 71

UC1, UC2, UC3 (see Atari ST file formats)
unencoded data, 155
Unisys Corporation, 174-177
UNIX, 19
up bit ordering, 94
uRay, 914
U.S. Product Data Association (USPRO), 486
USENET news, 1011-1017
resources via (see graphics files, resources for)
Utah Raster Toolkit (URT), 917
Utah RLE, 917
utility patents, 145
uuencode and uudecode, 1006-1009, 1016

V.42bis modem compression standard, 170
variable quantization, JPEG, 200
VDA (see TGA)
VDI (see GEM VDI)

Ex.1007 / Page 1139 of 1158
APPLE INC.

vector data, 75, 79-80, 212
vector file formats, 3
  CGM (see CGM)
  DXF (see AutoCAD DXF)
  EPSF (see Encapsulated PostScript)
  and floating-point, 94
  GEM VDI (see GEM VDI)
  Lotus DIF, 522-523
  NFF, 620-626
  POV, 720-724
  Presentation Manager Metafile, 725-727
vector files, 7, 13, 75-85, 1096
  applications of, 77
  compression of, 154
  converting to bitmap files, 104
  converting to metafiles, 105
  converting to vector files, 103
  device dependence of, 76
  extended formats, 15
  pros and cons of, 84
  scaling, 83
  size of, 82
  structure of, 77
  text in, 83-84
  versus bitmap files, 75
vectorization, 105
version marker segments, JPEG, 202
vertical replication packet, 168
VGA color palette, 42
VGA display adapter and PCX, 663
VICAR2 (Video Image Communication and Retrieval), 393, 921-924
video
  digital, 223-225
  full-motion, 1077
  information, 221
Video Compressor method, 748
Video Display Adapter (VDA) (see TGA)
video display processor (VDP) and DVI, 498
video file formats, 372
  FLI, 401-416
  GRASP, 451-456
  Intel DVI, 497-509
  Microsoft RIFF, 554-563
  TGA (see TGA)
  (see also animation files formats; multimedia file formats)
VIFF (Khoros Visualization/Image File Format), 925-934
virtual device, 61
virtual output, 4, 1097
Virtual Reality Modeling Language (VRML), 16
viruses in graphics files, 137-138
VIS-5D, 935-942
VisiCalc spreadsheet program
  and Lotus DIF, 522
Visualization-5D (see VIS-5D)
Vivid and Bob, 943-945
VMG (see RIX)
.VOC file format, 229
voxels, 345, 1097
VRML (Virtual Reality Modeling Language), 16
VST (see TGA)

WAV (see Microsoft RIFF)
Wavefront OBJ, 946-952
Wavefront Object (see Wavefront OBJ)
Wavefront RLA image, 953-963
weather data file formats (see meteorological file formats)
Windows BMP, Windows DIB (see Microsoft Windows BMP)
Windows Metafile (see Microsoft Windows Metafile)
.WK1, .WKS (see Lotus 1-2-3)
WMF (see Microsoft Windows Metafile)
WordPerfect Graphics Metafile (WPG), 13, 169, 964-982
work for hire, copyright and, 146
WorkdToolkit NFF (see Sense8 NFF)
Working Group on Astronomical Software (WGAS), 399
World Meteorological Organization (WMO), 318
World Wide Web (WWW), 1020, 1034
  browser helper applications, 1025
  browsers, using CD-ROM with, 1052-1053
  general graphics resources, 1020, 1040
    (see also graphics files, resources for)
WPG (see WordPerfect Graphics Metafile)
writing
  file format specifications, 141-144

Ex.1007 / Page 1140 of 1158
APPLE INC.

writing (cont'd)
file formats, 139-141
graphics files, 117-120, 125
suggestions for, 143
WWW (World Wide Web), 1034

X Bitmap (see XBM)
x origin of image, 60
X PixMap (see XPM)
X Windows Dump (see XWD)
XBM, 122, 983
and VIFF, 926
.XLS, 77
XPM (X PixMap), 987-991
XWD (X Window Dump), 992-997

y origin of image, 60
YAODL (Yet Another Object Description Language), 812-817
YCbCr color model
and digital video, 224
and JPEG, 195
YCC (see Kodak YCC)
Yellow Book, 221
Yet Another Object Description Language (see YAODL)
YIQ, 1098
and digital video, 224
YUV (Y-signal, U-signal, and V-signal) color model, 50
and digital video, 224
and JPEG, 195
YUV (Y-signal, U-signal, V-signal), 1098

ZBR, 998-1002
Zebra Metafile (see ZBR)
zoo program, 126, 170, 217, 228
and GIF files, 437

Ex.1007 / Page 1141 of 1158
APPLE INC.

About the Authors

James D. Murray started his computer career in 1981 on a Version 6 UNIX system running on a PDP-11/45 and programming in C. Over the years he has specialized in serial communications, image processing and analysis, and UNIX systems programming. Currently he works for a telecommunications company developing network management stations and as a staff writer for O'Reilly & Associates. James lives in Southern California, has a degree in cultural anthropology, has studied computer science and both Western and non-Western music, and practices the Japanese martial arts of Aikido and Iaido (Japanese swordsmanship).

William vanRyper has been writing graphics software since 1982 and was chief scientist at Flamingo Graphics. He is the president of uvr, a private consulting firm, and has designed and implemented drawing packages, animation software, and visualization systems for many major corporate clients. He is currently working on systems for the creation of 3D sacred geometry animations.

Colophon

The cover was designed and produced by Edie Freedman in QuarkXpress 3.3. The cover image is from the Dover Pictorial Archive. Inside layouts were designed by Jennifer Niederst, and modified by Nancy Priest. The cover and the contents of the book were formatted using Copperplate and New Baskerville fonts from Adobe.

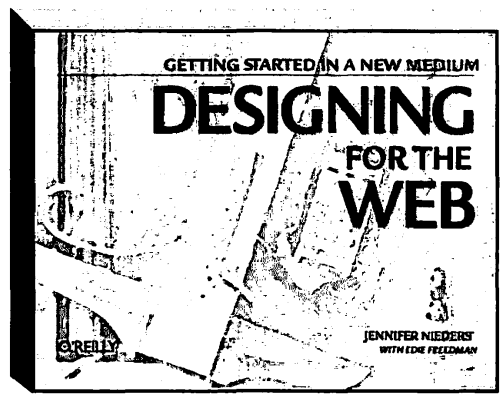
Text was prepared in SGML using the DocBook 2.1 DTD. The print version of this book was created by translating the SGML source into a set of gtroff macros using a filter developed at ORA by Norman Walsh. Steve Talbott designed and wrote the underlying macro set on the basis of the GNU gtroff -gs macros; Lenny Muellner adapted them to SGML and implemented the book design. The GNU groff text formatter version 1.08 was used to generate PostScript output.

Norman Walsh designed the GFF application and wrote the Microsoft Windows and UNIX implementations. Ken DeCanio (*Ken@LittleTrain.com*) at Little Train Software wrote the Macintosh implementation.

The figures were created by Chris Reilley in Aldus Freehand 4.0, and imported into the final page layouts.



THE IDEAL REFERENCE FOR DESIGNERS WHO NEED TO HIT THE GROUND RUNNING



You may have little or no experience designing for the Web but know you need to get up-to-speed fast. This book will show you what you need to know to design effective Web pages, from working with HTML to designing for a variety of browsers.

The Web design book you'll refer to again and again. Written with a designer's point of view, *Designing for the Web* shows you how to work with HTML documents, deal with the special problems of presenting information online, incorporate images into Web pages, and design for different browsers. Pick up a copy at your local bookstore or call O'Reilly at 1-800-889-8969 Weekdays 6AM-5PM PST.

Check out our online catalog at **http://www.ora.com/**

O'REILLY™

TO ORDER: **800-889-8969** *(CREDIT CARD ORDERS ONLY);* **ORDER@ORA.COM**



Books from O'Reilly & Associates, Inc.

FALL/WINTER 1995-96

Basics

Our UNIX in a Nutshell *guides are the most comprehensive quick reference on the market—a must for every* UNIX *user. No matter what system you use, we've got a version to cover your needs.*

UNIX in a Nutshell: System V Edition

By Daniel Gilly & the staff of O'Reilly & Associates
2nd Edition June 1992
444 pages, ISBN 1-56592-001-5



You may have seen UNIX quick-reference guides, but you've never seen anything like *UNIX in a Nutshell*. Not a scaled-down quick reference of common commands, *UNIX in a Nutshell* is a complete reference containing all commands and options, along with generous descriptions and examples that put the commands in context. For all but the thorniest UNIX problems, this one reference should be all the documentation you need. Covers System V, Releases 3 and 4, and Solaris 2.0.

"This book is the perfect desktop reference.... The authors have presented a clear and concisely written book which would make an excellent addition to any UNIX user's library."
—*SysAdmin*

"Whether you are setting up your first UNIX system or adding your fiftieth user, these books can ease you through learning the fundamentals of the UNIX system."
—Michael J. O'Brien, Hardware Editor,
ABA/Unix/group Newsletter

SCO UNIX in a Nutshell

By Ellie Cutler & the staff of O'Reilly & Associates
1st Edition February 1994
590 pages, ISBN 1-56592-037-6



The desktop reference to SCO UNIX and Open Desktop®, this version of *UNIX in a Nutshell* shows you what's under the hood of your SCO system. It isn't a scaled-down quick reference of common commands, but a complete reference containing all user, programming, administration, and networking commands.

Contents include:

- All commands and options
- Shell syntax for the Bourne, Korn, C, and SCO shells
- Pattern matching, with *vi, ex, sed,* and *awk* commands
- Compiler and debugging commands for software development
- Networking with email, TCP/IP, NFS, and UUCP
- System administration commands and the SCO sysadmsh shell

This edition of *UNIX in a Nutshell* is the most comprehensive SCO quick reference on the market, a must for any SCO user. You'll want to keep *SCO UNIX in a Nutshell* close by as you use your computer: it'll become a handy, indispensible reference for working with your SCO system.

FOR INFORMATION: **800-998-9938**, 707-829-0515; **INFO@ORA.COM**; **HTTP://WWW.ORA.COM/**

Learning the UNIX Operating System

By Grace Todino, John Strang & Jerry Peek
3rd Edition August 1993
108 pages, ISBN 1-56592-060-0



If you are new to UNIX, this concise introduction will tell you just what you need to get started and no more. Why wade through a 600-page book when you can begin working productively in a matter of minutes? It's an ideal primer for Mac and PC users of the Internet who need to know a little bit about UNIX on the systems they visit.

This book is the most effective introduction to UNIX in print. The third edition has been updated and expanded to provide increased coverage of window systems and networking. It's a handy book for someone just starting with UNIX, as well as someone who encounters a UNIX system as a "visitor" via remote login over the Internet.

"Once you've established a connection with the network, there's often a secondary obstacle to surmount.... *Learning the UNIX Operating System* helps you figure out what to do next by presenting in a nutshell the basics of how to deal with the 'U-word.' Obviously a 108-page book isn't going to make you an instant UNIX guru, but it does an excellent job of introducing basic operations in a concise nontechnical way, including how to navigate through the file system, send and receive E-mail and—most importantly—get to the online help...."
—Michael L. Porter, Associate Editor, *Personal Engineering & Instrumentation News*

Learning the vi Editor

By Linda Lamb
5th Edition October 1990
192 pages, ISBN 0-937175-67-6

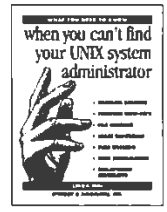


A complete guide to text editing with *vi*, the editor available on nearly every UNIX system. Early chapters cover the basics; later chapters explain more advanced editing tools, such as *ex* commands and global search and replacement.

"For those who are looking for an introductory book to give to new staff members who have no acquaintance with either screen editing or with UNIX screen editing, this is it: a book on *vi* that is neither designed for the UNIX in-crowd, nor so imbecilic that one is ashamed to use it."
—*;login*

When You Can't Find Your UNIX System Administrator

By Linda Mui
1st Edition April 1995
156 pages, ISBN 1-56592-104-6



This book is written for UNIX users, who are often cast adrift in a confusing environment. It provides the background and practical solutions you need to solve problems you're likely to encounter—problems with logging in, printing, sharing files, running programs, managing space resources, etc. It also describes the kind of info to gather when you're asking for a diagnosis from a busy sys admin. And, it gives you a list of site-specific information that you should know, as well as a place to write it down.

When You Can't Find Your UNIX System Administrator, part of our new What You Need to Know series, gives you tools for solving problems. The goal of this book is not to make you a guru, but to get you back to the job you'd rather be doing.

Learning the Korn Shell

By Bill Rosenblatt
1st Edition June 1993
363 pages, ISBN 1-56592-054-6



A thorough introduction to the Korn shell, both as a user interface and as a programming language. This book provides a clear explanation of the Korn shell's features, including *ksh* string operations, co-processes, signals and signal handling, and command-line interpretation. *Learning the Korn Shell* also includes real-life programming examples and a Korn shell debugger (*kshdb*).

"Readers still bending back the pages of Korn-shell manuals will find relief in...*Learning the Korn Shell*...a gentle introduction to the shell. Rather than focusing on syntax issues, the book quickly takes on the task of solving day-to-day problems with Korn-shell scripts. Application scripts are also shown and explained in detail. In fact, the book even presents a script debugger written for *ksh*. This is a good book for improving your knowledge of the shell."
—*Unix Review*

MH & xmh: Email for Users & Programmers

By Jerry Peek
3rd Edition April 1995
782 pages, ISBN 1-56592-093-7



There are lots of mail programs in use these days, but MH is one of the most durable and flexible. Best of all, it's available on almost all UNIX systems. It has spawned a number of interfaces that many users prefer. This book covers three popular interfaces: *xmh* (for the X environment), *exmh* (written with tcl/tk), and *mh-e* (for GNU Emacs users).

The book contains a quick tour through MH, *xmh*, *exmh*, and *mh-e* for new users; configuration and customization information; lots of tips and techniques for programmers—and plenty of practical examples for everyone; information beyond the manual pages, explaining how to make MH do things you never thought an email program could do; and quick-reference pages.

In addition, the third edition describes the Multipurpose Internet Mail Extensions (MIME) and how to use it with these mail programs. MIME is an extension that allows users to send graphics, sound, and other multimedia formats through mail between otherwise incompatible systems.

"The MH bible is irrefutably Jerry Peek's *MH & xmh: Email for Users & Programmers*. This book covers just about everything that is known about MH and *xmh* (the X Windows front end to MH), presented in a clear and easy-to-read format. I strongly recommend that anybody serious about MH get a copy."—James Hamilton, *UnixWorld*

Learning GNU Emacs

By Debra Cameron & Bill Rosenblatt
1st Edition October 1991
442 pages, ISBN 0-937175-84-6

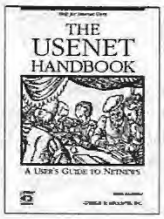


An introduction to the GNU Emacs editor, one of the most widely used and powerful editors available under UNIX. Provides a solid introduction to basic editing, a look at several important editing modes (special Emacs features for editing specific types of documents), and a brief introduction to customization and Emacs LISP programming. The book is aimed at new Emacs users, whether or not they are programmers.

"Authors Debra Cameron and Bill Rosenblatt do a particularly admirable job presenting the extensive functionality of GNU Emacs in well-organized, easily digested chapters.... Despite its title, *Learning GNU Emacs* could easily serve as a reference for the experienced Emacs user."
—Linda Branagan, Convex Computer Corporation

The USENET Handbook

By Mark Harrison
1st Edition May 1995
388 pages, ISBN 1-56592-101-1

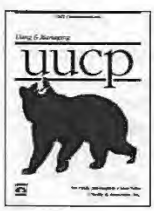


USENET, also called Netnews, is the world's largest discussion forum, encompassing the worldwide Internet and many other sites that aren't formally connected to any network. USENET provides a forum for asking and answering technical questions, arguing politics, religion, and society, or discussing most scientific, artistic, or humanistic disciplines. It's also a forum for distributing free software, as well as digitized pictures and sound.

This book unlocks USENET for you. It's not just a technical book, although it includes tutorials on the most popular newsreaders for UNIX and Windows (*tin*, *nn*, GNUS, and Trumpet). It also explains what goes on on the Net: where to look for information and what to do with it once you get it. And, it gives you an introduction into the culture: Net etiquette, the private language, and some of the history...including some of the more notable practical jokes.

Using and Managing UUCP

By Tim O'Reilly, Dale Dougherty, Grace Todino & Ed Ravin
1st Edition March 1996 (est.)
350 pages (est.), ISBN 1-56592-153-4



Using and Managing UUCP describes, in one volume, this popular communications and file transfer program. UUCP is regaining its popularity among computer users because it works efficiently on the PC-sized computers most people use today. UUCP is very attractive to computer users with limited resources, a small machine, and a dial-up connection.

Linux users admire the efficiency of UUCP. In fact, Taylor UUCP, which is described in this book, ships with the major Linux distributions. In addition to Taylor UUCP, this book covers the latest versions of HoneyDanBer UUCP, sometimes called Basic Network Utilities, or BNU. It also describes the specific implementation details of UUCP versions shipped by major UNIX vendors.

This book combines material about UUCP formerly contained in two other O'Reilly & Associates books, *Using UUCP and Usenet* and *Managing UUCP and Usenet*.

Running Linux

By Matt Welsh & Lar Kaufman
1st Edition February 1995
600 pages, ISBN 1-56592-100-3



Linux is the most exciting development today in the UNIX world—and some would say in the world of the PC-compatible. A complete, UNIX-compatible operating system developed by volunteers on the Internet, Linux is distributed freely in electronic form and for low cost from many vendors. Its software packages include the X Window System (X11R6); TCP/IP networking (including SLIP, PPP, and NFS support); popular software tools such as Emacs and TeX; a complete software development environment including C, C++, Perl, Tcl/Tk, and more; libraries, debuggers, multimedia support, scientific and database applications; and much more.

Running Linux covers topics not addressed in any Linux documentation, including everything you need to understand, install, and use the Linux operating system. This includes a comprehensive installation tutorial, complete information on system maintenance, tools for document development and programming, and guidelines for network administration.

X User Tools

By Linda Mui & Valerie Quercia
1st Edition November 1994
856 pages (CD-ROM included)
ISBN 1-56592-019-8



X User Tools provides for X users what *UNIX Power Tools* provides for UNIX users: hundreds of tips, tricks, scripts, techniques, and programs—plus a CD-ROM—to make the X Window System more enjoyable, more powerful, and easier to use. This browser's book emphasizes useful programs culled from the network, offers tips for configuring individual and systemwide environments, and includes a CD-ROM of source files for all—and binary files for some—of the programs.

Volume 3: X Window System User's Guide

Standard Edition
By Valerie Quercia & Tim O'Reilly
4th Edition May 1993
836 pages, ISBN 1-56592-014-7



The X Window System User's Guide orients the new user to window system concepts and provides detailed tutorials for many client programs, including the *xterm* terminal emulator and window managers. Building on this basic knowledge, later chapters explain how to customize the X environment and provide sample configurations. The *Standard Edition* uses the *twm* manager in most examples and illustrations. Revised for X11 Release 5. This popular manual is available in two editions, one for users of the MIT software, and one for users of Motif (see below).

"For the novice, this is the best introduction to X available. It will also be a convenient reference for experienced users and X applications developers."
—Computing Reviews

Volume 3M: X Window System User's Guide

Motif Edition
By Valerie Quercia & Tim O'Reilly
2nd Edition January 1993
956 pages, ISBN 1-56592-015-5

This alternative edition of the *User's Guide* highlights the Motif window manager for users of the Motif graphical user interface. Revised for Motif 1.2 and X11 Release 5.

Material covered in this second edition includes:

- Overview of the X Color Management System (Xcms)
- Creating your own Xcms color database
- Tutorials for two "color editors": *xcoloredit* and *xtici*
- Using the X font server
- Tutorial for *editres*, a resource editor
- Extensive coverage of the new implementations of *bitmap* and *xmag*
- Overview of internationalization features
- Features common to Motif 1.2 applications: tear-off menus and drag-and-drop

Advanced

UNIX Power Tools

By Jerry Peek, Mike Loukides, Tim O'Reilly, et al.
1st Edition March 1993
1162 pages (includes CD-ROM)
Random House ISBN 0-679-79073-X



Ideal for UNIX users who hunger for technical—yet accessible—information, *UNIX Power Tools* consists of tips, tricks, concepts, and freeware (CD-ROM included). It also covers add-on utilities and how to take advantage of clever features in the most popular UNIX utilities.

This is a browser's book...like a magazine that you don't read from start to finish, but leaf through repeatedly until you realize that you've read it all. You'll find articles abstracted from O'Reilly Nutshell Handbooks®, new information that highlights program "tricks" and "gotchas," tips posted to the Net over the years, and other accumulated wisdom. The goal of *UNIX Power Tools* is to help you think creatively about UNIX and get you to the point where you can analyze your own problems. Your own solutions won't be far behind.

The CD-ROM includes all of the scripts and aliases from the book, plus *perl*, GNU *emacs*, *pbmplus* (manipulation utilities), *ispell*, *screen*, the *sc* spreadsheet, and about 60 other freeware programs. In addition to the source code, all the software is precompiled for Sun3, Sun4, DECstation, IBM RS/6000, HP 9000 (700 series), SCO Xenix, and SCO UNIX. (SCO UNIX binaries will likely also run on other Intel UNIX platforms, including Univel's new UNIXware.)

"This substantial volume (1,100-plus pages) is about the power use of existing UNIX tools, with a CD distribution of others. It goes into the kind of wonderful detail that most administrators will relish. Take *find* for example. Most people use it only to find a file by name or age, but this book shows you how to do things such as finding all the root-owned executables with the set-user ID bit on that have been deposited in the last four months. (Naturally, this would be very handy for catching potential security holes.) You'll learn a lot from this book. I recommend it highly."—Bruce Hunter, *Open Systems Today*

Making TEX Work

By Norman Walsh
1st Edition April 1994
522 pages, ISBN 1-56592-051-1



TeX is a powerful tool for creating professional-quality typeset text and is unsurpassed at typesetting mathematical equations, scientific text, and multiple languages. Many books describe how you use TeX to construct sentences, paragraphs, and chapters. Until now, no book has described all the software that actually lets you build, run, and use TeX to best advantage on your platform. Because creating a TeX document requires the use of many tools, this lack of information is a serious problem for TeX users.

Making TEX Work guides you through the maze of tools available in the TeX system. Beyond the core TeX program there are myriad drivers, macro packages, previewers, printing programs, online documentation facilities, graphics programs, and much more. This book describes them all.

The Frame Handbook

By Linda Branagan & Mike Sierra
1st Edition November 1994
542 pages, ISBN 1-56592-009-0



A thorough, single-volume guide to using the UNIX version of FrameMaker 4.0, a sophisticated document production system. This book is for everyone who creates technical manuals and reports, from technical writers and editors who will become power users to administrative assistants and engineers. The book contains a thorough introduction to Frame and covers creating document templates, assembling books, and Frame tips and tricks. It begins by discussing the basic features of any text-formatting system: how it handles text and text-based tools (like spell-checking). It quickly gets into areas that benefit from a sophisticated tool like Frame: cross-references and footnotes; styles, master pages, and templates; tables and graphics; tables of contents and indexes; and, for those interested in online access, hypertext. Once you've finished this book, you'll be able to use Frame to create and produce a book or even a series of books.

Exploring Expect

By Don Libes
1st Edition December 1994
602 pages, ISBN 1-56592-090-2



Written by the author of Expect, this is the first book to explain how this new part of the UNIX toolbox can be used to automate Telnet, FTP, passwd, rlogin, and hundreds of other interactive applications. Based on Tcl (Tool Command Language), Expect lets you automate interactive applications that have previously been extremely difficult to handle with any scripting language.

The book briefly describes Tcl and how Expect relates to it. It then describes the Expect language, using a combination of reference material and specific, useful examples of its features. It shows how to use Expect in background, in multiple processes, and with standard languages and tools like C, C++, and Tk, the X-based extension to Tcl. The strength in the book is in its scripts, conveniently listed in a separate index.

"Expect was the first widely used Tcl application, and it is still one of the most popular. This is a must-know tool for system administrators and many others."
—John Ousterhout, John.Ousterhout@Eng.Sun.COM

sed & awk

By Dale Dougherty
1st Edition November 1990
414 pages, ISBN 0-937175-59-5



For people who create and modify text files, *sed* and *awk* are power tools for editing. Most of the things that you can do with these programs can be done interactively with a text editor; however, using *sed* and *awk* can save many hours of repetitive work in achieving the same result.

"*sed & awk* is a must for UNIX system programmers and administrators, and even general UNIX readers will benefit. I have over a hundred UNIX and C books in my personal library at home, but only a dozen are duplicated on the shelf where I work. This one just became number twelve."
—*Root Journal*

Learning Perl

By Randal L. Schwartz, Foreword by Larry Wall
1st Edition November 1993
274 pages, ISBN 1-56592-042-2



Learning Perl is ideal for system administrators, programmers, and anyone else wanting a down-to-earth introduction to this useful language. Written by a Perl trainer, its aim is to make a competent, hands-on Perl programmer out of the reader as quickly as possible. The book takes a tutorial approach and includes hundreds of short code examples, along with some lengthy ones. The relatively inexperienced programmer will find *Learning Perl* easily accessible. Each chapter of the book includes practical programming exercises. Solutions are presented for all exercises.

For a comprehensive and detailed guide to advanced programming with Perl, read O'Reilly's companion book, *Programming perl.*

"All-in-all, *Learning Perl* is a fine introductory text that can dramatically ease moving into the world of Perl. It fills a niche previously filled only by tutorials taught by a small number of Perl experts.... The UNIX community too often lacks the kind of tutorial that this book offers."
—Rob Kolstad, *;login*

Programming perl

By Larry Wall & Randal L. Schwartz
1st Edition January 1991
482 pages, ISBN 0-937175-64-1



This is the authoritative guide to the hottest new UNIX utility in years, coauthored by its creator, Larry Wall. Perl is a language for easily manipulating text, files, and processes. Perl provides a more concise and readable way to do many jobs that were formerly accomplished (with difficulty) by programming in the C language or one of the shells. *Programming perl* covers Perl syntax, functions, debugging, efficiency, the Perl library, and more, including real-world Perl programs dealing with such issues as system administration and text manipulation. Also includes a pull-out quick-reference card (designed and created by Johan Vromans).

TO ORDER: **800-889-8969** (CREDIT CARD ORDERS ONLY); **ORDER@ORA.COM**

At Your Fingertips—

A COMPLETE GUIDE TO O'REILLY'S ONLINE SERVICES

O'Reilly & Associates offers extensive product and customer service information online. We invite you to come and explore our little neck-of-the-woods.

For product information and insight into new technologies, visit the O'Reilly Online Center

Most comprehensive among our online offerings is the O'Reilly Online Center. You'll find detailed information on all O'Reilly products, including titles, prices, tables of contents, indexes, author bios, software contents, and reviews. You can also view images of all our products. In addition, watch for informative articles that provide perspective on the technologies we write about. Interviews, excerpts, and bibliographies are also included.

After browsing online, it's easy to order, too by sending email to **order@ora.com**. The O'Reilly Online Center shows you how. Here's how to visit us online:

☞*Via the World Wide Web*

If you are connected to the Internet, point your Web browser (e.g., `mosaic`, `netscape`, or `lynx`) to:

`http://www.ora.com/`

For the plaintext version, `telnet` to:
`www.ora.com` (login: `oraweb`)

☞*Via Gopher*

If you have a Gopher program, connect your `gopher` to:
`gopher.ora.com`
Or, point your Web browser to:
`gopher://gopher.ora.com/`
Or, you can `telnet` to: `gopher.ora.com`
(login: `gopher`)

A convenient way to stay informed: email mailing lists

An easy way to learn of the latest projects and products from O'Reilly & Associates is to subscribe to our mailing lists. We have email announcements and discussions on various topics. Subscribers receive email as soon as the information breaks.

☞*To join a mailing list:*

Send email to:
listproc@online.ora.com

Leave the message "subject" empty if possible.

If you know the name of the mailing list you want to subscribe to, put the following information on the first line of your message: `subscribe` "listname" "your name" `of` "your company."

For example: `subscribe ora-news`
`Kris Webber of Fine Enterprises`

If you don't know the name of the mailing list, listproc will send you a listing of all the mailing lists. Put this word on the first line of the body: `lists`

To find out more about a particular list, send a message with this word as the first line of the body: `info` "listname"

For more information and help, send this message: `help`

For specific help, email to: **listmaster@online.ora.com**

FOR INFORMATION: ***800-998-9938****, 707-829-0515;* ***INFO@ORA.COM****;* ***HTTP://WWW.ORA.COM/***

The complete O'Reilly catalog is now available via email

You can now receive a text-only version of our complete catalog via email. It contains detailed information about all our products, so it's mighty big: over 200 kbytes, or 200,000 characters.

To get the whole catalog in one message, send an empty email message to: **catalog@online.ora.com**

If your email system can't handle large messages, you can get the catalog split into smaller messages. Send email to: **catalog-split@online.ora.com**

To receive a print catalog, send your snail mail address to: **catalog@ora.com**

Check out Web Review, our new publication on the Web

Web Review is our new magazine that offers fresh insights into the Web. The editorial mission of Web Review is to answer the question: How and where do you BEST spend your time online? Each issue contains reviews that look at the most interesting and creative sites on the Web. Visit us at **http://gnn.com/wr/**

Web Review is a product of the recently formed Songline Studios, a venture between O'Reilly and America Online.

Get the files you want with FTP

We have an archive of example files from our books, the covers of our books, and much more available by anonymous FTP.

`ftp` to:

`ftp.ora.com` (login: `anonymous` – use your email address as the password.)

Or, if you have a WWW browser, point it to:

`ftp://ftp.ora.com/`

FTPMAIL

The ftpmail service connects to O'Reilly's FTP server and sends the results (the files you want) by email. This service is for people who can't use FTP—but who can use email.

For help and examples, send an email message to:

`ftpmail@online.ora.com`

(In the message body, put the single word: `help`)

Helpful information is just an email message away

Many customer services are provided via email. Here are a few of the most popular and useful:

info@ora.com
For general questions and information.

bookquestions@ora.com
For technical questions, or corrections, concerning book contents.

order@ora.com
To order books online and for ordering questions.

catalog@online.ora.com
To receive an online copy of our catalog.

catalog@ora.com
To receive a free copy of *ora.com*, our combination magazine and catalog. Please include your snail mail address.

international@ora.com
Comments or questions about international ordering or distribution.

xresource@ora.com
To order or inquire about *The X Resource* journal.

proposals@ora.com
To submit book proposals.

O'Reilly & Associates, Inc.

103A Morris Street, Sebastopol, CA 95472
Inquiries: **707-829-0515, 800-998-9938**
Credit card orders: **800-889-8969** (Weekdays 6 A.M.- 5 P.M. PST)
FAX: **707-829-0104**

O'Reilly & Associates—
LISTING OF TITLES

INTERNET

CGI Scripting on the World Wide Web (Winter '95-96 est.)
Connecting to the Internet: An O'Reilly Buyer's Guide
Getting Connected (Winter '95-96 est.)
HTML Handbook (Winter '95-96 est.)
Smileys
The USENET Handbook
The Whole Internet User's Guide & Catalog
The Whole Internet for Windows 95
Web Design for Designers (Winter '95-96 est.)
The World Wide Web Journal (Winter '95-96 est.)

SOFTWARE

Internet In A Box,™ Version 2.0
WebSite™ 1.1

WHAT YOU NEED TO KNOW SERIES

Bandits on the Information Superhighway (Winter '95-96 est.)
Marketing on the Internet (Winter '95-96 est.)
When You Can't Find Your System Administrator
Using Email Effectively

HEALTH, CAREER & BUSINESS

Building a Successful Software Business
The Computer User's Survival Guide
Dictionary of Computer Terms (Winter '95-96 est.)
The Future Does Not Compute
Love Your Job!
TWI Day Calendar - 1996

USING UNIX

BASICS

Learning GNU Emacs
Learning the bash Shell
Learning the Korn Shell
Learning the UNIX Operating System
Learning the vi Editor
MH & xmh: Email for Users & Programmers
PGP: Pretty Good Privacy
SCO UNIX in a Nutshell
UNIX in a Nutshell: System V Edition
Using and Managing UUCP (Spring '96 est.)
Using csh and tcsh

ADVANCED

Exploring Expect
The Frame Handbook
Learning Perl
Making TeX Work
Programming perl
Running Linux
Running Linux Companion CD-ROM (Winter '95-96 est.)
sed & awk
UNIX Power Tools (with CD-ROM)

SYSTEM ADMINISTRATION

Building Internet Firewalls
Computer Crime: A Crimefighter's Handbook
Computer Security Basics
DNS and BIND
Essential System Administration
Linux Network Administrator's Guide
Managing Internet Information Services
Managing NFS and NIS
Managing UUCP and Usenet
Networking Personal Computers with TCP/IP
Practical UNIX and Internet Security (Winter '95-96 est.)
sendmail
System Performance Tuning
TCP/IP Network Administration
termcap & terminfo
Volume 8: X Window System Administrator's Guide
The X Companion CD for R6

PROGRAMMING

Applying RCS and SCCS
C++: The Core Language
Checking C Programs with lint
DCE Security Programming
Distributing Applications Across DCE and Windows NT
Encyclopedia of Graphics File Formats
Guide to Writing DCE Applications
High Performance Computing
lex & yacc
Managing Projects with make
Microsoft RPC Programming Guide
Migrating to Fortran 90
Multi-Platform Code Management
ORACLE Performance Tuning
ORACLE PL/SQL Programming
Porting UNIX Software
POSIX Programmer's Guide
POSIX.4: Programming for the Real World
Power Programming with RPC
Practical C Programming
Practical C++ Programming
Programming with curses
Programming with GNU Software (Winter '95-96 est.)
Programming with Pthreads (Winter '95-96 est.)
Software Portability with imake
Understanding and Using COFF
Understanding DCE
Understanding Japanese Information Processing
UNIX Systems Programming for SVR4 (Winter '95-96 est.)

BERKELEY 4.4 SOFTWARE DISTRIBUTION

4.4BSD System Manager's Manual
4.4BSD User's Reference Manual
4.4BSD User's Supplementary Docs.
4.4BSD Programmer's Reference Man.
4.4BSD Programmer's Supp. Docs.
4.4BSD-Lite CD Companion
4.4BSD-Lite CD Companion: Int. Ver.

X WINDOW SYSTEM

Volume 0: X Protocol Reference Manual
Volume 1: Xlib Programming Manual
Volume 2: Xlib Reference Manual
Volume 3: X Window System User's Guide
Volume 3M: X Window System User's Guide, Motif Ed.
Volume 4: X Toolkit Intrinsics Programming Manual
Volume 4M: X Toolkit Intrinsics Programming Manual, Motif Ed.
Volume 5: X Toolkit Intrinsics Reference Manual
Volume 6A: Motif Programming Man.
Volume 6B: Motif Reference Manual
Volume 6C: Motif Tools
Volume 8: X Window System Administrator's Guide
Volume 9: X Window Window Programming Extentions (Winter '95-96 est.)
Programmer's Supplement for Release 6
The X Companion CD for R6
X User Tools (with CD-ROM)
The X Window System in a Nutshell

THE X RESOURCE

A QUARTERLY WORKING JOURNAL FOR X PROGRAMMERS

The X Resource: Issues 0 through 16

TRAVEL

Travelers' Tales France
Travelers' Tales Hong Kong (1/96 est.)
Travelers' Tales India
Travelers' Tales Mexico
Travelers' Tales Spain
Travelers' Tales Thailand
Travelers' Tales: A Woman's World

O'Reilly & Associates—
INTERNATIONAL DISTRIBUTORS

Customers outside North America can now order O'Reilly & Associates books through the following distributors. They offer our international customers faster order processing, more bookstores, increased representation at tradeshows worldwide, and the high-quality, responsive service our customers have come to expect.

EUROPE, MIDDLE EAST, AND AFRICA
(except Germany, Switzerland, and Austria)

INQUIRIES

International Thomson Publishing Europe
Berkshire House
168-173 High Holborn
London WC1V 7AA, United Kingdom
Telephone: 44-71-497-1422
Fax: 44-71-497-1426
Email: itpint@itps.co.uk

ORDERS

International Thomson Publishing Services, Ltd.
Cheriton House, North Way
Andover, Hampshire SP10 5BE, United Kingdom
Telephone: 44-264-342-832 (UK orders)
Telephone: 44-264-342-806 (outside UK)
Fax: 44-264-364418 (UK orders)
Fax: 44-264-342761 (outside UK)

GERMANY, SWITZERLAND, AND AUSTRIA

International Thomson Publishing GmbH
O'Reilly-International Thomson Verlag
Königswinterer Straße 418
53227 Bonn, Germany
Telephone: 49-228-97024 0
Fax: 49-228-441342
Email: anfragen@ora.de

ASIA *(except Japan)*

INQUIRIES

International Thomson Publishing Asia
221 Henderson Road
#08-03 Henderson Industrial Park
Singapore 0315
Telephone: 65-272-6496
Fax: 65-272-6498

ORDERS

Telephone: 65-268-7867
Fax: 65-268-6727

JAPAN

O'Reilly & Associates, Inc.
103A Morris Street
Sebastopol, CA 95472 U.S.A.
Telephone: 707-829-0515
Telephone: 800-998-9938 (U.S. & Canada)
Fax: 707-829-0104
Email: order@ora.com

AUSTRALIA

WoodsLane Pty. Ltd.
7/5 Vuko Place, Warriewood NSW 2102
P.O. Box 935, Mona Vale NSW 2103
Australia
Telephone: 02-970-5111
Fax: 02-970-5002
Email: woods@tmx.mhs.oz.au

NEW ZEALAND

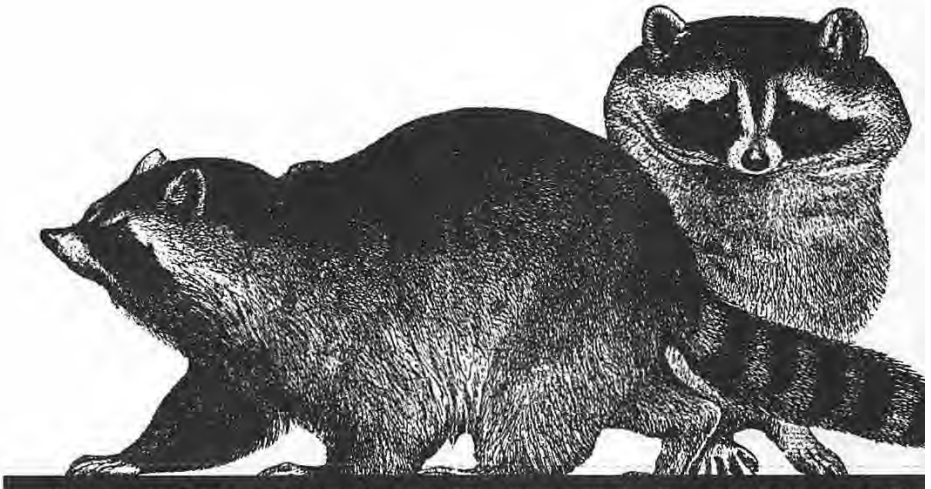
WoodsLane New Zealand Ltd.
21 Cooks Street (P.O. Box 575)
Wanganui, New Zealand
Telephone: 64-6-347-6543
Fax: 64-6-345-4840
Email: woods@tmx.mhs.oz.au

THE AMERICAS

O'Reilly & Associates, Inc.
103A Morris Street
Sebastopol, CA 95472 U.S.A.
Telephone: 707-829-0515
Telephone: 800-998-9938 (U.S. & Canada)
Fax: 707-829-0104
Email: order@ora.com

TO ORDER: ***800-889-8969*** *(CREDIT CARD ORDERS ONLY);* ***ORDER@ORA.COM***

Here's a page we encourage readers to tear out...



O'REILLY WOULD LIKE TO HEAR FROM YOU

Please send me the following:

❑ *ora.com*

O'Reilly's magazine/catalog, containing behind-the-scenes articles and interviews on the technology we write about, and a complete listing of O'Reilly books and products.

Which book did this card come from?

_____

Where did you buy this book?

❑ Bookstore ❑ Direct from O'Reilly

❑ Bundled with hardware/software ❑ Class/seminar

Your job description: ❑ SysAdmin ❑ Programmer

❑ Other _____

Describe your operating system: _____

_____

Please print legibly

Name Company/Organization Name

Address

City State Zip/Postal Code Country

Telephone Internet or other email address (specify network)

Nineteenth century wood engraving
of raccoons from the O'Reilly
& Associates Nutshell Handbook®
Applying RCS and SCCS.

POST CARD

PLACE
STAMP
HERE

O'Reilly & Associates, Inc., 103A Morris Street, Sebastopol, CA 95472-9902

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

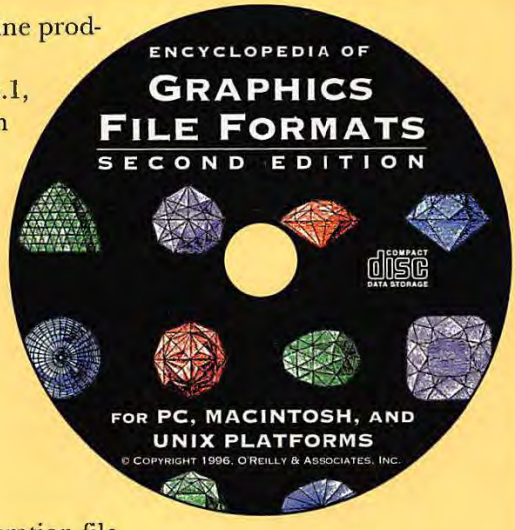
BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 80 SEBASTOPOL, CA

Postage will be paid by addressee

O'Reilly & Associates, Inc.
103A Morris Street
Sebastopol, CA 95472-9902

INSTALLATION QUICK REFERENCE

The Graphics File Formats (GFF) online product runs on a variety of platforms—Windows 95, Windows NT, Windows 3.1, the Macintosh, and UNIX. Installation is easy; follow the instructions below, and for more detailed installation instructions, see Appendix C, *Installation and Setup.*



- **For Windows 95 and Windows NT:**
  Run D:\PC\WIN95\SETUP.EXE
  where D: is the drive letter of your CD-ROM drive.
- **For Windows 3.1:**
  Run D:\PC\WIN31\SETUP.EXE
- **For the Macintosh:**
  The GFF application and its configuration file are in the Mac folder on the CD-ROM. You can run them directly from there, or drag them hoth into any folder you wish on your Macintosh.
- **For UNIX:**
  To install GFF on a UNIX workstation, follow the instructions in the INSTALL file in the appropriate directory on the CD-ROM. Look in */unix/platform*, where *platform* is the name of your platform (or an abbreviation of it).

To get started:
Click on the title page, and you'll see the Main Menu. From there, you can begin to explore the book text, file format specs, images, and software provided with the product.



ENCYCLOPEDIA OF

GRAPHICS FILE FORMATS

SECOND EDITION

O'Reilly's new edition of the *Encyclopedia of Graphics File Formats* is a book, a CD-ROM, and a World Wide Web product, all in one. In addition to the print book, you'll be able to access file format specifications, images, software, and the entire book on the CD-ROM. And you'll be able to link automatically from the product to the O'Reilly GFF Web Center—and from there to the larger world of the Internet—for even more complete and up-to-date information.

The *Encyclopedia of Graphics File Formats* is the definitive work on file formats—more than 100 of them—from Adobe Illustrator to ZBR. It's for graphics programmers, illustrators, and everyone who needs to deal with the low-level technical details of graphics files. The book includes detailed technical information on file formats (including new ones like PNG and SPIFF), as well as chapters on graphics and file format basics; conversion from one format type to another; compression methods like RLE, LZW, CCITT, JPEG, JBIG, ART, and fractal; and the use of graphics files on the Internet and the World Wide Web (how to post, download, and convert them).

The multi-platform CD-ROM included with this book contains:

- Updated vendor file format specifications (from vendors like Adobe, Apple, IBM, Microsoft, and Silicon Graphics)
- The best of the free graphics software and shareware—for Microsoft Windows, MS-DOS, OS/2, Macintosh, and UNIX platforms
- A variety of test graphics images
- The entire contents of the book, along with a World Wide Web browser

The new *Encyclopedia of Graphics File Formats* is a product that will never go out of date. If you have an Internet connection, you can go to the O'Reilly GFF Web Center to get updated versions of file format specs and software; see a roadmap to other resources of interest to graphics programmers; find out about online archives of graphics software and images; and learn what's new in the graphics world.

***"It seemed like fate. No sooner had I opened this resource when my beeper went off and I was asked about how to handle a file format conversion. The answer was in the* Encyclopedia."**

—Steve Jewett, Service Bureau Proprietor

ISBN 1-56592-161-5

US $79.95
CAN $113.95

90000

9 781565 921610

System Requirements to use CD-ROM:

- PC running Microsoft Windows (3.1, 95, or NT), Macintosh workstation, or UNIX platform supported by Spyglass Enhanced Mosaic
- Internal or external CD-ROM drive
- 256 color display recommended

Printed on Recycled Paper

ISBN 1-56592-161-5

