EXERCISE IMAGE FILE FORMATS FOR THE WEB

To complete this exercise, submit a jpg and a gif file format image. Make sure correct resolution before save for web. Make sure you choose the right format for the right image type.

Balance compression and quality

goal is the smallest possible file size w/out loosing too much quality

Png - portable network graphic format (Native file format for Fireworks

new format
compression based on deflation which is zip technology
lossless 24 bit color compression yields 16 million colors
.png-24 instead of **.jpg**
.png-8 instead of **.gif**
supports interlacing
256 levels of transparency (gif has 1)
saves gamma curve w/image - looks the same on both platforms
.png files bigger than jpg & gif
doesn't support animation
not supported universally by older browsers
expect it to gain popularity as it gains more browser support

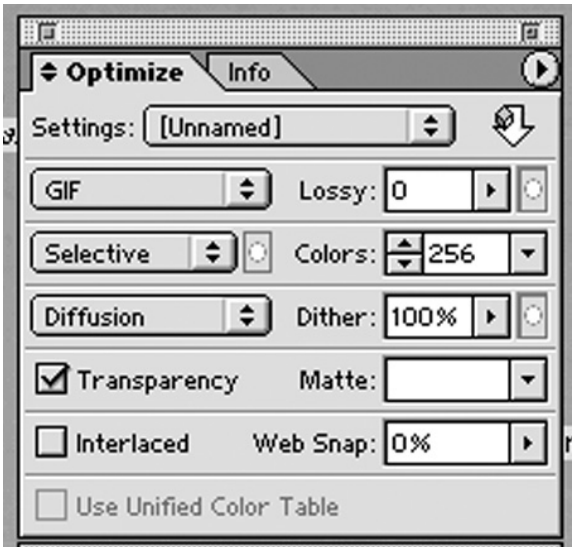
Jpg or Jpeg - (joint photographic experts group) PHOTOGRAPHS WITH MANY COLORS

equivalent to mpeg in movies but still image
single most important format for images w/more than 256 colors
saving splits brightness & color and compresses each individually
doesn't do well with high contrast areas
doesn't do well for grayscale images
not good for text
helps compression to blur image slightly
no transparency
not lossless - every time you save image shifts colors and degrades image
keep original tif and resave
minimize loss if you save with the same settings
progressive -simular to interlacing gif
appears in browser while downloading 3-5 passes
high quality compression 10:1-20:1 minimal loss of quality
medium quality compression 30:1-50:1 some visible color shifts
low quality 100:1 serious loss in quality
browsers view differently because of different decoders

Gif - (Graphical Interchange Format) FLAT AREAS OF COLOR IN IMAGE

most flexible format
2 compression techniques
LZW - from developers Lempel, Ziv & Welch (also in tif format)
based on pattern recognition
looks for pixels in a row (saves 5 x red instead of red 5 times)
CLUT - Color Look Up Table
few photos use full spectrum of colors
CLUT eliminates the colors not used cutting it down to 256 = 2/3 size reduction
this is called indexing and also available under mode - more options in save for web
Adjust color depth of CLUT relative to LZW compression
fewer colors in CLUT = better LZW
too much quality loss requires dithering which hurts LZW

Options in optimization palette



Lossy - uses patterns found by LZW algorithm and stores in compression table use slider to specify how much

Dithering - takes 2 colors from Color Table & makes missing color
improves visual quality of image
- increases file size because they compress less efficiently
- reduce the # of colors until drop in color quality > then adjust dithering to improve display keeping eye on file size

noise - best for the job
diffusion - simular to noise in effect, set amount in the slider
pattern - **don't use** mixes adjacent color in a regular pattern that you can see

Interlaced - important to use if you have a lot of images on pg
rearranges pixel rows so image displays gradually
viewers think its quicker but it actually takes slightly longer
.gif uses 4 passe to display - slight increase in size
.jpg lets you choose 3-5 passes - slightly smaller

Transparency & Matte - lets background color of web page show through transparent pixels halos-matte color select background of web page (anti-aliasing)

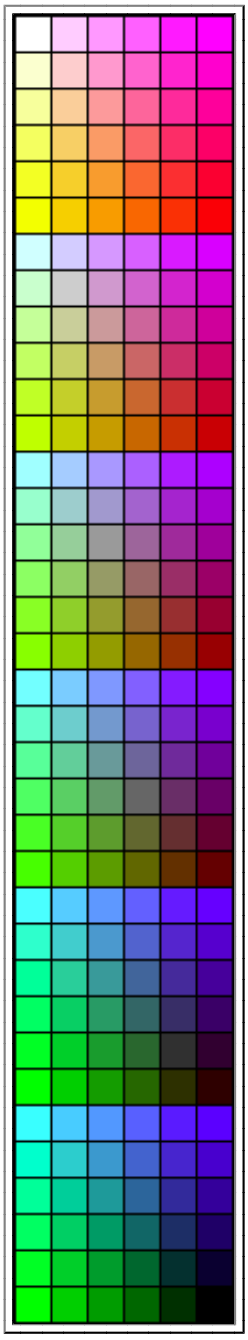
Can't use interlacing & transparency in same image unwanted pixels appear in transparent areas

Color Reducing Algorithms - decides which colors to include in CLUT
each image is unique & has different requirements
- adaptive - most often used with good results
  picks most frequently used colors in graphic not necessarily web safe colors so some dithering and color shifts
  web snap lets you shift colors to closets web safe
  ◆ marks web safe in color table
- selective & perceptual - simular
  selective is recommended by Adobe (default)
  favors broad areas of color and preservation of web colors
  perceptive - priority to color which human eye has greater sensitivity
- web - no dithering, no color changes, the same on both platforms except for gamma
  use only on graphics that have web safe colors
  worst mode for everything else/not used often

Change web snap slider to 25% and change between adaptive/selective/perceptive
VIEW>BROWSER DITHER VIEW>PREVIEW to see what it looks like in 256 colors
Other algorithms are exact, mac os/windows systems, previous, uniform
- Custom - make a common CLUT palette for all of your images
  use when many images with the same shades of color on the pg (colors don't shift)
  if lots of different images have their own adaptive color palette pc runs out of memory

HANDS -ON-

216 Colors



216 colors. That's all you're going to get from Netscape when your monitors color depth is set to 8 bits. And yuck, what ugly colors they are! I suppose there's a certain mathmatical consistency to the sequence of colors, but it seems that out of 16 million+ colors to choose from, they could have chosen a few more subtle tones. Really now, how many shades of flourescent green do we really need?

Well, my point was not to rag on Netscapes choice of colors, but to help those designers who've got to work with them. For the most part, I tend to rely on 24 bit JPEGs, or adaptive pallette GIFs. However, for some clients you must work to the specs of *their* machines. If your target audience is using Windows or Unix boxes with 256 color graphics cards, these are the only colors their Netscape browsers are capable of.

The purpose of this page is not to explain *why* these colors were chosen, but to help you use them in your own graphics. I'm approaching this mainly from the Photoshop users viewpoint, but their's some good information here for everybody.

The 216 Netscape colors come from a 6x6 color cube, equally spaced out. In RGB terms, it uses *only* the values of 0, 51, 102, 153, 204, or 255. In hex notation, this would be 00, 33, 66, 99, CC, or FF. For instance, when specifying a <BGCOLOR> attribute, "#FF0033" will *not* dither, while "#FF0032" will. To all intents and purposes they are the same color, so choose accordingly.

When working in Photoshop, there are two ways you can use this color pallette. First off, you need the CLUT file itself. Here it is, in BinHex format, and as a ZIP archive.

Here's how you use it. First, you can go down to your colors pallette, and select 'Swatches". Click on the arrow in the upper right, and select "Load Swatches". Navigate to the "216color.act" file, and you can now paint with the 216 colors of Netscape.

The other way you can use it is when dropping from RGB mode down to indexed color. When the options menu comes up, select "custom" for your pallette, and "diffusion" for your dithering. You'll have a color table displayed, and you'll notice another button for loading a pallette. Choose the "216color.act" file again, and now your image will be dithered using only those colors.

You may notice that your images are dithering down rather coarsely, especially if there are a lot of subtle gradations involved. This is unavoidable, unfortunately. The colors you have to choose from are geared more towards icons than photographic style images. On the plus side, they'll frequently be quite small, as Photoshop may only find 30-40 colors out of the table that it can actually use! However it turns out, you can rest assured that the image will appear the same to lower end machines, as well as the high-end workstations. You have to decide for yourself which market to gear your efforts towards. Good luck!

Your pal, -doc-

Here's something handy. I created this file for a friend who needed the 216 colors of Netscape in an Adobe Illustrator file. Here it is, in BinHex format, and as a ZIP archive.

- Today's date & time: Friday, February 18, 2005 03:47AM GMT -
- Page last updated: Wednesday, December 01, 2004 -

top

Application/Contract for Consultants/Contractors/Manufacturers Advertising

Please read and complete the entire form.

	Type of listing (See Descriptions Attached)	**Annual Fee**
☐	Bronze • Company name, address, telephone, fax and Web address • Hyperlinked Web address • Choice of one (1) consultant or manufacturer category	Free
☐	Silver Listing • Company name, address, telephone, fax and Web address • Hyperlinked Web address • Choice of four (4) consultant or manufacturer categories • 75 word promotional message	$600 USD/Year $850 CAN/Year
☐	Gold Listing • Company name, address, telephone, fax and Web address • Hyperlinked Web address • Choice of four (8) consultant or manufacturer categories • 75 word promotional message • Company logo at the top of the listing • Banner Ad • Up to 5 featured products listed	$1,000 USD/Year $1,400 CAN/Year

Please fill in the following:

Advertiser

Company Name:			
Address:			
City:		State/Province	
Zip/Postal Code:		Country:	
Telephone:		Toll-Free:	
Fax:			
Contact Person:			
Title:			
Email:			
Web site URL:			

75 word (or less) descriptive text (Silver and Gold listings only):

Category choices:

☐ Consultant/Contractor	☐ Manufacturer
☐ Network Cabling	☐ Network Cabling
☐ Networking Hardware	☐ Networking Hardware
☐ Computers	☐ Computers
☐ Instrumentation	☐ Instrumentation
☐ Process Control	☐ Process Control
☐ Software	☐ Software

Banner Ad file specifications:

- 468 pixels wide by 60 pixels high at 72 dpi
- 15 Kb (animated banner may be up to 30 Kb)
- GIF file format indexed to 216-color non-dithering CLUT
- Animated GIFs not to exceed 4 frames

Company or product information specifications:

- Microsoft Word ™ file format

Mail order form with cheque payable to:

MDW Engineering,
799 Des Fauvettes,
Longueuil, Québec
Canada,
J4G 2L7

APPLICATION FOR ADVERTISING CONTRACT ON WEBSITE "INDUSTRIAL COMPUTING.COM" BETWEEN:

_____

_____

_____

_____

_____

("the Advertiser")

and

MDW Engineering Reg'd,
799 Des Fauvettes,
Longueuil, Québec,
Canada, J4G 2L7
("the Agent")

() **Identity of Advertiser**: It is a condition of this contract that the Advertiser be a manufacturer, distributor or merchant offering to clients or customers the goods and services described in the advertisements contemplated herein.

() **Application for Contract**: The advertiser confirms having verified the current choice of listings and prices on the website of "industrial computing.com" ("the website"), from which the present form of application and contract may be downloaded, and hereby applies for the _____ listing according to the same terms and conditions, as more fully set forth below. Where applicable, the Advertiser also encloses a cheque to the order of the Agent for the appropriate amount, namely $_____, as specified on the website.

() **Principal Obligations of Agent**: The Agent shall post the applicable information supplied by the Advertiser on the website as provided herein, but shall have no liability for network failures, acts of God or a public enemy or hackers, fortuitous event, or any other cause beyond the Agent's reasonable control which might make the website temporarily unavailable to the public.

() **Initial Obligations of Advertiser**: The Advertiser shall supply the applicable advertising material in Microsoft Word TM file format and comply with current banner ad specifications, notably 468 pixels wide by 60 pixels high at 72 dpi, 15 Kb (though animated banner may be up to 30 Kb), GIF file format indexed to 216-colour non-dithering CLUT, animated GIFs not to exceed 4 frames, as more fully appears from Appendix "A" attached hereto as an integral part of this text.

() **Term of Contract**: Listings are for a period of twelve complete calendar months beginning on the first day of the month immediately following acceptance of this application by the Agent, as evidenced by posting of the desired information on the website and deposit of the cheque, if applicable, in a chartered bank. Contracts may be renewed according to the terms, conditions, and rates shown on the website during the eleventh month of the current term. Contract for free listings are tacitly renewed in this manner unless terminated by either party as provided herein. Contracts for paid listings are not tacitly renewed, but the Advertiser, if satisfied with the terms, conditions and prices currently posted on the website as aforesaid, may apply to the Agent for renewal in the same manner simply by posting a cheque for the appropriate amount during the eleventh month of the current term, together with a description of any desired changes in the listing. Contracts may also be terminated in writing by either party before expiration of the current term, such termination to take effect as of the end of the full calendar month immediately following the month of receipt by the other party of the written notice of termination. In such cases the Advertiser is entitled to a pro-rata refund for the unused months.

() **Changes in Listings**: The Advertiser may change a listing as of the beginning of the second full calendar month following a written notice to the Agent of the desired changes, together with the requisite cheque, if applicable, the whole in accordance with the terms, conditions and rates shown on the website during the calendar month in which the notice is sent.

() **Notices**: Unless otherwise agreed in writing, notices and enclosures intended by one party for the other shall be sent by ordinary mail at their mutual addresses specified above.

() **Appropriate and Inappropriate Advertisements**: The Advertiser warrants that no part of its advertisements on the website will infringe any patents, copyrights or other intellectual property rights, will constitute defamation, libel, slander, obscenity, indecency or any other matter in bad taste or violation of law. The Advertiser also warrants that it complies with all requirements for permits or otherwise in any and all jurisdictions having jurisdiction over contracts between the Advertiser and the Advertiser's clients or customers in respect to the provision of goods and services advertised in accordance with the present contract. The Advertiser undertakes to indemnify the Agent or hold the Agent harmless in respect to any claims for damages or otherwise arising out of the Advertiser's listing and material therein. If the Agent comes to believe, rightly or wrongly, that the aforesaid warranty has been breached, the Agent may immediately and in its sole discretion suspend the listing of any such inappropriate material without incurring liability toward the Advertiser, and shall promptly notify the Advertiser thereof in writing as provided herein. Such suspension may be lifted by mutual consent and at the sole risk of the Advertiser on such terms and conditions as may be agreed.

() **Indemnification by the Advertiser**: The Advertiser shall indemnify the Agent and hold the Agent harmless, together with the Agent's employees or other representatives, from any claims (with or without litigation), costs, damages, expenses or losses in respect to inappropriate, or allegedly inappropriate, material supplied (or wrongfully not supplied) for the website by the Advertiser or persons on the Advertiser's behalf, whether such persons were properly authorized or not. Without restricting the generality of the foregoing, such indemnification shall include claims for death or bodily injury, product liability, all matters listed in the clause concerning appropriate and inappropriate advertisements, and reasonable judicial and extra-judicial fees of the Agent's attorneys.

() **Indemnification by the Agent**: The Agent's liability for any and all claims of the Advertiser arising out of this contract shall not under any circumstances exceed the amount paid to the Agent by the Advertiser for the current term. Acceptance of an application from the Advertiser does not imply endorsement by the Agent of the Advertiser's products or services, and rejection thereof does not imply lack of merit in such products or services, nor does it give rise to any liability on the Agent's part. Acceptance and rejection are determined in all cases by the Agent in the Agent's sole discretion.

() **Assignments and Other Transfers**: The parties may assign or otherwise transfer rights and obligations under the contract, but no such transfer by one party shall be effective as regards the other party without the written consent of the latter and the written acceptance by the assignee or other transferee of all obligations assigned or transferred, whether known or unknown. Except as expressly provided herein, no obligations of either party shall create any personal liability on the part of shareholders, directors, officers, owners, employees, agents or other representatives of such party. Such persons shall benefit from all the immunities and rights of indemnification of the parties with which they are associated.

() **Jurisdiction**: Even though the present contract may be entered into on the world wide net outside national boundaries, it is agreed that it shall be governed by the laws and courts of the jurisdiction in which the Agent's head office is located.

Signature: _____

Title: _____

Printed Name: _____

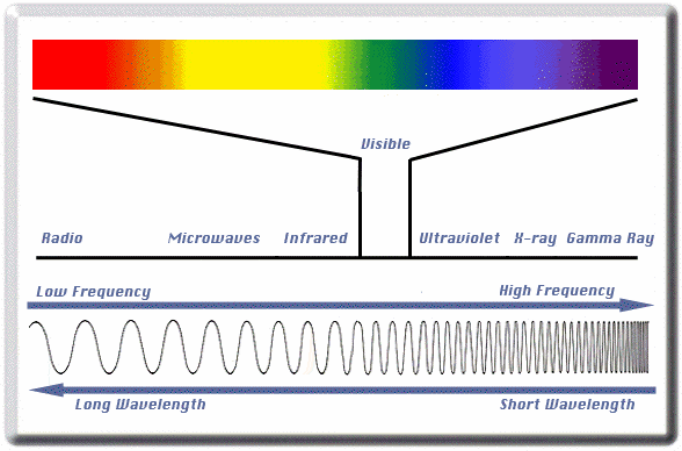
Date: _____

updated: 04/2003

CGT 211
Sampling and File Formats

The Physics of What We Do

2 types of waves - electromagnetic and pressure



Analog

- frequency variations, infinite
- defines color, brightness, pitch, volume

Digital Data

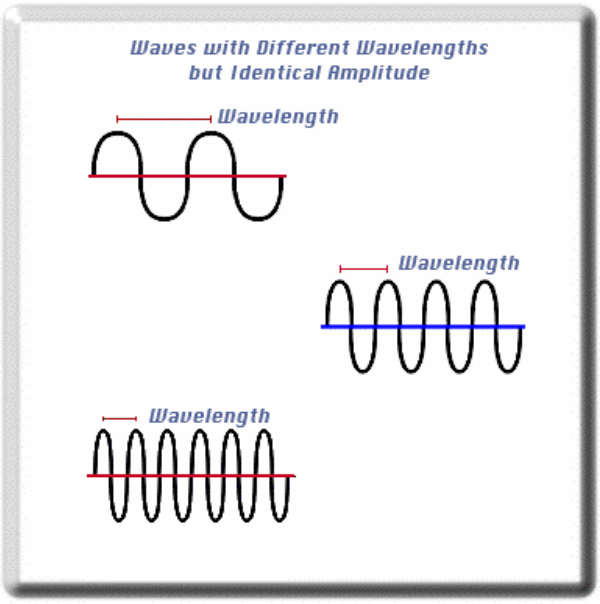
- Binary on and off states, discrete data
- Electronics are on or off

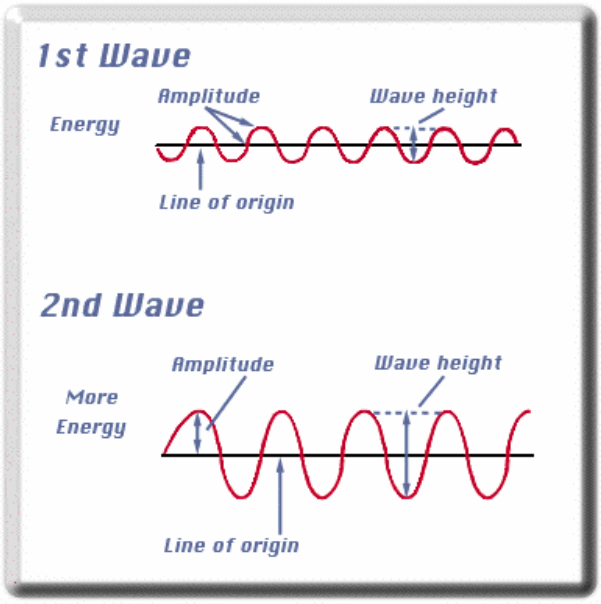
Cycle

- the repetitions of a wave
- determines the pitch or hue

Amplitude

- intensity of a wave
- is it positive (above baseline) or negative (below baseline)
- determines the brightness or volume (decibel)





The Sampling Process

What is Sampling?

1. Digitizing an image using a scanner
2. Digitizing a sound using a microphone or stereo
3. Digitizing a video clip using a capture/encoder

What Happens During Sampling?

- Portions of analog data are captured over time and converted to digital representations
- **ADC** - analog to digital conversion
- The rate of sampling is important b/c it affects the resolution.

Sampling Terminology

Sample - smallest unit
- an instance in time (audio) or unit from an image

Sampling Rate - the **frequency** of samples or how often the samples are taken
- affects resolution (sampled quality)
- controls the dpi/size of an image being captured
- in the case of sound controls the decibel level captured

Material frequency - the natural frequency
- the hue or pitch being captured

Bit depth - the number of bits to describe each sample

Nyquist Theorum - must sample at 2x the highest frequency in the material to capture

that frequency
- wave requires two values

So, the optimum sampling rate is 2x the highest frequency…

Aliasing occurs when sampling rate interferes or hinders the natural frequency of material.

Side bands - artifact frequencies as a result of aliasing

2 Solutions to Aliasing

Oversampling - sampling at greater than 2x the material

Filtering - massaging data with an algorithm to approximate data through interpolation or extrapolation

- To maximize a device's output quality, image resolution (size/dpi ratio) and bit depth must equal output device resolution and bit depth
- When sampled is less than 2x the highest frequency anomalies occur: called aliasing
- Aliasing occurs when the sampling device or output device interferes or cannot fully represent the material its normal frequency

Computer File Formats

File Format Types

1. **Graphics**
   - Bitmap
   - Vector
   - Metafile
2. **Animation/ Video**
3. **Multimedia**
4. **Hypermedia**
5. **3D/ Scene**

6. Audio

7. Fonts
- Bitmap
- Stroke
- Outline

8. Page Description Language (PDL)

9. Other/ Hybrid

Examples:

Adobe Illustrator	Metafile
Adobe Photoshop	Bitmap
AutoCAD DXF	Vector/ 3D
CALS Raster	Bitmap
CGM:	Metafile
Encapsulated Postscript	Metafile
GIF	Bitmap
JPG	Bitmap
Kodak Photo CD	Bitmap
Macromedia Freehand	Metafile
Macintosh PICT	Metafile
Microsoft RTF	Metafile
Microsoft Windows Bitmap	Bitmap
Microsoft Windows Metafile	Metafile
PCX	Bitmap
PNG	Bitmap
TGA	Bitmap
TIFF	Bitmap

Raster Graphics Formats (Bitmaps)

Multimedia and the Web – JPEG, GIF, PNG
Print Based Applications – TIFF, EPS, or DCS

Bitmap File Structure

Header - lists descriptive information about the image
Palette - optional

Bitmap Data - scan line or planar definition of pixel color
Footer - closing information – allows for backward compatibility

GIF (Graphic Interchange Format)

- 1 to 8 bits - 256 colors
- Transparency (1-bit)
- LZW compression
- Compuserve/ Unisys Patent
- Animations
- Interlacing
- Best for flat colors, use B-safe clut
- GIF 87a – normal
- GIF 89a – transparency

JPEG (Joint Photographic Experts Group)

- JFIF is the actual format - JPEG File Interchange Format
- up to 24 bit
- lossy compression
- Progressive (interlaced)
- subsampling, quantization
- Best for photographic colors
- Color shifting frequently occurs

PNG (Portable Network Graphics)

- Pronounced “ping”
- Extension: .png
- 1-bit to 48-bit
- LZ77 variant
- lossless compression
- Interlaced
- Transparency
- Developed by the W3C
- aimed at replacing GIF

BMP

- also known as Microsoft Windows Bitmap
- Extension: .bmp
- 1-32 bits (CYMK and RGB)
- Optional RLE
- Used way more than it probably should

TIFF (Tagged Image File Format)

- Extension .tif, .tiff
- 1-24 bit

- RLE, LZW, CCITT, JPEG
- Used as Mac clipboard format

Adobe Illustrator
- Extension: .ai, .eps
- Colors: unlimited
- Metafile format

Adobe Photoshop
- Extension: .psd
- Colors: Unlimited
- Should save as this because it retains the most number of options
- RLE used automatically
- Nice because selection channels are shrunk tremendously
- Files with multiple layers are compressed as well because transparent space shrinks to almost nothing
- Photoshop automatically saves a flattened image with your layer for programs that will read .psd files but not layers (ex. Freehand)
- Can turn this option off in "Preferences" to make file sizes smaller by turning off the "Maximize backwards compatibility in Photoshop format" option

EPS (Encapsulated Postscript)
- Extension: .eps, epsf, epsi
- Subset of Postscript Page Description Language (PDL)
- Postscript created in 1985 by Adobe Systems
- Metafile/ PDL

Macintosh Pict Resource
- Extension: .pct
- Metafile
- Up to 24-bit
- Developed by Apple
- Used to be used as Mac clipboard format (replaced by TIFF)

Microsoft Windows Metafile
- Extension: .wmf
- 24-bit max
- Used for windows clipboard

CALS Raster (Computer Aided Acquisition and Logistics Support Office)
- United States Department of Defense
- Extension: .cals, .cal, .ras
- Colors: Mono
- CCITT compression
- Bitmap format

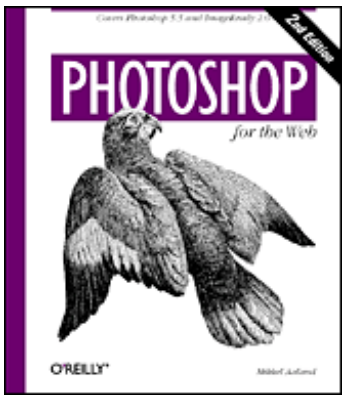
CGM (Computer Graphics Metafile)

- Extension: .cgm
- Colors: Unlimited
- RLE and CCITT Compression available

Kodak Photo CD

- 24-bit
- Proprietary compression
- Multiple resolution "stamps"
- Varies from 64x96 to 2048x3072
- 6 resolutions in one file

PCX
PDF (Portable Document Format)
TGA (Targa)
Scitex CT
Filmstrip
Amiga IFF
MacPaint
Pixar



Photoshop for the Web, 2nd Edition

By Mikkel Aaland
2nd Edition November 1999
1-56592-641-2, Order Number: 6412
246 pages, including multipage color insert, $29.95

Chapter 3
Making Great GIFs

What's more important--a site that loads fast or a site that looks great? If you simply convert your art to a GIF or JPEG file, accept the default settings, and save the file, chances are you'll wind up with graphics that are neither good enough nor small enough.

Through the next four chapters, you'll learn tons of tricks and techniques to make the most effective web graphics with Photoshop. In this chapter, we'll deal with converting existing images to GIF. Chapter 4, *Creating GIFs from Scratch*, shows you how to create original GIF files. Chapter 5, *Special Effects with Transparent GIFs*, covers transparent GIFs. And Chapter 6, *JPEG: All the Color You Want*, shows you how to make the best JPEG files.

GIF or JPEG?

The primary decision you have to make for any web graphic is file format. Make the wrong choice and you'll wind up with files that are larger than they need to be and images that don't look as good as they could. Make the right choice and you'll be assured that your images load fast, look reasonably good, and work for users of all platforms and screen displays.

Although there are many graphic file formats, GIF and JPEG are the most important ones for the Web, since they are directly supported by web browsers. All versions of Photoshop will read and write the basic GIF and JPEG formats. Photoshop 4 and 5 (and earlier versions, with the help of plug-ins) will read and write later variations of the formats (such as progressive rendering for JPEGs and transparency for GIFs). Photoshop 4, 5.0, 5.02, and 5.5 offer support for PNG, a relatively new file format that is gaining widespread support (and which is discussed in more detail in Appendix A, *The PNG Format*). Photoshop 5.5 offers improved PNG support.

Both the GIF and JPEG formats give you ways to control the file size of your web graphics--a critical feature considering the Web's current limited bandwidth capabilities. JPEG does this by compressing and actually throwing away data it considers unnecessary. GIF controls file size through a combination of compression and color reduction.

So, GIF or JPEG? The answer depends on the image's content and on how it will appear on your web page.

There are a number of crucial differences between the two formats:

- GIF is an 8-bit format. JPEG is a 24-bit format.
- GIF uses lossless compression (with an exception discussed later). JPEG uses lossy compression.
- GIF uses a color index. JPEG uses the full range of 16 million colors.
- GIF supports transparency. JPEG does not.

In general, GIF works better for images with flat colors and JPEG works better with photographic images, but there are many exceptions to the rule. Table 3-1 gives some general rules for when and when not to use GIF and JPEG.

Table 3-1: When to use and when to avoid GIF and JPEG

	Use for:	Avoid with:
GIF	Graphics containing limited numbers of colors, or intricate detail that you want to maintain	Large photographic images or illustrations that contain a lot of color, unless you want to create a floating effect
	Floating graphics from the page	
	Creating a simple animation (with the latest GIF file format)	
	Browser-safe colors--colors that won't dither or change when viewed on an 8-bit monitor	
JPEG	Most photographic images and continuous-tone art that contain subtle shifts in color	Text or graphics that contain detailed edges

Those are the rules of thumb. In the real world, things are a little less cut-and-dried. Take a look at Figures C-18 through C-23 in the color insert to see some actual situations and the decisions that web designers made.

Indexing your image

The most important thing to know about GIF files is that they're indexed images. That is, the file contains an index, or a palette, of all the colors in the image. Because GIF is an 8-bit format, the maximum number of colors in a GIF file is 256. This palette is used to assign a color to each pixel in the image. Actually, each pixel is assigned an index number, and RGB color values are associated with the index number. For instance, index number 22 may be assigned a navy blue color, and number 23 may be assigned sky blue. In the GIF file format, one index number can be tagged as transparent. We'll discuss GIF transparency more in Chapter 5.

Figures C-24 and C-25 make the point. Figure C-24 shows a simple illustration created by the artist James Yang and its color index. You can clearly see how the colors in the index are displayed in the image. In Figure C-25, we have the exact same image, but a different color index. The image changes accordingly.

When you realize how the index affects the image, you can see what havoc would be created if you applied the wrong index to an image.

With Photoshop 5.5, there are three ways to index an image in preparation for the GIF file format:

- File → Save for Web
- Image → Mode → Indexed Color
- File → Export → GIF89a Export module

The Save for Web feature is only available in 5.5, which also has a better Indexed Color feature than do Photoshop 4 and 5. The GIF89a Export module, which is primarily used to create GIFs with transparent areas, has remained the same since Photoshop 4.0.

Each method has its pros and cons:

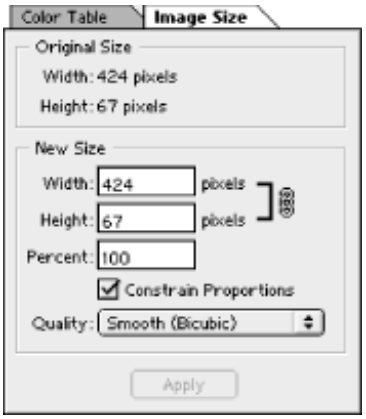
- The Save for Web plug-in is Photoshop's most useful web-friendly feature to date, but it is limited in the way it creates transparent GIFs.
- The Image → Mode → Indexed Color method is very flexible--you can do things such as applying selective dithering and tweaking the color palette through selections--however, it lacks the full interactive capabilities of the Save for Web plug-in.
- The GIF89a Export module, when used with the Image → Mode → Indexed Color method, is great for creating transparent areas, but its RGB-to-Indexed Color capabilities are rudimentary, especially when compared with the other methods.

Let's start by indexing an image with the Save to Web plug-in, then move on to using the Image → Mode → Indexed Color method. I'll describe in detail the GIF89a Export module in Chapter 5.

Using Save for Web

When you choose Save for Web, Photoshop 5.5 displays your image in a new window (see Figure C-26). Once you are in this window, you won't have access to any of Photoshop's editing or processing tools. You can, however, resize your image in the Save for Web box. Click on the Image Size tab in the lower-right corner to bring up the Image Size palette shown in Figure 3-1. Regardless of the original image's resolution, the optimized image will always be saved at 72 dpi.

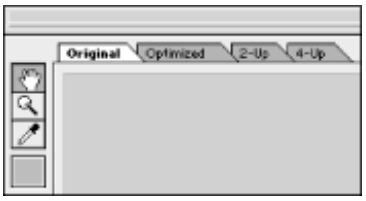
Figure 3-1.



As soon as your image appears in the Save for Web box, Photoshop automatically begins optimizing the image for the web. Photoshop uses your most recently selected optimization settings; if you've never run the plug-in before, the default settings will be used. You don't have to wait for the optimization process to complete in order to customize the settings. After the image loads, press Command-Period (Mac) or Escape (Windows) to stop the optimization process. Make sure you wait for the image to load in, though, as indicated in the progress bar at the bottom of the window; if you abort while the image is still loading, Save for Web will abort, and you'll have to start all over again

Next, click on the Original tab in the upper left of the interface, shown in Figure 3-2. This lets you change the optimized settings without waiting for Photoshop to apply each change. (If you select the adjacent 2-Up or 4-Up tab, you'll get a view of your original graphic displayed side by side with one or three optimized versions of the graphic. You can assign different optimizing settings to each version and compare the effect.)

Figure 3-2.



In order to illustrate what comes next, I'll show two ways to optimize the same image. Both ways will create a good-looking GIF that can be read by all browsers, but that's where the similarity ends. Only the second method will create a graphic that is as small as possible and contains key areas of browser-safe colors.

For my example, I've chosen a graphic created by Valerie Robbins for a *nationalgeographic.com* page. It contains type and a background with solid colors, and will not suffer from being reduced to 256 or less colors. In other words, it is a perfect candidate for GIF.

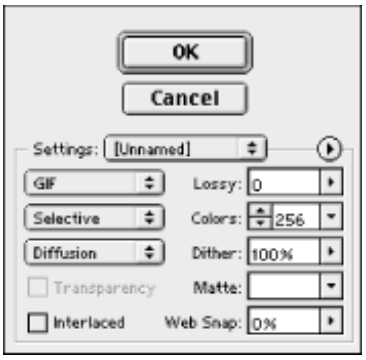
The easy way

The first way is the easiest: I simply apply Save for Web's default GIF settings. These settings give visually pleasing results for almost every graphic that meets the GIF criteria.

Here are the basic steps:

1. Run the Save for Web plug-in.
2. Wait for the optimizing process to stop, then select the Original tab.
3. If they aren't chosen already, choose the following settings from the drop-down menus in the upper-right corner, as shown in Figure 3-3.

Figure 3-3.



Format: GIF

Palette: Selective

Dither: Diffusion

Lossy: 0

Colors: 256

Dither: 100%

Web Snap: 0%

Interlaced box: unchecked

Figure 3-4.

1. Select Optimize from the menu tab.
2. Once the optimizing process is complete, toggle back and forth between the original and the optimized version using the Original and Optimized tabs. Zoom in for a closer look by selecting the zoom tool and clicking on the image. You can also zoom in by selecting the drop-down preview menu found by holding down Control/Shift while clicking anywhere in the Save for Web window.
3. If the optimized graphic looks acceptable, and it should, click OK and save the file. Photoshop automatically adds the *.gif* extension.

Following these steps, I successfully optimized the web graphic shown in Figure C-26, weighing in at 14.31K. The fact is, I can do better. (By the way, the file size circled in red in the lower-left side of the window is accurate, unlike file sizes found in Photoshop's main window, which are approximations.)

Batch optimization

To apply custom settings to similar graphics, choose Save Settings from the drop-down menu. See Figure 3-4. To embed your settings into the original Photoshop file, hold down the Option/Alt key. The OK button turns into a Remember button. If you select Remember, the next time you open the Photoshop file with the Save for Web plug-in, it will apply the same settings that were applied at the time you choose Remember. (If you don’t do this, the next time you open your Photoshop file, it will apply the most current optimized settings.)

The better way

Now, I'll try again, keeping the GIF file format but tweaking the other settings. I've set my window to display 4-Up and zoomed in 300%

percent so you can clearly see the effects of the different settings.

In Figure C-27, I've applied three different palettes, Perceptual, Adaptive, and Web, keeping my other settings the same as in the first example.

As you can see, the Perceptual palette created a slightly larger file size (14.38K) than the default Selective palette (14.31K). Otherwise, the result is very similar, which isn't surprising because both palettes create a custom color table by prioritizing colors for which the human eye has greater sensitivity. Selective is just slightly better than Perceptual at maintaining broad areas of color and preserving browser-safe colors.

The Adaptive palette (formerly the "best" option in Photoshop 4 and 5), which samples colors from the spectrum appearing most commonly in the image, doesn't do as good a job, at least not in this graphic. Notice the chunky look of the gold around the word "light." The Adaptive palette also produced a slightly bigger graphic (14.59K). Even though the Web palette resulted in a file size of only 11.86K, the graphic looks awful. That's because the Web palette only applies the 216 colors shared by both the Mac and PC platforms.

Based on what I see here, I will stay with the Selective palette. (In general, I find the Selective palette does the best job; however, I use the Adaptive palette when I index a color photograph, and I use the Web palette when a graphic contains a limited number of flat colors.)

In Figure C-28, I've applied three different dither percentages, 0%, 50%, and 100%, leaving the dither type set to Diffusion. The Diffusion option always produces a better result than Pattern. If you are going to slice your image into smaller parts, you might want to try 5.5's new Noise dither pattern, which reduces the seam patterns along sliced edges.

As you can see, as I decreased the amount of dither I've also decreased the file size. However, the difference is so small, I will leave my settings at 100%, which produces a slightly better image.

Figure C-29 shows the effect of Photoshop's Lossy GIF setting. Adobe has come up with a clever technique that actually turns the normally lossless GIF compression process into a lossy one. I'll spare you the technical details of how they do it, but the result is that you can apply greater compression to the GIF file format, albeit with a trade-off in quality. When I set a value of 15, there is little noticeable difference in quality but a nice drop in file size from 14.31K to 12.18K. At a lossy value of 100, the file size is 7.49K, but the quality is unacceptable.

Keep in mind that you cannot use the Lossy option when you've selected the interlace option or when Noise or Pattern dither algorithms are selected. (Interlace, by the way, creates a GIF that loads on a browser window gradually, in successive passes. Interlacing adds file size, so it is best avoided.)

Figure C-30 shows the effect of reducing the number of colors in the graphic. Reducing palette size always yields a substantial reduction in file size. The quality difference between 256 colors and 128 colors is slight and yet there is a drop in file size from 12.18K to 10.84K. Dropping to 8 or 32 colors was unacceptable.

Putting this all together, I now have a graphic with a file size of only 10.84K.

One last thing. How will this graphic look on monitors that can only display 256 colors (8-bit monitors)? Those systems dither any colors not found in the 216-color browser-safe palette. I am not so concerned about the detailed areas of the graphic--they'll still look OK if they dither. It's the solid blocks of color that worry me. If they dither, they will detract from the overall quality of the graphic.

How do you know if a color is browser-safe? How do you change it if it is not? It's easy with the Save for Web plug-in:

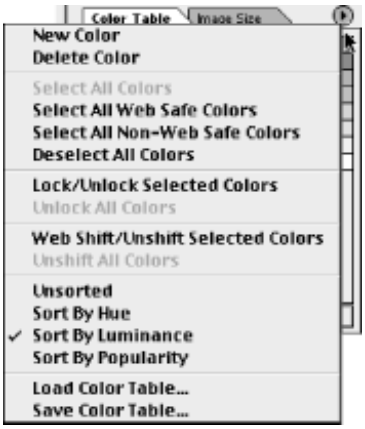
1. In the Save for Web box, select the eyedropper tool.
2. Click on a color in your graphic.
3. Turn your attention to the Color Table to the right of your graphic. (See Figure C-31.) The color you selected is represented and highlighted in the Color Table with a white border around it. If the selected colors box contains a black diamond in the center, it's browser-safe. If it doesn't, it's not.
4. To change an unsafe color to a browser-safe color, simply click on the cube icon at the bottom of the Color Table. This will shift the color to the nearest browser-safe color.
5. To prevent the color from being dropped or changed if you alter your color settings, click on the padlock icon at the bottom of the Color Table. A white square with a red center appears in the lower-right corner of each locked color.

In Val's graphic, the dominant red in the background is already browser-safe. But the yellows that make up the gold box are not, as shown in Figure C-31. Using the method outlined above, I'll shift several of the yellows to browser-safe colors to prevent them from dithering. By holding down the Shift key while I use the eyedropper tool, I can make multiple selections and use the cube to shift them all at once, as shown in Figure C-32. I could have also used Save for Web's Web Snap feature. By entering a higher numerical value, it automatically shifts more colors to browser-safe. I prefer, however, to shift colors individually.

Here are some other important things you should know about the Color Table:

- You can change the way the colors are sorted in the Color Table panel pop-up menu. See Figure 3-5.
- You can add colors from the Color Table by clicking on the color sample box. Double-clicking brings up the Color Picker. Choose a new color; when you close the Color Picker, the new color will replace the old one in the sample box. Remember, 5.5's Color Picker now has an Only Web Colors option.
- You can delete a color by clicking on it in the sample box and then clicking on the trash icon at the bottom of the Color Table panel. You can delete a number of colors at the same time by holding down the Shift key while selecting and then clicking on the trash icon.

Figure 3-5.



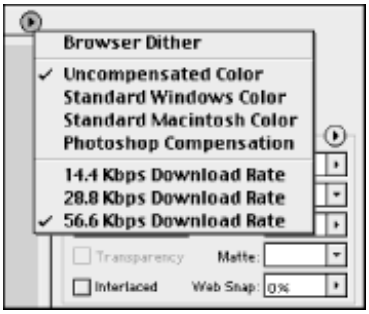
Save for Web or ImageReady?

Photoshop's Save for Web has many of the same optimizing controls as ImageReady. So when should you use Save For Web and when should you use ImageReady? Although ImageReady 2.0 ships with Photoshop 5.5, it is still a standalone program. That means it must be launched separately and therefore takes up RAM and hard disk space. Switching back and forth between the two programs is relatively easy, especially between Photoshop 5.5 and ImageReady 2.0, but you still need to leave one work environment for the other, which is inconvenient at best and troublesome at worst. Some web producers have reported data corruption when they've tried to toggle between Photoshop and ImageReady with files containing several type and mask layers. I suggest you stick with Photoshop 5.5 for image optimization and use ImageReady only for such features as slicing, animation and rollovers.

Previewing compression options

In Chapter 1, *Making Photoshop Web-Friendly*, we discussed that differences in monitor gamma will make images that look good on a Mac look dark on a PC and images created on a PC look washed out on a Mac. The Save for Web plug-in includes a feature to preview how your image will look on different platforms. Click on the small arrow in the upper right of the Save for Web window and a drop-down menu appears, shown in Figure 3-6. Choose the platform in which you wish to preview your work. You can also choose Browser Dither, which shows how your graphic will look if it is forced to dither by a browser.

Figure 3-6.



To preview your graphic in an actual browser window, click on the browser icon on the bottom right of the Save for Web window. This brings up a pop-up menu where you can select the browser to use, as shown in Figure 3-7. Photoshop places aliases (or shortcuts, if your prefer) of all browsers found on your hard disk in a folder called Preview In, located in Photoshop's Helpers folder. One of these aliases is surrounded by square brackets. To change the default browser, remove the brackets from the existing browser and add them to the program you want to be your default.

Figure 3-7.

Undocumented feature

In the Save for Web window, clicking Control-Shift (on the Mac) or Control-Shift then right-click (on Windows) brings up a menu combining the Magnify, Platform Preview, and Browser Dither menus.

Working backwards

In the previous example, I started with the assumption that quality was the first priority. With the Save for Web plug-in you can also work backwards and prioritize file size. This can be especially useful when you are creating an ad banner that has a specific file size requirement.

To do this, go to the arrow in the upper, far right corner. This Settings pop-up menu includes a command called Optimize to File Size, as shown in Figure 3-8. Selecting this command brings up the dialog box shown in Figure 3-9. Here you can set the desired file size you want to squeeze the graphic into. There are also two radio buttons. Current Settings will keep your file format, changing palette or compression options to fit to the file size. Auto Select GIF/JPEG not only cycles through various palette options until it arrives at the desired file size, but it will also automatically choose whether GIF or JPEG is the better file format.

Figure 3-8.

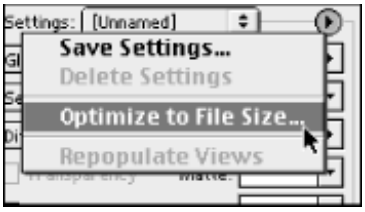


Figure 3-9. Auto Select GIF/JPEG not only optimizes to file size but also picks the best format.

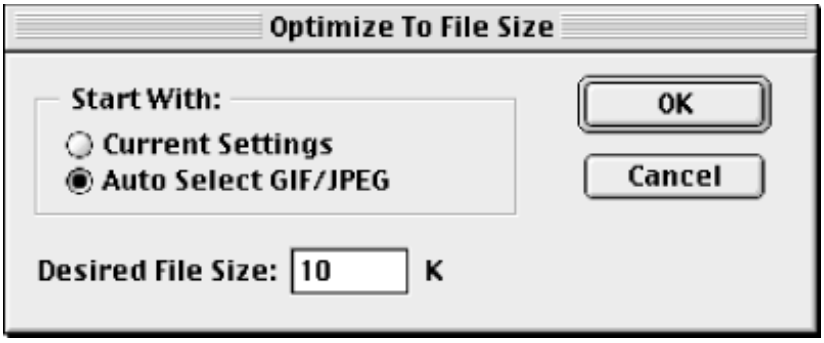


Figure 3-10. The Action palette's Batch dialog box.

Automate/Batch

You can create a folder containing a batch of similar images and use a combination of Photoshop 5's Automate/Batch feature (File → Automate → Batch) and an action to automatically optimize all the images for you. See Figure 3-10. Start by opening an image from the folder and name and record an action that optimizes that image to your liking. You can use either 5.5's Save for Web method or Image → Mode → Indexed Color method.

When you are finished, choose File → Automate, select the recorded action and the folder containing the images you wish to optimize, and Photoshop applies your action to the entire batch of images automatically. If you use Save for Web to optimize your representative image, a Photoshop action will record the various optimizing settings. Unfortunately, however, if you choose Optimize to File Size → Auto Select GIF/JPEG, Photoshop will not automatically analyze each image in your folder but will only apply the optimized settings that were calculated when you optimized your first image and recorded the action.

The Indexed Color option

In most versions of Photoshop, choosing Image → Mode → Indexed Color is an effective way to index your image in preparation for saving as a GIF. Using this method you can even do things that can't be done with the Save for Web plug-in, such as tweaking the palette to emphasis certain areas of your image and applying selective dithering; in addition, the GIF89a Export module offers much more control over transparency than does Save for Web.

Photoshop 5.5 has the best Indexed Color capabilities to date. Not only can you preview the effects of your optimizing choices in real-time, but you can apply variable amounts of dither as well. With 5.5's Color Table (Image → Mode → Color Table), there is a certain degree of interactivity which makes selectively adding browser-safe colors easier. There are also more ways to create transparency in 5.5, something I will discuss in more detail in Chapter 5. In Photoshop 5.5, you can go directly from an RGB image to a GIF file via File → Save a Copy; when you choose CompuServe GIF as the file format, the Indexed Color dialog box automatically appears.

To illustrate the Image → Mode → Indexed Color method, I'll index another of Valerie's nationalgeographic.com graphics. I'll also note where 5.5's features differ from earlier versions of Photoshop.

Again, I'll try two approaches. One way is easy but won't necessarily produce the optimal graphic. The other way takes a little more time, but the results are better.

Here are the steps to the first method:

1. Working with a RGB image, select Image → Mode → Indexed Color. This brings up the dialog box shown in Figure 3-11.
2. Apply the following settings:
3.
   Palette: Selective

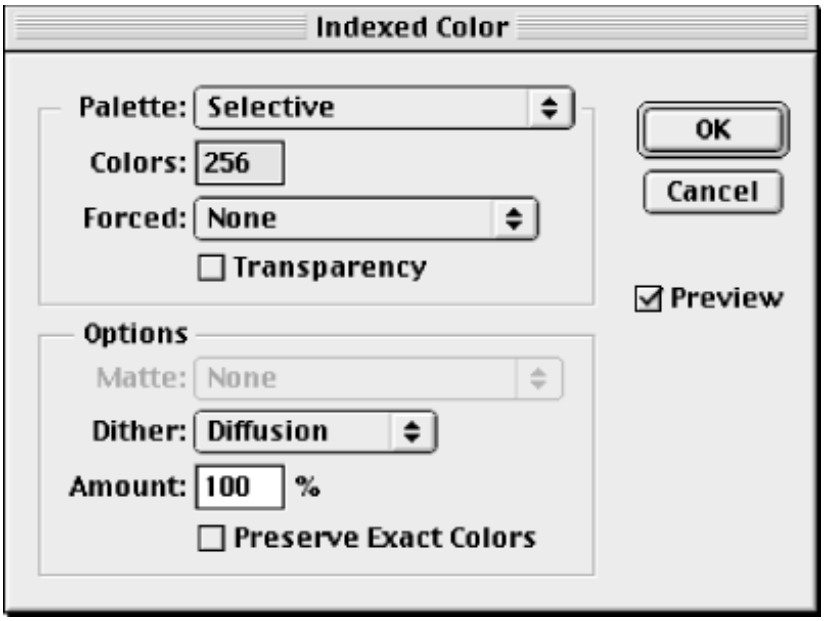
   Colors: 256

   Forced: None

   Dither: Diffusion

   Amount: 100%
4. Save the image, and then Save As to save it as a CompuServe GIF file.

Figure 3-11. The Indexed Color dialog box.



In most cases, when a graphic contains more than 256 colors, these settings will produce an acceptable GIF, as illustrated by Val's graphic shown in Figure C-35.

I'll try again. I'll keep my palette set to Selective because I know from experience that it will do a good job on a graphic like this one. (If your graphic contains less than 256 colors, the Exact palette will automatically be selected, which is fine unless you want to use the Web palette. If you apply the Web palette, be sure to follow the instructions outlined in "Special Web palette considerations" later in this chapter.)

Browser dither

One of the web browser's many jobs is to manage color. This is an issue because web pages typically have numerous images on a single page, so it would be impossible for an 8-bit system to display each image's palette.

That's why browsers have their own palette, generally known as the browser-safe palette, shown in Figure C-33. This is a 216-color palette, the colors of which are common to both Mac and Windows platforms, shown in Figure C-34. Technically, it's called a 6 × 6 × 6 color cube because the 216 colors are generated by combining just six colors.

The use of browser-safe colors is an important concept to understand if you want your images and graphics to look their best on the greatest number of display systems. First of all, if everyone surfing the Web used a display system capable of displaying millions or even thousands of colors, we wouldn't need a discussion about browser-safe colors. There would be plenty of colors to go around. But the fact is the majority of Web surfers use 8-bit display systems.

In general, 8-bit Macs use the 256-color system palette, which contains the 216 browser-safe colors plus 40 more, and Windows computers always use the browser-safe palette. There do seem to be times, however, when Macs use the smaller palette.

As long as your image contains only the 216 colors in the browser-safe palette, there's no problem. What happens when your image contains non-safe colors? The browser either uses dithering to simulate the missing color, or it entirely changes the color to one of the colors in its palette.

Now imagine an image on a blue background. If you pick whatever blue you want and index it, do you get a nice flat color? Not necessarily. If it's a non-browser-safe blue you get dithering, because the browser will try to approximate the color you asked for from the colors in the browser palette. No good.

The solution, of course, is to pick a browser-safe blue for that field. Although you don't have control over what kind of display system your audience is using, nor do you have control over the way in which a browser dithers or changes colors in your image or graphic, you do have control over which colors you choose to use. When you work with the 216 colors that are common to both the Mac and Windows machines, you can be assured that they will not dither or be replaced by a different color. Of course, these colors may not always look the way that you expect because of individual monitor calibration (or lack thereof) and the gamma characteristics of various computer platforms--but the differences are relatively predictable.

The best way to use browser-safe colors is to apply them to areas of large expanses of flat colors, while using dithering on areas of continuous tone, subtle gradation, and anti-aliasing.

I'll try reducing the number of colors and see what that does. At 64 colors the graphic still looks fine, but at 32 colors (see Figure C-36), it doesn't, so I'll stick with 64 colors. (Under the word Preview in the Indexed Color box, a short dash pulses until the optimizing is complete.) Zooming in and out (Control/Command zooms in and Alt/Option zooms out), I can carefully examine the effects of my changes. With Photoshop 5.0 and 5.02, which also have preview capabilities, you can't zoom in and out while your Indexed Color box is open. You must magnify your image before choosing Image → Mode → Indexed Color. With other versions of Photoshop, preview is not an option.

With 5.5, you can fine-tune the amount of dither. After some trial and error, I notice that my graphic still looks good at 80% dither. The less dither, the smaller the file size, so this reduction in dither is good. With earlier versions of Photoshop, you can only turn dither on or off, and in nearly all cases when you have a complex graphic or a photograph, you'll want to leave it on. Turn dither off for graphics with less than 256 colors.

I'm happy with what I see and now the file size is only 19.8K.

Replacing non-browser-safe colors

Let's turn now to the browser-safe color issue. As mentioned earlier, it is not visually pleasing when solid, large expanses of color (such as those found in a background) dither. You could apply the Web palette to the entire graphic, but with a complex graphic, the results generally are not good. Instead, it's better to selectively replace a background or solid color with a browser-safe color and not worry about the other areas.

With 5.5's Save for Web plug-in, selective color replacement is easy. It's a bit more work if you are using the Image → Mode → Indexed Color method, but it can still be done. Here are two methods: the first method requires that you work in RGB mode, replace background colors with browser-safe colors from the swatch or color picker, and *then* index your image; the second method occurs while in Indexed Color mode and uses the Color Lookup Table.

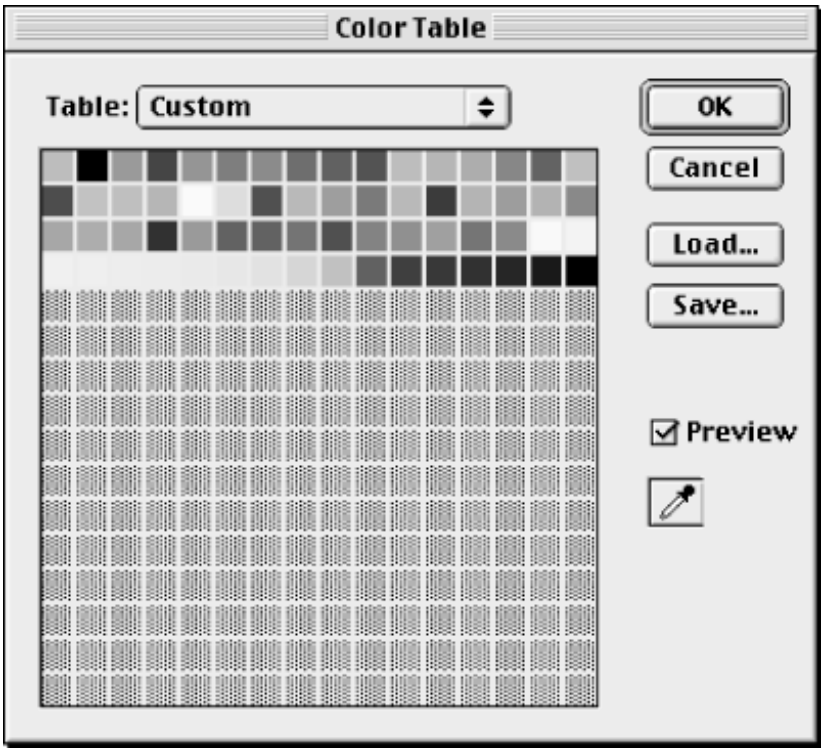
The RGB method:

1. Fill the foreground color box with a browser-safe swatch. (You can select browser-safe colors from your swatch palette or from your color picker.)
2. Use either Photoshop's Magic Wand, magnetic lasso tool, or manual selection tools to select the area of the image that you want to change. You could also use the Color Range tool. (If you use the Magic Wand tool, be sure to experiment with the Grow and Similar commands found under the Select menu. These options are especially useful to isolate large expanses of background or areas of similar color.)
3. Fill the selected area with the color that you chose from a browser-safe palette.
4. To ensure that your browser-safe colors don't shift when you apply the Indexed Color command, check the Preserve Exact Colors checkbox in the Indexed Color dialog box. (This is only an option in Photoshop Versions 5.0, 5.02, and 5.5. With earlier versions of Photoshop, you will need to manually check your colors after you've indexed your graphic to make sure the colors haven't shifted. Sometimes they shift and sometimes they don't. It depends on the image and the palette you apply.)

The Indexed Mode method:

1. Choose Image → Mode → Color Table (see Figure 3-12).
2. Select the eyedropper tool.
3. Click on the area in your image that you wish to alter. The color box in the Color Table will be highlighted and the color you selected will be deleted from the box and from the image.
4. Double-click on the highlighted color box in the Color Table to bring up the Color Picker. Select the browser-safe color of your choice and click OK. That color will now replace your old, non-browser-safe color.
5. Repeat this process on other flat colors that you don't want to browser-dither.

Figure 3-12. The Color Table dialog box.



In pre-5.5 Indexed Color Mode, you can use the Color Lookup Table to replace a non-browser-safe color in earlier Photoshop versions. However, this method is a bit awkward because the earlier Color Lookup Tables are not interactive. You can choose a color in the table to edit the image, but you can't click on a color in the image to bring up that color in the table and change it. This makes finding the color you want a hit-and-miss proposition.

To change a color using this method, follow these steps:

1. Open your indexed image.
2. Choose Image → Mode → Color Table.
3. Find the background or dominant color in the Color Table that you want to change to a browser-safe color. (This process is hit-and-miss and requires some guesswork.) Click on the color and the Color Picker appears. Either type in the RGB values that equate to a browser-safe color or simply use the eyedropper tool to click on a browser-safe color in the Swatches palette.

Special Web palette considerations

If you apply the Web palette, you'll end up with an indexed image that contains 216 browser-safe colors. But what if you don't need 216 colors? What if your image contains only 64 colors, or, for that matter, 16? By saving your file with 216 colors you have created an unnecessarily large file. What to do now? You can get rid of the unused colors by following these steps:

1. Switch to RGB mode.
2. Without doing anything else, switch back to Indexed Color mode. If your image has less than 216 colors, the Exact palette should be selected, and the number of colors in the image will be displayed.
3. Click OK, and save the indexed image as a GIF.

The new image will look exactly the same as the original, while the file size will shrink.

Customizing the Adaptive, Selective, and Perceptual palettes

When Photoshop converts an image from the RGB mode to the Indexed Color mode, it treats all parts of the image equally. It doesn't recognize, for example, that you might be more concerned about the foreground of the image than the background. But you're not stuck with this situation--you can create a palette that best represents the important parts of your image.

There are two ways to do this:

- "Tweak" (or influence) the Adaptive, Selective, or Perceptual palettes.
- Create a custom palette.

The first way is easier. To tweak one of the three above mentioned palettes, simply select the area that you want to emphasize with a selection tool. Now when you convert from RGB to Indexed Color mode, Photoshop will automatically weigh the conversion in favor of the selected

area.

In Figure C-37, for example, Rebecca's face is the most important element in the image. However, if the Adaptive palette is applied normally to this photograph, too much emphasis will be placed on the colorful background and not enough of the colors that make up the skin tone will be represented. As Figure C-38 shows, the results are not flattering.

But we can tweak the palette by having Photoshop emphasize the colors in the face and de-emphasize the colors in the background. To do this, simply select the face before indexing, as shown in Figure C-39. The new palette, shown in Figure C-40, contains many more of the subtle tonal variations that make the skin tone more realistic.

In the previous example, we got good results by influencing the Adaptive palette. But we could have gotten even better results if we had created a CLUT (Color Look-Up Table) that put total rather than partial emphasis on the selected area. To do this, you need to create a custom palette. Once you've created this custom palette, you can apply it to your original image or to one comparable in color.

To create a custom palette, follow these steps:

1. Open an RGB image.
2. Make a copy of that image (Image → Duplicate), merging layers if necessary. You'll work on this copy to create the custom palette, then apply the palette to your original.
3. Crop the duplicate image to the most color-critical area. This part of the image will control the color palette.
4. Index this cropped image with the adaptive palette.
5. Save this palette by selecting Image → Mode → Color Table and clicking Save. Give the palette a name and click OK.

Now that we've created a custom palette, we can apply it to the original image. Just index the original and choose Custom palette. This brings up the Color Table dialog. Click on the Load button and select the custom palette you just created. Photoshop will apply the custom palette to your image.

As you can see in Figure C-41, Rebecca's face is nearly color-perfect. However, since our custom palette doesn't include any colors from the background, the background looks terrible. A solution to this problem would have been to include a few background colors in our custom palette, which could have been done by cropping our image looser.

Selective dithering

When you command Photoshop to dither an image, it applies the dither pattern to the entire image, even to areas that you might not want to dither, such as areas containing flat colors. Wouldn't it be great if you could "tell" Photoshop to selectively dither a small part of an image and yet keep the other parts intact?

clnet's Casey Caston has found a way to do just that. To use Caston's method of selective dithering, follow these steps:

1. In RGB mode, copy the part of the image to dither.
2. Index the image.
3. Paste the copied portion back from the Clipboard to the indexed but undithered image. Photoshop automatically dithers the pasted RGB selection, leaving the rest of the image untouched.

Caston regularly uses this technique to give him very specific control over the way his clnet graphics look.

The next three color figures show the results of this approach to indexing a GIF. Figure C-42 shows the graphic indexed to the Web palette. Figure C-43 shows the image with the adaptive palette. And Figure C-44 shows the results of the combination approach: some of the image is dithered, some isn't.

How big is your image file anyway?

As a creator of web content, you need to pay attention to the file size of your image or graphic. Many web designers have decided that the total size for a web page shouldn't be more than 30K, so every kilobyte counts. How can you determine the actual file size of an image you are currently working on?

Photoshop 5.5's Save for Web plug-in displays an accurate file size in the lower-left corner of its window. However, the size that is displayed on the bottom left of Photoshop's main image window has very little to do with actual number of kilobytes that make up the image. If you are working with a JPEG image, it has nothing to do with the actual compressed file size. (Display the size by positioning the pointer over the triangle in the bottom border, holding down the mouse button, and choosing Document Sizes.) These numbers are useful when you want to know the approximate difference in size between a layered and a flattened image. The first value shows the file size of the final file as it would be if its layers were flattened. The second value shows a file size that includes all layers and channels.

When you choose Image → Image Size, the file size number displayed is only a rough approximation of the actual number of bytes of your image.

In versions other than 5.5, you'll need to leave Photoshop and click on your desktop to find the actual file size. In Windows, the file size you see in the file list is accurate. On the Mac, however, this number is usually way off. The Macintosh Finder displays not the size of the file but the amount of hard drive space consumed. To find the actual size of a file, use Command/Control-I (or File → Get Info). The first number shown next to Size is the amount of hard drive space the file occupies. The number next to it in parentheses is the more accurate number.

Ultimately, what really matters is the final size of the image file once it is on a web server. This final file size is often much smaller than you think, because unwanted and unneeded data is stripped in the process of putting an image on a server. This is especially true for Macintosh files because the Mac adds a "resource fork" that is always stripped away by a web server.

Getting better tiled backgrounds

When an image or graphic is used as the background for a web page, you have to be especially careful to use browser-safe colors. This is because browsers handle background images differently than other images. When it comes to backgrounds, the browser will shift non-browser-safe colors instead of dithering them.

This became quite apparent to designer Gregg Hartling when he tried to use a GIF background tile that he had created using the Adaptive palette. Gregg knew that by using the Adaptive palette he ran the risk of introducing colors that weren't browser-safe. He thought the image might dither when displayed on an 8-bit system, but he didn't see that as a problem because the dither would add a nice effect to the image.

But the background GIF didn't dither. Instead, as you can see in Figure C-45, the browser substituted colors from the Web palette and that's all it did. The result was a clumpy, banding-like effect. Gregg quickly went back to his original image and used the browser-safe palette with diffusion dither selected and got the results he wanted, as shown in Figure C-46.

Saving time with Actions

Earlier versions of Photoshop don't have a Indexed Color Preview option, and it is time-consuming to find the happy medium between file size and quality. You have to convert and save many different versions of your image until you find the perfect balance. If you are using Photoshop 4, you can automate the process of applying different color depths and palettes to a single image with Actions.

Take this tip from Sean Parker, the technical wizard at Parker Grove. Sean created an Action that applies the different color depths to the Adaptive palette with diffusion dither, and then applies the Web palette with dither. He has also created an action that applies the different color depths to the adaptive palette with no dither, and then applies the Web palette without dither.

A little more work

As you've seen in this chapter, using Photoshop to index a graphic in preparation for the GIF file format takes a bit of work, especially if you are trying to get it completely right. Remember, what may seem daunting at first becomes easier and easier as you become familiar with the various Photoshop settings and how they affect different types of images and graphics. In the next chapter, we'll show you ways to maintain even more control over the color in your graphics by using browser-safe colors from the start.

Back to: Photoshop for the Web, 2nd Edition

oreilly.com Home | O'Reilly Bookstores | How to Order | O'Reilly Contacts
International | About O'Reilly | Affiliated Companies | Privacy Policy

Color Modes

as Described by Adobe

RGB Color mode

Photoshop's RGB Color mode uses the RGB model, assigning an intensity value to each pixel ranging from 0 (black) to 255 (white) for each of the RGB (red, green, blue) components in a color image. For example, a bright red color might have an R value of 246, a G value of 20, and a B value of 50. When the values of all three components are equal, the result is a shade of neutral gray. When the value of all components are 255, the result is pure white; when the values are 0, pure black.

RGB images use three colors, or channels, to reproduce colors on-screen. The three channels translate to 24 (8 bits x 3 channels) bits of color information per pixel. With 24-bit images, up to 16.7 million colors can be reproduced. With 48-bit images (16 bits per channel), even more colors can be reproduced. In addition to being the default mode for new Photoshop images, the RGB model is used by computer monitors to display colors. This means that when working in color modes other than RGB, such as CMYK, Photoshop interpolates the CMYK image to RGB for display on-screen.

Although RGB is a standard color model, the exact range of colors represented can vary, depending on the application or display device. Photoshop's RGB Color mode varies according to the working space setting that you have specified in the Color Settings dialog box. (See About working spaces.)

CMYK Color mode

In Photoshop's CMYK mode, each pixel is assigned a percentage value for each of the process inks. The lightest (highlight) colors are assigned small percentages of process ink colors, the darker (shadow) colors higher percentages. For example, a bright red might contain 2% cyan, 93% magenta, 90% yellow, and 0% black. In CMYK images, pure white is generated when all four components have values of 0%.

Use the CMYK mode when preparing an image to be printed using process colors. Converting an RGB image into CMYK creates a color separation. If you start with an RGB image, it's best to edit first in RGB and then convert to CMYK at the end of your process. In RGB mode, you can use the Proof Setup commands to simulate the effects of a CMYK conversion without changing the actual image data. (See Soft-proofing colors.) You can also use CMYK mode to work directly with CMYK images scanned or imported from high-end systems.

Although CMYK is a standard color model, the exact range of colors represented can vary, depending on the press and printing conditions. Photoshop's CMYK Color mode varies according to the working space setting that you have specified in the Color Settings dialog box. (See About working spaces.)

Lab Color mode

In Photoshop, the Lab Color mode has a lightness component (L) that can range from 0 to 100. In the Adobe Color Picker, the a component (green-red axis) and the b component (blue-yellow axis) can range from +127 to -128. In the Color palette, the a component and the b component can range from +120 to -120.

You can use Lab mode to work with Photo CD images, edit the luminance and the color values in an image independently, move images between systems, and print to PostScript Level 2 and

Level 3 printers. To print Lab images to other color PostScript devices, convert to CMYK first.

Lab images can be saved in Photoshop, Photoshop EPS, Large Document Format (PSB), PDF, Photoshop Raw, TIFF, Photoshop DCS 1.0, or Photoshop DCS 2.0 formats. 48-bit (16 bits per channel) Lab images can be saved in Photoshop, Large Document Format (PSB), Photoshop PDF, Photoshop Raw, or TIFF formats.

Note: The DCS 1.0 and DCS 2.0 formats convert the file to CMYK when opened.

Lab color is the intermediate color model Photoshop uses when converting from one color mode to another.

Bitmap mode

This mode uses one of two color values (black or white) to represent the pixels in an image. Images in Bitmap mode are called bitmapped 1-bit images because they have a bit depth of 1. (See Specifying 8-bit color display (Photoshop).

Indexed Color mode

Indexed Color mode produces 8-bit image files with at most 256 colors. When converting to indexed color, Photoshop builds a color lookup table (CLUT), which stores and indexes the colors in the image. If a color in the original image does not appear in the table, the program chooses the closest one or uses dithering to simulate the color using available colors.

By limiting the palette of colors, indexed color can reduce file size while maintaining enough visual quality for certain applications like a multimedia presentation or a Web page. Limited editing is available in this mode. For extensive editing, you should convert temporarily to RGB mode. Indexed Color files can be saved in Photoshop, BMP, GIF, Photoshop EPS, Large Document Format (PSB), PCX, Photoshop PDF, Photoshop Raw, Photoshop 2.0, PICT, PNG, Targa, or TIFF formats. (See Converting to indexed color (Photoshop).)

Duotone mode

Duotone mode creates monotone, duotone (two-color), tritone (three-color), and quadtone (four-color) grayscale images using two to four custom inks. (See Printing duotones.)

Multichannel mode

Multichannel mode uses 256 levels of gray in each channel. Multichannel images are useful for specialized printing. Multichannel mode images can be saved in Photoshop, Photoshop 2.0, Photoshop Raw, or Photoshop DCS 2.0 format.

These guidelines apply to converting images to Multichannel mode:

- Color channels in the original image become spot color channels in the converted image.
- When you convert a color image to multichannel, the new grayscale information is based on the color values of the pixels in each channel.
- Converting a CMYK image to multichannel creates cyan, magenta, yellow, and black spot channels.
- Converting an RGB image to multichannel creates cyan, magenta, and yellow spot channels.
- Deleting a channel from an RGB, CMYK, or Lab image automatically converts the image to Multichannel mode. (See About color channels for more information on channels.)
- To export a multichannel image, save it in Photoshop DCS 2.0 format.

Home Page

www.Inkjetcolorsystems.com
American Imaging Corp.
32 Broadway, Hillsdale NJ 07642
Tel: 201 263 9177 fax: 201 263 9533
inkjetcolorsystems@verizon.net

To order by E-Mail
inkjetcolorsystems@verizon.net

Video Compression Workshop Outline

Digitized video is now commonly viewed via the web, in electronic presentations such as PowerPoint slide shows, and on CD and DVD. Depending upon the environment, it needs to be compressed in different ways to play on these digital devices.

Outcome: Prepare video files for playback over the web and within a PowerPoint presentation using Cleaner 5 software.

Goals:

- Gain a general understanding of what video compression is and why it is necessary.
- Understand the differences between commonly used codecs, and when to apply them.

Workshop Outline:

- Lecture: Where compressed files are viewed; definition of compression; why compression is important; a computer is not a TV; ways of reducing video file size; commercial codecs: what they are, how they work and which to use for different outputs; compression in Premiere vs. Cleaner 5; choosing a file format

- Demonstration: Introduction to Cleaner 5 and preparing a file using the Wizard

- Exercise: Prepare a video for the web

- Demonstration and Exercise: Crop and prepare a video for a PowerPoint presentation

- Demonstration: How to change specific compression parameters, compressing multiple files in a batch

Handouts:

1. Course outline: compression overview and why it is necessary, choosing a codec, choosing a file format, links, definitions
2. How to export a Premiere video composition for use in Cleaner 5
3. HTML code to insert a QT movie for the web
4. Copies of the Workshop posters (Compression Choices and 3 Major Points about Compression)
5. Compression chapter from Terran compression booklet
6. Video compression techniques used by codecs and other factors to consider about a codec
7. MPEG format review article

Why is compression important?

All video files must be made smaller to allow them to play back in normal multimedia environments. There is a lot of information in an uncompressed video file. The DV format is 720 x 480 pixels, with 3 bytes of data per pixel, playing at 29.97 frames per second. This means that for one second of video, you need almost 27 MBytes of disk space! A 10 minute video would need almost 16.2 GB of space. A computer would need a special accelerated processor to playback at 27 MBytes per second. A 300 MHz computer cannot play back such a file even if hard disk space was not a problem.

What is video compression?

In this workshop, we define compression as formatting a video file so it can play back and be stored on a personal computer. A compressor can be a piece of software, or hardware, or both. The goal of a good compressor for acceptable playback is to produce a video with a small file size that looks similar to the original movie when being played on a computer. A poorly compressed file has unacceptable image quality, jerky motion during play back, and/or large file size.

In the Video Editing class, the term compression was also discussed but for a different purpose. A DV signal (i.e., coming directly from a digital camera or DV tape) can be transferred to a computer's hard drive via firewire. Firewire does not compress the signal. The other way to import a video signal to a computer, i.e., an analogue signal from a VHS tape or Hi-8 tape, is to use a special computer capture card. This hardware digitizes and compresses the signal. This kind of process tries to compress as little as possible and still allow the video file to be manageable by the computer. These files can be huge and retain as much information about the video as possible.

This workshop focuses on how to distribute digitized video files over the web and within a PowerPoint presentation. In this case, compression is used to shrink the file size as much as possible while maintaining high quality output and successful delivery.

- Compression when capturing from a camera or deck tries to compress as little as possible to get a robust source file.
- Compression for delivery tries to compress as much as possible and still maintain high image quality and successful playback.

Your computer is not a TV!

Although a computer monitor looks like a television screen, a computer is not designed to be one. There are important differences between the two. A television is a specially designed receiver of broadcast signals that tunes to different frequencies by the changing of channels. It displays a relayed signal. A computer is a processor. When it displays a movie, it is processing every piece of information about that movie, such as the frame size, color depth of each pixel, etc. A computer has to process an incredible amount of data to play back a video file.

A television screen is typically 640x480 pixels. The broadcast signal it receives contains data for all pixels and thus fills the screen. The dimensions of DV format are 720 x 480 pixels. A typical computer monitor can display 1024 x 768 pixels. There are usually more pixels on the computer screen than needed for the video. This means that the movie does not fill the entire screen.

To get a video file to a format that a computer can process requires throwing away some of the information in that file. A video playing back on a computer will not look as good as on a television. Although compression algorithms are quite clever in representing the video file with as little loss of information as possible, they just aren't quite there yet.

What are some easy ways to compress a video file?

There are some simple ways to reduce the amount of information a computer has to process when playing the video file. These are reducing the **frame size**, **frame rate**, and image quality.

Frame Size:

Make the frame size smaller. File size can be reduced by a factor of 4.5 by reducing the movie's displayed frame size from 720 x 480 pixels to 320 x 240 pixels. Compression choices are compromises. All else being equal, the video with larger dimensions will have poorer image quality. Choose frame sizes that are multiples of 4. Common sizes are 320 x 240 pixels and 240 x 180 for playback from a hard drive. Here are more guidelines considering the delivery media:

Delivery Medium	**Suggested Frame Size (pixels)**
56.6 Modem	160x120
ISDN	192x144
T1 line	240x180
CD-ROM	320x240
Hard Drive (e.g., PowerPoint)	320x240

Frame Rate:

Decrease the frame rate. Video's inherent frame rate is 29.97 fps. This rate is high enough for our eye to perceive smooth motion. Depending on the video content, you can decrease this rate to half that amount and still play an acceptable video. We recommend first trying 15 fps. If you are trying to show very smooth motion, then you may want to go higher than 15 fps but never exceed 29.97 fps for normal playback speed. A good starting point for playback on a new Macs and PC's with Pentium chips is 320x240 pixels frame size and a frame rate of 15 fps.

Image Quality:

Reduce the pixel's bit depth. Depending on the nature of your movie, it may not need all 24 bits of data per pixel. Simply reducing the bit depth of each pixel multiplicatively reduces the data rate, i.e., reducing a 24-bit video to a 16-bit video reduces the file size by a factor of 1.5.

What else can be done to compress a video file even more?

Adjusting the above settings in conjunction with using a **codec** gives the most successful files for video playback. A **codec**, a **co**mpressor and **dec**ompressor, creates a new representation of the original video file by applying a mathematical operation to compress the video file. It reverses the algorithm during playback of the compressed file. The reversal typically loses some of that original information and causes the video to look different from the original file.

There are hundreds of codecs. In this workshop, we focus on three mainstream codecs: Sorenson Video, Radius Cinepak, and Apple Animation.

To play back a movie, you use a software called a "Player". The top three (competing) players are Apple's QuickTime, Microsoft's Windows Media Player (mainly for PC's only), and RealNetworks' RealOne Player. QuickTime and Windows Media Player can play a variety of **formats**. It is similar to Photoshop being able to open a variety of formats, such as TIFF, JPEG, GIF, PostScript, etc.

We need to distinguish between a Format and a Codec. A format is also known as an architecture. The format controls all the elements of the movie: the video track, the audio track, the text tracks, synchronization between tracks, etc. The two most common formats are ".mov" – the Apple format; and ".avi" – an early PC format from Microsoft. The architecture "understands" or "contains" the various codecs and puts them in motion when needed. (See more on formats below.)

Choosing a codec:

Commercial codecs are algorithms based on several methods of compressing and restoring video data. Some are better (i.e., they produce small file size, smooth playback, and great image quality) for animation than for preserving fast motion. The five most common codecs are: (1) Sorenson, (2) Cinepak, (3) Animation, (4) WindowsMediaVideo, (5) RealVideo.

1. Sorenson: This codec places high demands on the CPU for playback. It is a good choice for movies with a low data rate: less than 100 KBps. It is also good for video with lots of motion. It achieves higher image quality at a fraction of the data rate compared with other codecs.

If you are compressing a movie for a PowerPoint presentation and are writing it out in AVI format, the Sorenson codec is not available. AVI format does not support this codec. Use the Cinepak codec.

2. Radius Cinepak: This codec has low CPU demands for playback but doesn't look good for data rates less than 250 KBps. It doesn't have the great image quality for lots of motion compared with Sorenson but does a fine job with slower motion subjects. An advantage to Cinepak is that older versions of QuickTime recognize it and it plays on older machines. Cinepak plays well on most mid-range and faster machines (fast 68040 Mac or any PowerMac, fast 486 PC or and Pentium). It is a good choice for AVI format for PowerPoint presentations. If you need a movie for playback on a Sparc and/or Unix stations, try QuickTime format with a Cinepak codec and play it with the xanim tool.

Cinepak must always compress movies at least 10:1, so it is less useful at higher data rates (4x CD-ROM and above). This algorithm uses the Color Look Up Table (CLUT) method *(see the Video Compression Techniques used by Codecs handout for more information)* and processes the video in averaged 4 x 4 pixel blocks.

3. Apple Animation: This codec was created to compress and playback computer generated animation clips. On normal video clips, the results may be disappointing. The algorithm uses the Run Length Encoding (RLE) technique *(see the Video Compression Techniques used by Codecs handout for more information)* that does a poor job encoding information of frequent color transitions. This codec boasts of lossless compression and decompression of sequences of line art and rendered still images. This codec cannot be used to create AVI files.

Choosing a format:

This workshop focuses on QuickTime format (.mov) and the Microsoft Audio Visual Interleave format (.avi).

There is another format, Moving Pictures Experts Group (MPEG, .mpg) that comes in different versions for different purposes.

- MPEG1 (.mpg): For video playback on a Unix machine; to make a Video CD
- MPEG2 (.mp2): DVD video format, full screen (720x480), basis for new digital television broadcast standard (HDTV)
- MPEG4 (.mp4): new emerging format for Internet 2/handheld devices

MPEG3 was developed but was folded into MPEG2. There is a format called MP3 (.mp3) which is an audio only format and is actually a layer of the MPEG1 format. Video compression in the MPEG2 format offers the highest quality playback available with special hardware and software. See the handout on MPEG formats and MPEG encoding for more information.

Compression Quality in Cleaner 5 versus Premiere

In the Video Editing class, clips were captured from the digital camera for the video assembly process at approximately 3.5 MBytes per second (MBps). A one minute video requires 210 MB of disk space. Most computers could not play this video file back due to the great demands of the data rate. To allow playback of edited movies, Premiere has several codecs to choose from when exporting movies. Here are some available ones under the Video Settings when you choose to Export a Movie: Planar RGB, Intel Indeo Video 4.4, Sorenson Video, Cinepak, DV-NTSC, DV-PAL, H.261, H.263, Motion JPEGA, Motion JPEG B, and None.

Frame size, frame rate, color depth, and data rate can also be set. Different file formats Premiere can export are found under the General Settings including QuickTime, Microsoft AVI, TIFF sequence, Targa sequence, GIF sequence, Animated GIF, and Windows Bitmap sequence.

Premiere and Final Cut Pro both do good jobs at compressing movies. However, Cleaner 5 does it better and generally faster. Cleaner 5 is software specifically designed to encode video and files. Results of compressed files from Premiere versus Cleaner 5 show that Cleaner 5 is superior. However, the information taught in this workshop about how to compress files and what parameters are important apply to all tools which compress.

The Digital Media Center uses Cleaner 5 to compress video and audio files. Movies are exported from Premiere in an uncompressed format such as a DV stream or uncompressed QuickTime.

Summary of choices for different delivery media

Delivery Medium	Frame Size	Computer Platform	File Format	Codec
Power Point	≤ 320x240	PC	AVI	• Cinepak • Apple Animation
		Macintosh	QuickTime	• Sorenson • Cinepak • Apple Animation
Web (download) 56 Kb modem LAN	160x120 ≤ 320x240	PC, Macintosh	QuickTime Windows Media Video	• Sorenson • Cinepak • Apple Animation • Windows Media
CD-ROM	356x240	PC, Macintosh, Unix	MPEG 1	MPEG1
DVD Video	720x480	PC, Macintosh, DVD Player	MPEG 2	MPEG2

Cautionary notes for PowerPoint video files:

PowerPoint is the most commonly used slide presentation program today. To include video files, a computer needs at least 128 MB of RAM. From our tests at the DMC, the .avi format is the most reliable to use on a PC. PowerPoint claims to support other formats but those formats either stopped and/or stuttered during playback or crashed the computer entirely. Although the .avi format is old, it is reliable. PowerPoint takes up memory as the presentation progresses, so it can "run out" before the end of the presentation. Make sure to do a rehearsal run through of the entire presentation! A movie that plays when it was inserted when preparing the presentation may not play, especially if it is at the end of the presentation.

References

links:

- http://www.discreet.com/support/codec/ – all you want to know about codecs (formerly CodecCentral)
- http://www.adamwilt.com/DV-FAQ-editing.html#codecs – hardware versus software codecs comparison
- http://vlab.ee.nus.edu.sg/~csh/interframe/dvcompintro.html- good video compression definition and discussion with a diagramed analogy
- http://bmrc.berkeley.edu/frame/research/mpeg/index.html- details of MPEG and its history and development, great starting point with lots of links about MPEG basics
- http://www.jmcgowan.com/avialgo.html- description of AVI encoding
- http://www.apple.com/quicktime/products/tutorials/ General (and very good) information on Apple's QuickTime product to produce movies.
- http://www.shortcourses.com/- short courses for Digital Video; what kind of digital camera to buy
- http://www.2-pop.com/ - a well-moderated discussion site. You need to register for it.
- http://www.dvddemystified.com/ - an excellent resource on all aspects of DVD. Jim Taylor keeps it up to date.
- http://www.virtualdub.org/links Good compression links and technologies, compiled by Avery Lee, the genius compressionist with a free compression program for PC users.

books:

- *How to Digitize Video* by Nels Johnson (very basic book about DV and compression)
- Cleaner 5 manuals
- *How to Produce High-Quality QuickTime* by Terran Interactive
- *DVD Demystified*, Jim Taylor
- *DVD Authoring and Production*, Ralph LaBarge

software:
Cleaner 5, by discreet, http://www.discreet.com
Sorenson Squeeze/Sorenson Flash MX, by Sorenson, http://www.sorenson.com
VirtualDub, freeware by Avery Lee, http://www.virtualdub.com

Definitions

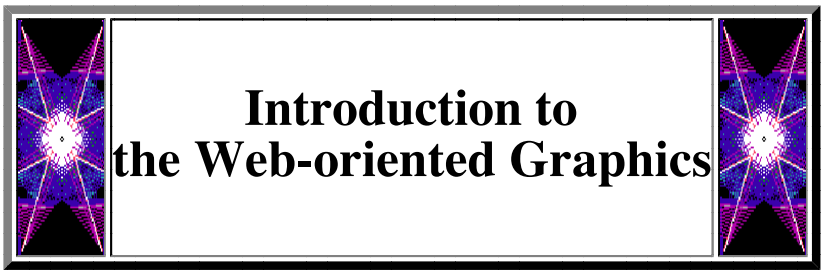
Bandwidth: The amount of information that can be sent, processed, etc. in a given amount of time. For instance, a 28.8 modem has a bandwidth of about 3KBps. Firewire has a bandwidth of 12.5-50 MBps.

Data Rate: The amount of information per second used to represent a movie. Expressed in units of bits/second (bps), Kilobits/second (Kbps), KiloBytes/second (KBps = 8*Kbps), etc. A 1x CD-ROM movie is made at a data rate of 100 KBps. The data rate of uncompressed NTSC video is about 27 Megabytes per second.

Frame Rate: The number of frames displayed per second in a video. NTSC (American) video format is 29.97 frames per second (fps). This parameter can be changed when a video is encoded to a lower rate.

Frame Size: The x and y dimensions of a movie. DV format is 720 x 480 pixels. A TV displays 640 x 480 pixels. An average computer monitor displays 1024x768 pixels. Compressed movies are usually much smaller, around 320x240 pixels, to play within a web page or PowerPoint presentation.

Key frame: A static, complete frame. A frame in a movie is either a key frame or a differenced frame. A key frame does not require another image to create a complete frame. A differenced frame contains only the information that makes up the difference between the current frame and the previous frame. A differenced frame is smaller in file size because it contains less information. For example, if your movie subject is a speaker at a podium with a fixed background, a differenced frame would only contains pixels of the movement of the speaker. It would not have information on the background because those pixels don't change from frame to frame.



Introduction to the Web-oriented Graphics

Qinxue Chen

1. Web graphics formats

Creating images is one of the biggest parts of designing content for the Web. There are two popular "in-line" Web graphics formats, GIF and JPEG. They have compression bulit-in.

GIF, which stands for Graphics Interchange Format, is a lossless method of compression. All that means is that when the program that creates a GIF squashes the original image down it takes care not to lose any data. It uses a simple substitution method of compression.

This image format can store black-and-white, grayscale, or color images but limited to a maximum of 256 colors (or gray shades), per image. There are two versions of GIF format. The GIF87A and the GIF89A. Most of us just call it GIF, butthe latter one has several features that the GIF87A does not has. GIF89A lets you make a transparent graphic and create an animation graphic known as Animated GIF, which are not supported by any other Web graphics format.

A big advantage using GIF format is that images can be stored in relatively small files. Access through the Internet can be very fast. GIFs are ideal for flat colour objects, and natural images that contain a fairly narrow range of tones. As the basic palette of a GIF is 256 colours, a complex image will ask Photoshop to bodge any intermediate colours. One thing GIFs are definitely not suited to is handling gradients: these nearly always appear banded no matter how carefully you adapt the image.

JPEG stands for Joint Photographic Experts Group and so anyone could easy understand that this image format has especially designed and developed for handling photographic content images. It is very good with photographs because 24-bit photographs do not dither. Unlike GIF it supports at maximum 16.7+ millions colors; an almost infinite number of colors. Like GIF it stores images in a compressed form. The images will be smaller than a GIF. JPEG has a very efficient compact digital form but unlike GIF file format, and it requires both compression and decompression. That is why JPEGs take longer to download even if the file(s) is smaller from GIF file(s). JPEG is a lossy compression method. In other words, to save space it just throws away parts of an image. The lossy compression method can generate artifacts - unwanted effects such as false colour and blockiness - if not used carefully. JPEGs are better suited to displaying natural, complex images. If on the other hand, quality isn't an issue, but size is then go for a lower quality JPEG, as it will be substantially smaller than the equivalent GIF. The trade-off is that you lose the control over the image that GIF gives you: it's less easy to specify colours and the quality can be dire.

Although GIF and JPEG are dominant there are other Web graphics formats, such as PNG or Lightning Strike compressed images. PNG (Portable Network Graphics) is an extensible file format for the lossless, portable, well-compressed storage of raster images. PNG provides a patent-free replacement for GIF and can also replace many common uses of TIFF. Indexed-color, grayscale, and truecolor images are supported, plus an optional alpha channel. Sample depths range from 1 to 16 bits.

PNG is designed to work well in online viewing applications, such as the World Wide Web, so it is fully streamable with a progressive display option. PNG is robust, providing both full file integrity checking and simple detection of common transmission errors. Also, PNG can store gamma and chromaticity data for improved color matching on heterogeneous platforms.

2. Built-in image-resizing ability in Web browsers

In HTML files, there is an option to specify the size of an image in the img tag. This means that you include the original image in the tag but can specify a smaller size for it to display. For Netscape only, you can re-size either the width or the height, and the browser will "know" how to scale the other attribute to the right proportion. IE does not do this, so just to be safe, you should determine the original pixel dimensions of the image, decide what you want the width to be, and get out your calculator to determine the proper height.

If you tell the browser to display the image at a smaller size, the browser will still be required to download the image at its full size. This takes extra time, creates a bigger load on your server, and most importantly increases the amount of limited bandwidth available to other Internet users. All graphical browsers will be able to display the tiny image, you should use an image tool to shrink the original image down to a small size, and then save the image under a new name.

3. Spatial and color resolutions approprite for the Web browser

With those techniques, you can put images on the web. Since your medium on which you will deliver your work is a computer screen and not any printing medium like high resolution printers, high resolution files and the Web do not go along. Usually computer screens display 72 pixels per inch, so everything you prepare on these dimensions is defined as low resolution. Everything above is defined as high resolution and it will appear at least twice in size on a monitor.

Bit-Depth is what defines how many colors your image will have. Of course the numbers of colors on the image has an enormous affect in your image's file size as well as to the image's quality. The more bits devoted to each pixel, the more colors can be displayed on the screen. 8-bit depth color resolution is suitable for Web browsers. With this resolution, the most-used 256 colors on the Web can be displayed. Using more or less than 256 colors requires other mechanisms to display, which will be discussed in the next section.

4. "Web-safe" and not "Web-safe" colors

The problem with making sure that someone is seeing your picture exactly as you intended it to be viewed, stems from the fact that all computers are different. Windows users see a different set of system colors than Macintosh users who in turn see different system colors than those on Unix-based X Window System computers. The colors are frequently very similar, but differ enough to cause some difficulties.

The developers who wrote the first World Wide Web browsers realized that palettes would be different on various computers. All of the most common browsers use the same basic color palette, called the browser CLUT. Since they had to design this CLUT for use on more than one platform, the programmers left room so that the various operating systems could insert their basic system palette into this CLUT without having to lose any of the colors. The standard Web browser CLUT uses 216 colors. Those colors are the same on every Browser, Platform and Operating System and these colors are often called Browser Safe Colors, or Browser Independent. The colors other than the 216 colors are not browser safe colors. These 216 colors come from a "6x6x6" mathematical formula, calculating the 216 colors using six values each of red, green, and blue (6x6x6=216). The other 40 colors in the CLUT are left undefined so that the operating system can use its system palette.

The browser CLUT can only work if the operating system leaves at least 216 colors available. Sometimes this doesn't happen. Such is often the case with the various flavors of Unix that have more than one Window open at one time. Unlike Windows and Macintosh/OS computers, Unix allows individual programs to reserve separate sections of the palette. This means that the number of colors available varies depending on which programs are running at the time. If there are less than 216 colors, the browser will have to limit its CLUT to a smaller palette (usually using similar mathematical formulas to find the best fit for the number of colors available, such as a 125-color 5x5x5 CLUT or a 64-color 4x4x4 CLUT.)

To render colors which are not "Wed-safe", there are three ways to do it. If the image does not have any reduction color method the browser will use the nearest color of the system palette. The other is reducing colors by Dithering the image and the last one is to reduce the colors by Banding the image. Certainly by using the nearest color, quality is lost because there is no certain way of knowing what colors the browser is going to select for this process.

Dithering method is used as seen for decreasing the number of colors contained within an image. What really happens is, using a 256 color palette different color pixels are placed in the image so that it can simulate a color that does not exist. Of course the quality is not the same as before. A much better dithering method is using an adaptive palette. Adaptive palettes are created from colors relative to the image. That is from colors that exist in the image. In most cases the adaptive-based dithering method gives the best results concerning quality.

Banding is a method of reducing colors without dithering. The reduction is done by creating areas of solid color. This is a good method for images and graphics in general that contain large areas of solid color.

5. Mechanisms for speeding up graphics display on the Web.

Other than compression, minimizing image sizes and color reduction can be used to speed up Web graphics display. The way to minimize image sizes has been presented in section 2. Color reduction has old and new ways.

The Old way includes Flat or Synthetic Images, Continuous Tone Images and Histogram Influencing.

To map the non-dithering CLUT to an existing image, first fill all of the flat-color areas with colors from the 216-color palette. Any other colors in the non-flat areas of your image will dither on some systems, which is usually fine. Now convert the image from RGB to indexed colors and load the special 216-color CLUT (Mode/Indexed Colors/Custom Palette/Dither = None). Photoshop will reduce the image to 216 colors and map the image to this palette. Of course, during this color mapping some posterization may occur.

Most full-color photographic-like images do not reduce well with the 216-color CLUT. Using an adaptive palette with diffusion dithering generally gives the best results. Adaptive palettes are tuned to the colors present in the image, and result in less dithering and a more pleasing image. On 24-bit systems, 8-bit adaptive-palette images look nearly 24-bit in quality. However, on 8-bit machines browsers will remap the adaptive palette into the client's system palette. Usually the results are acceptable.

Using an adaptive palette with diffusion dithering generally gives the best results. Adaptive palettes are tuned to the colors present in the image, and result in less dithering and a more pleasing image. On 24-bit systems, 8-bit adaptive-palette images look nearly 24-bit in quality. However, on 8-bit machines browsers will remap the adaptive palette into the client's system palette. Usually the results are acceptable.

When you index to an adaptive palette Photoshop creates a histogram of the colors in your image, ranked by popularity. Photoshop uses the histogram to determine how much weight to give certain colors in the resulting palette. For simple images, this method works well. For more complex images, Photoshop's choice of color palette may be less than optimal. A little known feature of Photoshop is that you can influence the histogram by using selections. Photoshop normally calculates the histogram using the entire image, but for more control you can select the part(s) of the image that contain the colors you want included in the palette. Photoshop weights the conversion towards the colors in the active selection(s). This technique can be used to create the absolute minimum palette for a given image.

The new way is using Human Visual System's HVS Color. HVS is a patented, psychovisual algorithm that models the way the eye perceives and masks color. HVS represents graphics in a color space that more closely matches how our eye's actually perceive color. The net effect is less discernible color banding at lower bit depths, with no dithering.

6. How to design Web graphics

To sum up, the following is the tips to create your Web graphics with Photoshop:

1. Minimize dimension and maximize crop.

2. Minimize the initial number of colors.

3. Choose your colors from "Web-safe" colors.

4. Use aliased sans-serif fonts like Geneva, Chicago. Anti-aliasing increases the number of colors.

5. Use flat, horizontal areas of color: avoid gradations of color, especially large, gradual ones which are radial or horizontal, and have a large shift in hue and/or tint.

6. Use histogram influencing to optimize and minimize palettes.

7.Reduce the resolution to 72 dpi as your last step.

8. Turn off icons/Photoshop 2.0 support for file sizes that match the Web. Group multiple graphics into one CLUT.

9. Use multiple small graphics and/or backgrounds instead of one large graphic.

10. Choose proper Web graphics format to display on the Web.

References

http://www.widearea.co.uk/designer/compress.html

http://www.lynda.com/hex.html

http://www.netscapeworld.com/common/nw.color.html

http://www.aphids.com/susan/imhelp/thumb.html

http://www.netscapeworld.com/common/nw.color.html

http://www.hogent.be/chroma/ditherig.html

GeoZui3D Data File Format Specifications

CONTENTS

Object Formats

GZX Objects

Composite Object File Formats

Non-Object Formats

Texture Map Formats

OBJECT FORMATS

General Object Format

All object formats native to GeoZui3D have a similar structure. Elements in this format that act unchanged across objects only appear in this section (to reduce clutter and repetition). The structure is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN < *Data-Type*> < *Version-Number*>	Each file format is designated by a *Data-Type*. The *Version-Number* indicates the version of the general format in the digit before the first decimal, and the version of the format for the specific *Data-Type* after the first decimal.
[Header Contents]	Each *Data-Type* has its own header format. Each line in the header has a keyword optionally followed by parameters. Each keyword consists only of capital letters, numbers, and underscores. The contents of a header line following a percent sign "%" will be treated as a comment and ignored by GeoZui3D. Blank lines in the header are also ignored.
[**DEGRADATION** {**OFF** | **ON** | < *Object-specific-keyword*>}]	Specifies whether or not to degrade the object. If "OFF", minimal or no degradation is done. If "ON", degradation occurs according to the default for the object. Various objects define additional possible types of degradation. If this line is not present, defaults to the default for the object (same as "ON").
[**REVERSED_XY**]	Using this keyword, objects support the reversal of X and Y coordinates, allowing Y-coordinates to appear first. This reversal is especially useful when information is given in latitude-longitude pairs, but the format expects x- and y- coordinates. The reversal applies to LOCAL_ORIGIN and PROJECT...CENTER coordinates as well.
[**PROJECT** {**AUTO** | < *Projection-Name*>} [**CENTER** <*Longitude*> < *Latitude*> < *Height*>] [< *arg*>]*]	Specifies that the data in this file is not in UTM, and should be projected first (currently, only projections from latitude-longitudes are supported). All objects that support the PROJECT keyword take at least one argument (Projection-Name) and an optional CENTER clause. The Projection-Name is one of the projections available in the .Projection hierarchy (accessible through the "gz projection list" command). If AUTO is specified, then the currently active projection in GeoZui3D is used, or if no projection is present then an appropriate UTM zone is automatically determined and made the active GeoZui3D projection. The CENTER clause is optional, and specifies what a point to be used to select an AUTO projection, as well as what to use as the internal reference point from which the remaining points will be offset. Further arguments may be present in some formats, such as the GUTM format. The PROJECT line replaces the LOCAL_ORIGIN line as indicating the end of the header, and should not appear in the header for files that contain data already in UTM coordinates.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*> [**IN** < *Projection-Name*>]	The end of each header is designated by the local origin of the object given in UTM coordinates (meters). The height is with respect to "sea level": a negative height specifies an origin below sea-level, and a positive height specifies an origin above sea-level. If a Projection-Name is given, GeoZui3D attempts to set this as the current projection. If another projection is already in place, however, GeoZui3D produces a warning and loads the object as if the LOCAL_ORIGIN coordiates are relative to the current projection already in place.
[Data]	Each *Data-Type* has its own data format, although some have no data whatsoever.

GUTM (.gutm) Format

The Gridded UTM file type is intended for bringing in regularly gridded bathymetry, or other regularly gridded surfaces. It allows holes and oddly shaped boundaries, but uses a rectangular bounding grid area for its specification. The structure of the GUTM file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN GUTM < *Version-Number*>	The current version is 1.1. Features that were not present in version 1.0 have the number of the first version they appear in at the end of the description.
COLS_ROWS < *Column-Count*> < *Row-Count*>	Specifies the number of points in each column and row of the bounding grid.
CELLSIZE {< *Cell-Size*> | **AUTO**}	Specifies the distance between grid points in meters. AUTO can only be specified if a PROJECT line is present, and is the default.
SHAPE { **FULLGRID** | **PARTIAL** }	Specifies whether the Strips that follow specify the start and end of a line (PARTIAL) or not (FULLGRID). PARTIAL is the default if no SHAPE is given.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the Southwestern corner of the bounding grid.
[**PROJECT** {**AUTO** | < *Projection-Name*>} [**CENTER** <*Longitude*> < *Latitude*> < *Height*>] < *Conversion-Method*> [<*arg*>]*]	Specifies that the data in this file is to be converted to GUTM format from another gridded format, and that the data following the PROJECT line is in the Conversion Grid Format. The Projection-Name is one of the projections available in the .Projection hierarchy (accessible through the "gz projection list" command), and the Conversion-Method is one of the methods available in the .Conversion hierarchy (accessible through the "gz address Conversion list" command). If AUTO is specified, then the currently active projection in GeoZui3D is used, or if no projection is present then an appropriate UTM zone is automatically determined. If a CENTER clause is present, it is *not* used to give an internal (x, y, z) reference point. The Longitude is used to determine the projection if AUTO is specified, and the Height is used as the base of the GUTM (the Latitude is for all purposes ignored). Further arguments after the Conversion-Method are passed to the conversion method, as are information from COLS_ROWS, CELLSIZE, and SHAPE parameters. The PROJECT line replaces the LOCAL_ORIGIN line as indicating the end of the GUTM header, and should not appear in the header for normal GUTM files that contain triangle-strip records. (New in 1.1)
{< *Triangle-Strip*>}* | < *Conversion-Grid*>	If no PROJECT line is in the header, Triangle-Strips are specified from South to North. There must be as many strips as there are rows. The format of the strips are defined in the following table. If a PROJECT line is in the header, a full Conversion Grid Format is expected.

Triangle Strip Records

The definition of < *Triangle-Strip*> for the GUTM file format is as follows:

[< *Start-Column*> < *End-Column+1*>]	Indicates the starting and ending columns in the current strip (row) from West to East. The GUTM is assumed to be empty outside this range, but data is expected for every point within this range (excluding the column of End-Column+1). This line is only present if "PARTIAL" is chosen for the SHAPE.
{< *Height-Value*>}*	The height for the current row and column. Any values less than or equal to 0.0 are assumed to be holes. The number of heights in a strip is equal to < *End-Column*> - < *Start-Column*> + 1.

Example

The following example shows a 5x4 bounding grid with a surface defined at the points represented in the following schematic. The bounding grid area is 100x80, as determined by the cellsize and the dimensions of the grid.

```
***
 ***
 ****
** **
```

```
BEGIN GUTM 1.0
COLS_ROWS 5 4
CELLSIZE 20.000000
LOCAL_ORIGIN 483785.000000 5297590.000000 -2700.000000
0 5
204.495850
206.458008
0.000000
205.375000
204.011719
1 5
216.243408
225.353516
231.915527
233.036865
1 4
228.806641
240.465088
233.954102
0 3
231.991943
237.942139
233.266113
```

Images (.img) Format

The Images file type is intended for displaying 2D jpeg files as flat georeferenced images in a 3D scene. There are three formats for specifying the placement of these images, as described below. The structure of the Images file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN IMAGEFILE < *Version-Number*>	Currently, there is only one version (1.0).
{**ROV** | **BILLBOARD** | **HOVER**}	Specifies the format for the Image-Records that follow.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the origin from which image locations are specified.
{< *Image-Record*>}*	Specified in any order. There can be any number of Image-Records, as there is no line specifying how many to expect. The format of the Image-Records are defined in the following table.

Images Format Options

A < *Image-Record*> within the Images file format takes one of the following formats (all location values are specified in meters from the LOCAL_ORIGIN):

<X> <Y> <Z> < Heading> < Filename> < Time>	"ROV" Option. The x-y-z position indicates the location of the "camera" that took the picture. Default values for the size and distance of the images from the camera are assumed. Heading is given in degrees, and indicates the direction the camera was facing (in absolute terms). The filename is the name of the jpeg to be shown. The time field is currently unused, but a value must be present here. The format expected is HH:MM:SS, although any format that does not contain whitespace would be acceptable at this time.
<X> <Y> <Z> < Heading> < Height> < Width> < Filename>	"BILLBOARD" Option. The x-y-z position indicates the location of the "camera" that took the picture. Heading is given in degrees, and indicates the direction the camera was facing (in absolute terms). Height and Width parameters specify what size the image should have when displayed. The distance from the camera is assumed to be 1.5 * Width. The filename is the name of the jpeg to be shown.
<X1> <Y1> <X2> <Y2> <X3> <Y3> <X4> <Y4> <Z> < Filename>	"HOVER" Option. The X and Y positions specify the four corners of the image, all using the same Z. This has the effect of generating an image that is parallel with the sea-plane. The filename is the name of the jpeg to be shown.

Examples

The following example shows a 10-image ROV-style file.

```
#        X         Y         Z   HEADING  FILENAME      TIME
BEGIN IMAGEFILE 1.0
ROV
LOCAL_ORIGIN 493419   5312806     -2170
     -16.4      62.2       -13     115.4    R5091.jpg          14:43:37
     -17.3      64.2       -13       115    R5092.jpg          14:46:4
     -14.1      63.4     -13.1     118.7    R5093.jpg          14:46:19
     -18.5        70       -13     115.2    R5094.jpg          14:46:45
     -17.9      57.9       -13     107.6    R5095.jpg          14:48:46
     -24.5      56.8     -13.1      56.9    R5096.jpg          14:52:6
     -23.9      66.7     -10.7     174.7    R5097.jpg          14:55:8
     -36.8      15.6     -14.9      65.5    R5098.jpg          15:9:36
      -1.7        40      -9.5        81    R5099.jpg          15:11:38
      43.3      41.1       1.8     285.4    R50910.jpg          15:24:21
```

The following example shows a 6-image HOVER-style file.

```
BEGIN IMAGEFILE 1.0
HOVER
LOCAL_ORIGIN 356000 4772000 0
   5324.41 -2547.43 5324.41 -2472.13 5399.71 -2472.13 5399.71 -2547.43 14 310_103_1650_3_m.jpg
   5301.3 -2769.38 5301.3 -2686.48 5384.2 -2686.48 5384.2 -2769.38 15.4 310_103_1650_4_m.jpg
   5273.23 -1493.46 5273.23 -1386.56 5380.13 -1386.56 5380.13 -1493.46 11.6 310_103_1650_5_m.jpg
   5056.23 -1817.33 5056.23 -1732.03 5141.53 -1732.03 5141.53 -1817.33 17 310_105_1704_1_m.jpg
   4988.94 -1846.31 4988.94 -1759.81 5075.44 -1759.81 5075.44 -1846.31 19.4 310_105_1704_2_m.jpg
   5072.8 -1887.51 5072.8 -1799.11 5161.2 -1799.11 5161.2 -1887.51 17.3 310_105_1704_3_m.jpg
```

Plane (.plane) Format

The Plane file type generates a finite, filled or hollow grid parallel to the ground. The structure of the Plane file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN PLANE < *Version-Number*>	Currently, there is only one version (1.0).
BACKGROUND {**ON** | **OFF**}	Specifies whether the grid is filled ("ON") or hollow ("OFF")
BGCOLOR < *Red-Byte*> < *Green-Byte*> < *Blue-Byte*>	Specifies the color of the background when it is on (even if it is off, this field is required anyway). Each color component is specified as a decimal integer between 0 and 255.
GCOLOR < *Red-Byte*> < *Green-Byte*> < *Blue-Byte*>	Specifies the color of the grid itself. Each color component is specified as a decimal integer between 0 and 255.
CELLSIZE < *X-Size*> < *Y-Size*>	Specifies the size of a grid rectangle (might not be square) in meters.
CELLCOUNT < *X-Count*> < *Y-Count*>	Specifies the number of grid rectangles in each direction. Must be an integer.
GRIDOFFSET < *Height*>	Specifies the distance between the background and the grid, when the background is on (if it is off, this field is required anyway). The offset is always added to the LOCAL_ORIGIN to determine the location of the plane; the background is always at the height of the LOCAL_ORIGIN. The offset is given in meters.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the center of the background plane.
	The Plane format has no data after the header.

Example

The following example shows a sea-level plane with a blue background and a white grid. The grid is 5km x 6km, with 100m x 100m squares.

```
BEGIN PLANE 1.0
BACKGROUND ON
BGCOLOR 50 50 250
GCOLOR 255 255 255
CELLSIZE 100 100
CELLCOUNT 50 60
GRIDOFFSET 1
LOCAL_ORIGIN 493410.000000 5312810.000000 0.000000
```

Points (.point) Format

The Points file type is intended for representing points of interest within a scene. Each point of interest is represented by a cone and a 50-meter line dropped from the cone to help indicate x-y position. The shape, size, and color of each point can be changed (within limits), and text can optionally be associated with each point. The structure of the Points file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN POINTS < *Version-Number*>	Currently, there is only one version (1.0).
N_RECORDS < *Point-Count*>	Specifies the number of point records to follow
{**XYZC** | **XYZDC** | **XYZDCT**}	Specifies one of three formats for the point records to follow. All of them expect x-y-z coordinate triples. "XYZC" expects color specificaiton, "XYZDC" expects cone diameter in addition to color, and "XYZDCT" expects diameter, color, and text.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the origin from which points are specified.
{< *Point*>}*	Specified in any order. There must be as many points as specified in the "N_RECORDS" line. The format of the points are defined in the following table.

Point Format Options

A < *Point*> within the Points file format takes one of the following formats (all x-y-z triples are specified in meters from the origin, and supported colors are listed in the next subsection):

<*X*> <*Y*> <*Z*> < *Color*>	"XYZC" Option.
<*X*> <*Y*> <*Z*> < *Diameter*> < *Color*>	"XYZDC" Option. The *Diameter* is actually a (multiplicative) scale factor on the entire point representation.
<*X*> <*Y*> <*Z*> < *Diameter*> < *Color*> "< *Text*>"	"XYZDCT" Option. The *Diameter* is actually a (multiplicative) scale factor onthe entire point representation. Text must appear between double-quotes, and may not contain double-quotes within it. Newlines and other characters are acceptable within the double-quotes.

Supported Colors

A	Auburn (brown)
B	Blue
C	Cyan
D	Dark Green
E	Elephant Grey
G	Green (bright)
R	Red
Y	Yellow

Example

The following example shows a 7-point object, with diameter and text fields included.

```
BEGIN POINTS 1.0
N_RECORDS 7
XYZDCT
LOCAL_ORIGIN 493410.000000 5312810.000000 -2140.000000
0.500000 -59.400002 38.700001 1.0  B "Blue Point"
4.600000 -28.900000 42.099998 1.0  A "Brown Point"
4.600000 -28.900000 42.599998 3.0  B "Big Blue Point"
10.000000 -2.000000 41.500000 1.0  C "Multi-line
Cyan Point
With three lines of text."
10.000000 -2.000000 42.000000 1.0  R "Red Point"
17.900000 8.900000 41.900002  1.0  Y "Yellow Point"
31.200001 28.600000 40.900002 1.0  E "Grey Point"
```

Tube (.tube) Format

The Tube file type is intended for representing 3D paths as a tube. Tubes can be rendered as given, or with automatically generated splines. The structure of the Tube file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN TUBE < *Version-Number*>	Currently, there is only one version (1.0).
N_RECORDS < *Point-Count*>	Specifies the number of point records to follow
{**SPLINE** | **NO_SPLINE**}	Specifies whether or not a spline is automatically generated. Currently, the "SPLINE" option can only be used with the "XYZ 0" below.
DIA < *Diameter*>	Specifies the diameter of the tube.
COLOR < *Red-1*> < *Green-1*> < *Blue-1*> < *Red-2*> < *Green-2*> < *Blue-2*>	Specifies two colors for striping the tube, or for using in choosing blended colors.
XYZ < *Num-Params*>	Specifies the number of parameters for the tube. x-y-z triples are always present. If *Num-Params* > 1, then floats between 0 and 1 are expected (indicating a blend of the colors specified above). If *Num-Params* > 2, then diameter multipliers are also expected (multiplied by the base diameter given above). If *Num-Params* is greater than zero, "NO_SPLINE" must be specified (for now, at least).
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the origin from which tube points are specified.
{< *Point-Record*>}*	Specified in order from first to last. There must be as many point-records as specified in the "N_RECORDS" line. The format of the point-records are defined in the following table.

Tube Format Options

A < *Point-Record*> within the Tube file format takes one of the following formats (all x-y-z triples are specified in meters from the origin):

<*X*> <*Y*> <*Z*>	"XYZ 0" Option.
<*X*> <*Y*> <*Z*> < *Color-Blend*>	"XYZ 1" Option. The *Color-Blend* number is a float between 0 and 1. If *a* is the first color specified in the "COLOR" header attribute, *b* is the second color, and *C* is the value of the *Color-Blend* for this particular record, then the resulting color is (1 - *C*)*a* + *Cb*.
<*X*> <*Y*> <*Z*> < *Color-Blend*> < *Size-Factor*>	"XYZ 2" Option. The *Color-Blend* is described above in the "XYZ 1" option. The *Size-Factor* is a (multiplicative) factor on diameter of the tube at the current point.

Examples

The following example shows a 10-point tube, with changing diameters and colors

```
BEGIN TUBE 1.0
N_RECORDS 10
NO_SPLINE
DIA 2.0
COLOR 0.7 0.7 0.7 0.6 0.85 0.85
XYZ 2
LOCAL_ORIGIN 493400.000000 5312810.000000 -2100.000000
43.200001 73.000000 -64.400002 0.0 0.5
66.500000 170.000000 -64.500000 0.2 0.5
127.199997 152.000000 -56.299999 0.4 0.50
103.000000 129.000000 -41.599998 0.6 1.0
83.599998 93.000000 -50.400002 0.8 1.0
63.000000 67.000000 -51.200001 1.0 1.0
49.599998 39.000000 -51.799999 0.8 2.0
32.500000 11.000000 -45.700001 0.6 2.0
16.600000 -11.000000 -57.000000 0.4 1.0
4.300000 -38.000000 -59.200001 0.2 1.0
```

The following example shows the same 10-point tube as a spline (without colors or sizes given).

```
BEGIN TUBE 1.0
N_RECORDS 10
SPLINE
DIA 2.0
```

```
COLOR 0.7 0.7 0.7 0.6 0.85 0.85
XYZ 0
LOCAL_ORIGIN 493400.000000 5312810.000000 -2100.000000
43.200001 73.000000 -64.400002
66.500000 170.000000 -64.500000
127.199997 152.000000 -56.299999
103.000000 129.000000 -41.599998
83.599998 93.000000 -50.400002
63.000000 67.000000 -51.200001
49.599998 39.000000 -51.799999
32.500000 11.000000 -45.700001
16.600000 -11.000000 -57.000000
4.300000 -38.000000 -59.200001
```

Geographic Data Points (.gdp) Format

The Geographic Data Points file type is intended for representing points that have scalar values associated with them. The format is designed to make it very easy to cut and paste data from another format into this format. The structure of the Points file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN DATA_POINTS < *Version-Number*>	Currently, there is only one version (1.0).
N_RECORDS < *Point-Count*>	Specifies the number of Point-Records to follow
N_COLUMNS < *Column-Count*>	Specifies how many whitespace delineated data items to expect in each Point-Record.
{**XYZ** | **XYZ_COLUMNS** < *X-Column*> < *Y-Column*> < *Z-Column*>}	If "XYZ", specifies that the first three columns should be used for x-y-z triples. If "XYZ_COLUMNS", specifies which column contains the coordinate data for each dimension. Each column number must be greater than or equal to 1 (the first column) and less than or equal to N_COLUMNS.
COLOR { **ATTRIBUTE** < *Attrib-Column*> [AS_HEIGHT] | **RGB_COLUMNS** < *Red-Column*> < *Green-Column*> < *Blue-Column*> | **CODE** < *Index-Column*> | **CONSTANT** {**RGB** | **CODE**} < *Color-Info*> }	If "ATTRIBUTE", specifies which column contains the data attribute to be mapped to color. If AS_HEIGHT is specified, applies the proper georeference height offset to the attribute before mapping it to color. If "RGB_COLUMNS", specifies which columns contain the color data for each color dimension. Unused components are designated by a 0. If "CODE", specifies which column contains a character representing color (given in the table that follows). If "CONSTANT", indicates that the following color should be used for all points. Color-Info consists of three floating point numbers between 0 and 1 if RGB, or a supported code if CODE.
GLYPH {**POINT** | *Shape* | **COLUMN** < *Glyph-Column*>}	Specifies how to represent a point. If "POINT" is selected, then each point is represented as a scale-independent square, with a size in pixels determined by the SIZE option. Otherwise, Shape can be one of glyphs listed in the "Supported Glyphs" table below. If COLUMN is selected, then the following argument specifies which column contains the glyph specification for each point.
	Specifies whether or not a line is dropped

[**DROP_LINE** {**ON** | **OFF** | **FRAGILE** | **TRANSIENT**}]	from the point, to provide better depth-cueing. If specified as "FRAGILE", then the dropped line only appears in high-resolution drawing (it is "non-robust" with respect to interaction; it "breaks" when you move and recovers when you stop moving). If specified as "TRANSIENT", then the dropped line only appears in low-resolution drawing (it is only there during interaction). Default is FRAGILE.
SIZE {**CONSTANT** < *Size-Factor*> | **COLUMN** < *Size-Column*> < *Size-Factor*> | **COLUMNS** [X][Y][Z][C][K] < *Dim-Size-Column*>* [< *Size-Factor*>]}	Specifies a constant size, or which column(s) contains the size information for each data point. If the GLYPH selected is a point, the size is multiplied by Size-Factor to indicate the number of pixels, otherwise the size is multiplied by Size-Factor to scale the GLYPH selected. Size-Column should be specified only if COLUMN is present. If COLUMNS is specified, then what follows is a 1-, to 5-letter designation for the numbers that follow respectively. X, Y, and Z indicate that the associated number is the column (starting from 1) in which the size for that dimension is specified. C indicates that the associated number is the column in which a multiplier on every dimension can be found. K indicates a that the associated number is a constant (Size-Factor) for any unspecified dimensions, and a multiplier for the specified dimensions. COLUMNS size specification can only be used with non-point GLYPH's.
TEXT {**NONE** | **COLUMN** < *Text-Column*>}	If "NONE" is specified, then no text is expected. If "COLUMN" is specified, then the following argument specifies which column contains the text information for each data point. This text can be displayed in the Information window of the Objects panel in GeoZui3D. Text must appear between double-quotes, and may not contain double-quotes within it. Newlines and other characters are acceptable within the double-quotes. A Text-Column of 0 indicates that no text should be associated with any points.
DEGRADATION {**OFF** | **SIMPLIFY_GLYPH** | **NTH**}	Specifies how to degrade the collection of points. If "OFF", no degradation is done. If "NTH", degradation occurs by only displaying every nth point.
DATA COLUMN < *Data-Column*> < *Name*> [**AS_NUMBER** | **AS_STRING** | **AS_EXPONENTIAL**]	Specifies that some data column should be read in for each row, and exported as an attribute named *Name* (rather than being actively displayed in some way). "AS_NUMBER" indicates the data should be treated as integer or floating point data, while "AS_STRING" indicates the data should be treated as literal strings. "AS_EXPONENTIAL" indicates that a log (base 2) should be taken on the data, then treated as floating point data. Defaults to "AS_NUMBER".
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the origin from which points are specified.
{< *Point-Record*>}*	Specified in any order. There must be as many points as specified in the "N_RECORDS" line. The format of the points are defined by the settings specified above.

Supported Color Codes

A	Auburn (brown)
B	Blue
C	Cyan
D	Dark Green
E	Elephant Grey
G	Green (bright)
K	Black
M	Magenta
R	Red
W	White
Y	Yellow

Supported Glyphs

These are the supported glyphs for the Geographic Data Points format. In practice, only the first three letters of the name are looked at, so you could specify CONE instead of CON, for instance.

CON	Cone
CUB	Cube
CYL	Cylinder
DIA	Diamond
INV	Inverted Cone
SPH	Sphere
UCU	UCube (Unit Cube, width of 2.0)
UCY	UCylinder (Unit Cylinder, radius and height of 1.0)
USP	USphere (Unit Sphere, radius of 1.0)

Example

The following example creates 7-points, represented as cones with dropped lines, and with diameter and text fields included.

```
BEGIN DATA_POINTS 1.0
N_RECORDS 7   % Number of points
N_COLUMNS 6   % Number of columns per point
XYZ    % First 3 columns have XYZ data
COLOR CODE 5   % Column 5 has a color code
GLYPH CONE
DROP_LINE ON
SIZE COLUMN 4 1.0  % Column 4 has a scale factor, multiply it by 1.0
TEXT COLUMN 6   % Column 6 has informational text
DEGRADATION NTH
LOCAL_ORIGIN 493410.000000 5312810.000000 -2140.000000
0.500000 -59.400002 38.700001 1.0  B "Blue Point"
4.600000 -28.900000 42.099998 1.0  A "Brown Point"
4.600000 -28.900000 42.599998 3.0  B "Big Blue Point"
10.000000 -2.000000 41.500000 1.0  C "Multi-line
Cyan Point
With three lines of text."
10.000000 -2.000000 42.000000 1.0  R "Red Point"
17.900000 8.900000 41.900002  1.0  Y "Yellow Point"
31.200001 28.600000 40.900002 1.0  E "Grey Point"
```

Geographic Data Lines (.gdl) Format

The Geographic Data Lines file type is intended for representing lines that have scalar values associated with their vertices. The format is designed to make it easier to cut and paste data from another format into this format. The structure of the Lines file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN DATA_LINES < *Version-Number*>	Currently, there is only one version (1.0).
N_RECORDS < *Point-Count*>	Specifies the number of Line-Records to follow
N_COLUMNS < *Column-Count*>	Specifies how many whitespace delineated data items to expect in each line of a Line-Record.
{**XYZ** | **XYZ_COLUMNS** < *X-Column*> < *Y-Column*> < *Z-Column*>}	If "XYZ", specifies that the first three columns should be used for x-y-z triples. If "XYZ_COLUMNS", specifies which column contains the coordinate data for each dimension. Each column number must be greater than or equal to 1 (the first column) and less than or equal to N_COLUMNS.
COLOR { **ATTRIBUTE** < *Attrib-Column*> [AS_HEIGHT] | **RGB_COLUMNS** < *Red-Column*> < *Green-Column*> < *Blue-Column*> | **CODE** < *Index-Column*> | **PER_RECORD** {**ATTRIBUTE** | **RGB** | **CODE**} | **CONSTANT** {**RGB** | **CODE**} < *Color-Info*> }	If "ATTRIBUTE", specifies which column contains the data attribute to be mapped to color. If AS_HEIGHT is specified, applies the proper georeference height offset to the attribute before mapping it to color. If "RGB_COLUMNS", specifies which columns contain the color data for each color dimension. Unused components are designated by a 0. If "CODE", specifies which column contains a character representing color (given in the color table for .gdp's). If "PER_RECORD", indicates that the color will be given at the beginning of a record in one of these formats. If "CONSTANT", indicates that the following color should be used for all lines. Color-Info consists of three floating point numbers between 0 and 1 if RGB, or a supported code if CODE.
[**LINE** {**OFF** | **ON** | **FRAGILE** | **TRANSIENT**}]	Specifies whether to show a line or not. If specified as "FRAGILE", then the line only appears in high-resolution drawing (it is "non-robust" with respect to interaction; it "breaks" when you move and recovers when you stop moving). If specified as "TRANSIENT", then the line only appears in low-resolution drawing (it is only there during interaction). Default is ON.
[**TUBE** {**OFF** | **ON** | **FRAGILE** | **TRANSIENT**} [**FACETS** < *FacetCount*>]]	Specifies whether to show a tube or not. If specified as "FRAGILE", then the tube only appears in high-resolution drawing (it is "non-robust" with respect to interaction; it "breaks" when you move and recovers when you stop moving). If specified as "TRANSIENT", then the tube only appears in low-resolution drawing (it is only there during interaction). If FACETS is given, TUBE must not be OFF, and the FacetCount specifies how many rectangles to use to approximate a tube. Default is OFF.
SIZE {**PER_RECORD** | **CONSTANT** < *Size*>}	Specifies whether the size (width) for a line is given at the beginning of a Line-Record, or is constant and given here.

TEXT {**NONE** | **PER_RECORD**}	Specifies whether text is supplied at the beginning of a Line-Record or not.
[**DEGRADATION** {**OFF** | **NTH** | **FACETS** | **FACETS_NTH**}]	Specifies how to degrade the collection of points. If "OFF", no degradation is done. If "NTH", degradation occurs by only displaying every nth point. "FACETS" and "FACETS_NTH" are respective degradations of "OFF" and "NTH" that degrade a tube to have fewer facets. The default is FACETS_NTH.
DATA {**COLUMN** < *Data-Column*>| **PER_RECORD**} < *Name*> [**AS_NUMBER** | **AS_STRING** | **AS_EXPONENTIAL**]	Specifies that some data should be read in and exported as an attribute named *Name* (rather than being actively displayed in some way). If "COLUMNS" is specified, the data will be expected for every point. If "PER_RECORD" is specified, the data will be expected after all other fields in each "NEW" line (but before any DATA lines declared after this line). "AS_NUMBER" indicates the data should be treated as integer or floating point data, while "AS_STRING" indicates the data should be treated as literal strings. "AS_EXPONENTIAL" indicates that a log (base 2) should be taken on the data, then treated as floating point data. Defaults to "AS_NUMBER".
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the origin from which points are specified.
{< *Line-Record*>}*	Specified in any order. There must be as many Line-Records as specified in the "N_RECORDS" line. The format of the Line-Records are given in the folowing table.

Line-Records

A < *Line-Record*> within the Geographic Data Line file format takes the following format:

NEW < *Point-Count*> [< *Color*>] [< *Size*>] [< *Text*>]	Indicates the number of points in this line (>= 2). Any attributes that were designated as PER_RECORD in the header are listed in the order shown here. If COLOR was specified as RGB, three floats between 0.0 and 1.0 are expected. Size is either a scale-independent line width or a scale-dependent tube diameter. Text must appear between double-quotes, and may not contain double-quotes within it. Newlines and other characters are acceptable within the double-quotes.
{< *Point-Record*>}*	Specified in order of the line. There must be as many points as specified in the Point-Count above. The format of the points are defined by the settings specified in the header.

Example

The following example creates three lines, represented as tubes with lines running down the center, and with diameter and text fields included. Color is given independently for each point.

```
BEGIN DATA_LINES 1.0
N_RECORDS 3   % Three line records
N_COLUMNS 6   % 6 columns per line record
XYZ    % First three columns have XYZ
COLOR CODE 5   % Column 5 contains a color code
LINE ON
TUBE ON
SIZE PER_RECORD   % Expect a tube size in each record
TEXT PER_RECORD   % Expect informational text in each record
DEGRADATION OFF
LOCAL_ORIGIN 493410.000000 5312810.000000 -2140.000000
NEW 4 2.0 "2m Diameter Line, with multiline text,
 blue-green-yelow-auburn"
```

```
0.500000 -59.400002 38.700001 junk  B junk
4.600000 -28.900000 42.099998 junk  G more_junk
4.600000 -28.900000 42.599998 1.0   Y just_junk
10.000000 -2.000000 41.500000 x     A 6.2
NEW 2 1.0 "Thin grey tube"
10.000000 -2.000000 42.000000 1.0   E 16
17.900000 8.900000 41.900002  1.0   E hut_hut
NEW 2 5.0 "Fat grey tube semi-connected to the thin grey tube"
17.900000 8.900000 41.900002  1.0   E hut_hut
31.200001 28.600000 40.900002 1.0   E 1594
```

Geographic Data Triangles (.gdt) Format

The Geographic Data Triangles file type is intended for representing triangles and surfaces that have scalar values associated with their vertices. The format is designed to make it easier to cut and paste data from another format into this format. The structure of the Geographic Data Triangles file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN DATA_TRIANGLES < *Version-Number*>	Currently, there is only one version (1.0).
N_RECORDS < *Point-Count*>	Specifies the number of Triangle-Records to follow
N_COLUMNS < *Column-Count*>	Specifies how many whitespace delineated data items to expect in each line of a Triangle-Record.
{**XYZ** | **XYZ_COLUMNS** < *X-Column*> < *Y-Column*> < *Z-Column*>}	If "XYZ", specifies that the first three columns should be used for x-y-z triples. If "XYZ_COLUMNS", specifies which column contains the coordinate data for each dimension. Each column number must be greater than or equal to 1 (the first column) and less than or equal to N_COLUMNS.
COLOR { **ATTRIBUTE** < *Attrib-Column*> [AS_HEIGHT] | **RGB_COLUMNS** < *Red-Column*> < *Green-Column*> < *Blue-Column*> | **CODE** < *Index-Column*> | **TEXTURE** | **PER_RECORD** {**ATTRIBUTE** | **RGB** | **CODE**} | **CONSTANT** {**RGB** | **CODE**} < *Color-Info*> }	If "ATTRIBUTE", specifies which column contains the data attribute to be mapped to color. If AS_HEIGHT is specified, applies the proper georeference height offset to the attribute before mapping it to color. If "RGB_COLUMNS", specifies which columns contain the color data for each color dimension. Unused components are designated by a 0. If "CODE", specifies which column contains a character representing color (given in the color table for .gdp's). If "TEXTURE", the user will assign a geo-referenced texture at run-time. If "PER_RECORD", indicates that the color will be given at the beginning of a record in one of these formats. If "CONSTANT", indicates that the following color should be used for all triangle strips. Color-Info consists of three floating point numbers between 0 and 1 if RGB, or a supported code if CODE.
	Specifies whether to render hollow triangles or fill them in. If specified as "FRAGILE", then triangles are only filled in during high-resolution drawing (it is "non-robust" with

[**FILL** {**PER_RECORD**	**OFF**	**ON**	**FRAGILE**	**TRANSIENT**}]	respect to interaction; it "breaks" when you move and recovers when you stop moving). If specified as "TRANSIENT", then the triangles are filled in only during low-resolution drawing (it is only there during interaction). Default is ON.
NORMALS {**PER_RECORD**	**RIGHT_HANDED**	**LEFT_HANDED**	**UP**	**DOWN**	**XYZ_COLUMNS** < *X-Column*> < *Y-Column*> < *Z-Column*> [**SCALE_Z** < *Z-Factor*>]}	Specifies how the normals should be calculated for the purposes of lighting. If XYZ_COLUMNS, specifies which column contains the vector normal component for each respective dimension. Each column number must be greater than or equal to 1 (the first column) and less than or equal to N_COLUMNS. If a SCALE_Z is present, then the Z-factor that follows is multiplied by each normal's z-component, which can be useful when normals are calculated in lat-long (normals do not get transformed by projections). If not XYZ_COLUMNS, the normals are automatically generated by taking the cross-product of the lines between the first two points of a triangle and the second two points, modified as follows (If PER_RECORD, each triangle strip will define its own option from RIGHT_HANDED, LEFT_HANDED, UP, or DOWN). RIGHT_HANDED and LEFT_HANDED indicate whether the right-hand rule (standard) or the left-hand rule (multiply each component of the cross product result by -1) should be used to determine the direction of the normal for the first triangle--after this, the rule is alternated with each successive triangle so that a "regular" strip has all normals pointing in the same z-direction. UP or DOWN indicates that all normals should be forced to point in the same z-direction (either UP or DOWN).
SIZE {**PER_RECORD**	**CONSTANT** < *Size*>}	Specifies whether the size (width) for lines are given at the beginning of a Triangle-Record, or is constant and given here. This specifies the line thickness.
TEXT {**NONE**	**PER_RECORD**}	Specifies whether text is supplied at the beginning of a Triangle-Record or not.
DATA {**COLUMN** < *Data-Column*>	**PER_RECORD**} < *Name*> [**AS_NUMBER**	**AS_STRING**	**AS_EXPONENTIAL**]	Specifies that some data should be read in and exported as an attribute named *Name* (rather than being actively displayed in some way). If "COLUMNS" is specified, the data will be expected for every point. If "PER_RECORD" is specified, the data will be expected after all other fields in each "NEW" line (but before any DATA lines declared after this line). "AS_NUMBER" indicates the data should be treated as integer or floating point data, while "AS_STRING" indicates the data should be treated as literal strings. "AS_EXPONENTIAL" indicates that a log (base 2) should be taken on the data, then treated as floating point data. Defaults to "AS_NUMBER". *Not actively implemented yet!*
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the origin from which points are specified.
{< *Triangle-Record*>}*	Specified in any order. There must be as many Triangle-Records as specified in the "N_RECORDS" line. The format of the Triangle-Records are given in the folowing table.

Triangle-Records

A < *Triangle-Record*> within the Geographic Data Triangles file format takes the following format:

NEW < *Point-Count*> [< *Color*>] [< *Fill-Mode*>}] [< *Normal-Rule*>] [< *Size*>] [< *Text*>]	Indicates the number of points in this triangle strip. Any attributes that were designated as PER_RECORD in the header are listed in the order shown here. If COLOR was specified as RGB, three floats between 0.0 and 1.0 are expected. The Fill-Mode can be one of ON, OFF, FRAGILE, or TRANSIENT, with the effect described for the FILL parameter above. The Normal-Rule can be one of RIGHT_HANDED, LEFT_HANDED, UP, or DOWN, with the effect described for the NORMALS parameter above. Size indicates a scale-independent line width. Text must appear between double-quotes, and may not contain double-quotes within it. Newlines and other characters are acceptable within the double-quotes.
{< *Point-Record*>}*	Specified as in GL_TRIANGLE_STRIP. In othe words, for a list of *n* points, the first triangle is formed by p0, p1, and p2; the second triangle is formed by p1, p2, and p3; all the way up to p(*n*-2), p(*n*-1), p*n*. There must be as many points as specified in the Point-Count above. The format of the points are defined by the settings specified in the header.

Example

The following example creates three triangle stips using a texture supplied through GeoZui3D at runtime.

```
BEGIN DATA_TRIANGLES 1.0
N_RECORDS 3   % Expect 3 Triangle records
N_COLUMNS 6   % Each record has 6 columns
XYZ    % First three columns have XYZ
COLOR TEXTURE   % Use a texture for coloring
FILL PER_RECORD   % Expect a FILL designation in each record
SIZE CONSTANT 1.0   % Keep line widths at 1.0
TEXT PER_RECORD   % Expect information text in each record
DEGRADATION OFF
LOCAL_ORIGIN 493410.000000 5312810.000000 -2140.000000
NEW 3 FRAGILE "Filed triangle"
0.500000 -59.400002 38.700001 junk  junk junk
4.600000 -28.900000 42.099998 junk  junk more_junk
-4.600000 28.900000 42.599998 1.0   Y just_junk
NEW 5 FRAGILE "Mesh of three triangles"
10.000000 -2.000000 41.500000 x     Q 6.2
10.000000 2.000000 45.000000 1.0   E 16
17.900000 8.900000 41.900002  1.0   E hut_hut
17.900000 -8.900000 35.900002  1.0   E hut_hut
31.200001 28.600000 40.900002 1.0   E 1594
NEW 6 FRAGILE "Prism"
4.5  4.5  4.5  a b c
0.0  0.0  0.0  d e f
10.0 10.0 0.0  g h i
4.5  8.0  0.0  j k l
4.5  4.5  4.5  a b c
0.0  0.0  0.0  d e f
```

Coordinate Grid/Axis(.cga) Format

The Coordinate Grid/Axis file type generates a finite grid or set of axes, filled or hollow, in 1, 2, or 3 dimensions. The structure of the Coordinate

Grid/Axis file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN COORD < *Version-Number*>	Currently, there is only one version (1.0).
DIMENSIONS < *XYZ-String*> *{*< *Length*>*}*+	Specifies the dimensions of the grid/axis. XYZ-String can be one of the folowing: X, Y, Z, XY, XZ, YZ, or XYZ. There is one Length value for each character in the string, representing the length in the respective dimension.
REFERENCE {**LOCAL** | **GLOBAL**}	Specifies whether gridlines and ticks are displayed at local offsets or at global coordinate locations.
[**BACKGROUND** < *Num-Planes*> {< *Plane-String*>}*]	Specifies how many background planes are generated, and which ones they are. The Plane-Strings are selected from the following: XYMIN, XYMAX, XZMIN, XZMAX, YZMIN, YZMAX. The "MIN" designation means the plane should be at the minimum position of the orthogonal dimension, while "MAX" designates the maximum position. The BACKGROUND field is not required if no backgrounds are desired.
[**BGCOLOR** < *Red*> < *Green*> < *Blue*>]	Specifies the color of the backgrounds when they are on. Each color component is specified as a float between 0.0 and 1.0. The BGCOLOR field is not required.
LINECOLOR < *Red*> < *Green*> < *Blue*>	Specifies the color of the grid/axis lines. Each color component is specified as a float between 0.0 and 1.0.
[**BACKGROUND_OFFSET** < *Distance*>]	Specifies the distance between the background and the grid/axis, when a background is on. The offset is always subtracted from MIN planes and added to MAX planes. The offset is given in meters. The BACKGROUND_OFFSET field is not required.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the origin of the coordinate grid/axis. The origin corresponds to the MIN position in all three dimensions.
{[{FRAGILE | TRANSIENT}] < *Coord-Spec*>}*	Specified in any order. Determines where lines, axis ticks, and/or labels are to go. Optionally, FRAGILE or TRANSIENT may appear before each record. If FRAGILE is specified, then the entity only appears in high-resolution drawing (it is "non-robust" with respect to interaction; it "breaks" when you move and recovers when you stop moving). If TRANSIENT is specified, then the entity only appears in low-resolution drawing (it is only there during interaction). The format of the Coord-Specs are defined in the following table.

Coordinate Grid/Axis Format Options

A < *Coord-Spec*> within the Coordinate Grid/Axis file format takes one of several formats. Some options are present across formats, as listed:

< *Location**>	{ **MIN** | **MAX** | **MIN_MAX** | **THROUGH**}	The Location takes the value of MIN, THROUGH, MAX, or MIN_MAX, indicating where lines, ticks, or labels should appear in the dimension being specified. For instance, an axis drawn through the X dimension might be drawn at the local-"0" values for both Y and Z, which would require MIN to be chosen for each of these dimensions. If MAX were chosen for both, the axis would be drawn at the opposite edge of the bouding-box (specified in the DIMENSIONS field in the header). THROUGH indicates that a line, tick, or label should be repeated at a regular interval throughout the respective dimension.
< *Spacing-clause*>	{**SPACING_EVEN** < *Spacing*> | **SPACING_XYZ** < *X-Spacing*> < *Y-Spacing*> < *Z-Spacing*> | **SPACING_DYNAMIC** [< *Spacing-Factor*>] }	If the first token in the Spacing-clause is "SPACING_EVEN", then the following argument describes the distance between lines, ticks, or labels given in meters. If the token is "SPACING_XYZ", then three arguments must follow, each describing the distance between lines, ticks, or labels, given in meters for each respective dimension. If the first token is intstead "SPACING_DYNAMIC", then the following argument describes the factor at which new sets of ticks/axes are created as the environment is zoomed in and out. If no argument is given for SPACING_DYNAMIC, the Spacing-Factor defaults to 10.0.

The formats for the different types of Coord-Spec's are as follows:

LINES < *Direction*> < *Location1*> [< *Location2*>] [< *Spacing-clause*>]	Lines for grid markings, axis. Direction is given as a letter indicating which axis the line is parallel to, followed by a dash, followed by one or two letters indicating which axes the line should be replicated in. For instance, X-Y is a line parallel to the X axis, potentially replicated in the XY plane; while X-YZ is also parallel to the X axis, but is potentially replicated throughout space. The Location indicates where the line(s) should appear relative to the portion of the direction after the dash. If the portion of the direction after the dash is a single letter, then Location2 must be specified (otherwise it must not be specified), and indicates the location with respect to the missing dimension (othogonal to both in the Direction argument). If THROUGH is specified for any dimension, then a Spacing-clause is required.
{**TICKS** | **LABELS** | **TICKS_LABELS**} < *Direction*> < *Location1*> < *Location2*> < *Distance*> < *Spacing-clause*>	Tick marks and/or labels for axes. Direction is given as a letter indicating the dimension along which ticks and/or labels should be placed. Location1 and Location2 indicate where on the other two orthogonal axes the tick marks and/or labels should go. Location1 specifies either X or Y, and Location2 specifies either Y or Z, Direction != Location1 != Location2. LABELS are never really drawn THROUGH, but instead appear as if specified as MIN_MAX. < *Distance*> specifies how long the tick marks should be (in meters) and/or how far away the labels should be from the axis. When SPACING_DYNAMIC is chosen as the method of spacing, the meaning of Location and Distance information changes somewhat. The Distance gets scaled according to the scale of the level being drawn. For instance, if Distance is 2.0 and the Spacing-Factor is 10.0, then for a grid level with spacing of 100 meters, the length of a tick would be 20 meters. With LABELS, if Location is THROUGH, labels will appear on either end of each dynamic area, as if specified as MIN_MAX for each dynamic area.
GRID < *Direction*> < *Location*> < *Spacing-clause*>	Special case of LINES for simple grid creation. Direction is one of XY, XZ, YZ, or XYZ. The Location indicates where the plane should appear in the dimension orthogonal to the grid plane (or in the case of XYZ, it specifies the location for all 3 orthogonal directions at once).

Example

The following example shows a sea-level plane with a blue background and a white grid. The grid is 5km x 6km, with 100m x 100m squares.

```
BEGIN COORD 1.0
DIMENSIONS XY 5000 6000  % 5km east, 6km north
REFERENCE LOCAL   % (0,0) is at LOCAL_ORIGIN
BACKGROUND 1 XYMIN   % Draw one background plane
BGCOLOR 0.2 0.2 0.95   % Blue
LINECOLOR 1.0 1.0 1.0   % White
BACKGROUND_OFFSET 10.0  % Draw plane 10m below grid
LOCAL_ORIGIN 493410.000000 5312810.000000 0.000000
GRID XY MIN SPACING_EVEN 100.0
```

The following example shows a set of coordinate axes in all three dimensions. The gobal reference means that major ticks will not necessarily align with the axes as one might normally expect. The dynamic spacing on the X and Y axes generates different sized markers for each power of 10, based on the current viewing scale. For demonstration, the dynamic spacing on Z is set to 10.0 (not necessary since it defaults to 10 anyway).

```
BEGIN COORD 1.0
DIMENSIONS XYZ 5000 6000 3000  % 5km east, 6km north
REFERENCE GLOBAL   % (0,0) is at UTM origin
LINECOLOR 1.0 1.0 1.0   % White grid, numbers
LOCAL_ORIGIN 493410.000000 5312810.000000 -2048.000000
LINES X-Y MIN MIN
TICKS_LABELS X MIN MIN 10.0 SPACING_DYNAMIC
LINES Y-X MIN MIN
TICKS_LABELS Y MIN MIN 10.0 SPACING_DYNAMIC
LINES Z-X MIN MIN
TICKS_LABELS Z MIN MIN 10.0 SPACING_DYNAMIC 10.0
```

Curtain Plot Data (.ctn) Format

The Curtain Plot file type is intended for representing data that should be displayed as a 3D curtain plot. A curtain is constructed of columns of data, where each column is georeferenced in UTM coordinates, with a height locating the top of the column. Each column consists of any number of data-points equally spaced below the position of the top of the column. For good interactive performance, a maximum of up to 512 data points per column is suggested.

The structure of the Curtain Plot file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN CURTAIN < *Version-Number*>	Currently, there is only one version (1.0).
{ **CLUT** | **RGB** | **RGBA** }	"CLUT" specifies that each data point is represented in the file as a single pair of hex digits representing an index between 0 (00) and 255 (FF). The index value 0 is reserved to indicate an invalid reading. For rendering purposes, the index will be used to index an unassigned color lookup table. "RGB" specifies that each data point is represented by three pairs of hex digits representing the red, green, and blue components of the color to be used to represent the data point. "RGBA" specifies that each data point is represented by four pairs of hex digits representing the red, green, blue and alpha components of the data point's color. Alpha is an opacity value, where 255 (FF) means fully opaque and 0 (00) means fully transparent. In index (CLUT) mode, a standard GeoZui3D color lookup table must be assigned to the curtain, via the user interface. This lookup table is indexed by the curtain to determine colors (i.e. the height value in the lookup table is ignored, and each unique height entry in the table is given an index, starting at 0). Since index value 0 is reserved for an invalid reading, the color table should be constructed with its first entry being the one to represent bad data.
NCOLS < *Column-Count*>	Specifies the number of columns of data to follow
Z_STEP < *Depth-Increment*>	Specifies the distance in meters between the readings in a single column of data.
DEGRADATION {**OFF** | **NTH**}	Specifies how to degrade the curtain of points. If "OFF", no degradation is done. If "NTH", degradation occurs by creating the curtain using every nth point, potentially changing the shape and continuity of the curtain.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the origin from which data columns are specified.
{< X>< Y>< Z>< NVALS>{XX}}**	This is the column data. There is one line in the data file per column, in order by column number. There must be as many columns as specified in the "NCOLS" line. The X, Y and Z values are the coordinates of the top of the column. NVALS is the number of data values in the column. The data values XX themselves appear next, in order from the top of the column. These values are hex encoded as explained above and must not be comma or space separated.

Example

The following example shows a curtain plot file for a curtain consisting of only 8 columns, each with at most 15 data points, and with data coded by an index. The curtain reference height is 500 meters below sea level.

```
BEGIN CURTAIN 1.0
CLUT
NCOLS 6
Z_STEP 10.0
LOCAL_ORIGIN 483785.00 5297590.00 -500.00
501.23 -5935.73 10.54 15 58605d55514f464948474948414741
510.66 -6029.69 13.33 15 575151514c484d46463f4340474242
520.39 -6126.67  8.67 14 5e69625c545146403e41433f3b43
529.82 -6220.62  8.30 10 d5f5b51564a4b4644424
```

```
540.22 -6323.12  9.97  8 666d674f494a4342
551.37 -6431.33 11.41  8 535b584947464240
562.51 -6539.54 12.73 12 625950484c424a4442474443
574.02 -6651.24 11.05 15 656b6c6863655e5855514e40434543
```

Label (.label) Format

The Label file type creates bitmap text anchored at a specific 3D position, with the potential for different renderings at different scales. The header describes default settings for how the label should be rendered, while records after the LOCAL_ORIGIN line specify how the default should be modified to render the label within particular scale ranges. The structure of the Label file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN LABEL < *Version-Number*>	Currently, there is only one version (1.0).
N_RECORDS < *Record-count*>	Specifies the number of records that will follow for specifying the scale-dependent representations.
SCALE < *Min-Scale*> < *Max-Scale*>	Specifies the minimum and maximum scales at which the label should be drawn at all. If Max-Scale is 0, the label is always visible.
[**LABEL** {**ROBUST** | **OFF** | **FRAGILE** | **TRANSIENT** | **DEFAULT**}]	Specifies when a label is drawn with respect to interaction. If specified as "FRAGILE", then the label only appears in high-resolution drawing (it is "non-robust" with respect to interaction; it "breaks" when you move and recovers when you stop moving). If specified as "TRANSIENT", then the label only appears in low-resolution drawing (it is only there during interaction). ROBUST means it always shows, while OFF means it does not show. Default is ROBUST.
[**TEXT** < *Label-String*>]	Specifies what should be displayed as the label text. If Label-String includes whitespace, it should be enclosed in quotes.
[**FONT** < *Font-Type*>]	Specifies what font should be used in displaying the label. Valid Font-Types are STANDARD_9_BY_15, STANDARD_8_BY_13, TIMES_ROMAN_10, TIMES_ROMAN_24, HELVETICA_10, HELVETICA_12, and HELVETICA_18. The default is HELVETICA_10.
[**COLOR** < *Red*> < *Green*> < *Blue*> [< *Alpha*>]]	Specifies the foreground color of the text, as well as any lines or wireframe boxes drawn with the label. Valid values for each channel are between 0.0 and 1.0, inclusive. The default is all 1.0's (opaque white).
[**PRIORITY** < *Priority-Value*>]	The PRIORITY syntax is reserved for future use and currently has no effect.
[**LINE** {**ALWAYS** | **AUTO** | **DEFAULT** | **OFF**} [**ROBUST** | **OFF** | **FRAGILE** | **TRANSIENT** | **DEFAULT**]]	The LINE syntax is reserved for future use and currently has no effect.
[**BACKGROUND** [**COLOR** < *Red*> < *Green*> < *Blue*> <*Alpha*>] {**SHADOW** | **OUTLINED_BOX** | **BOX** | **OFF** | **DEFAULT**} [**ROBUST** | **OFF** | **FRAGILE** | **TRANSIENT** | **DEFAULT**]]	Specifies that a background of some sort should appear behind the label. If COLOR is present, all four channels must be present as well, with each in the range of 0.0 to 1.0. Otherwise, the background color defaults to all 0.0's (transparent black). The "background" can take the form of shadowed text, a solid box, or an outlined solid box. In the case of the outlined solid box, the outline color is determined by the toplevel COLOR line (not from this BACKGROUND line). If a robustness argument is given, "FRAGILE" indicates the background should only appear in high-resolution drawing (it is "non-robust" with respect to interaction; it "breaks" when you move and recovers when you stop moving). If specified as "TRANSIENT", then the background only appears in low-resolution drawing (it is only there during interaction). ROBUST means it always shows, while OFF means it does not show. The default if no BACKGROUND line is present is OFF, but the default if a BACKGROUND line is present and no robustness keyword is used is ROBUST.
[**SURFACE** < *GUTM-Name*>]	The SURFACE syntax is reserved for future use and currently has no effect.

[**INTEREST** {**POINT**	**REGION**} *x1 y1 z1* [*x2 y2 z2*]]	Specifies where the label should appear relative to the LOCAL_ORIGIN. If POINT is given, appears centered at (x1, y1, z1), relative to the LOCAL_ORIGIN. The REGION syntax and use of (x2, y2, z2) is reserved for future use and may cause instability if used currently.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The base coordinates of the label.
{< *Label-Record*>}*	Specified in order of increasing scale (magnification). There must be as many Label-Records as specified in the "N_RECORDS" line. The format of the Label-Records are given in the folowing table.

Label-Records

A < *Label-Record*> within the Label file format takes the following format (label records must appear in the order of increasing scale):

NEW_SCALE < *Min-Scale*>	Indicates the lower scale bound for the scale region in which the record applies.
{< *Label-Specification*>}*	Most of the syntax that is valid in the header is also valid in a Label-Record. LABEL, TEXT, FONT, COLOR, PRIORITY, LINE, BACKGROUND, SURFACE, and INTEREST syntax are all applicable here.

Examples

The following example shows a label that appears in a larger font between scales of 0.5 and 5.0 in the window that is rendering it, and a smaller font between scales of 5.0 and 50.0. When shown as the larger text, the text appears in cyan, with a grey box and a cyan outline, all centered at a position 100 meters from the LOCAL_ORIGIN in each dimension. When shown as the smaller text, it appears in light-blue with a white background box (no outline), 50 meteres lower than the larger text was.

```
BEGIN LABEL 1.0
N_RECORDS 1
SCALE 0.5 50.0
TEXT "This is a test label"
FONT STANDARD_9_BY_15
COLOR 0.0 1.0 1.0
INTEREST POINT 100 100 100
BACKGROUND COLOR 0.3 0.3 0.3 1.0 OUTLINED_BOX
LOCAL_ORIGIN 493278.000000 5312620.000000 -2500.000000
NEW_SCALE 5.0
COLOR 0.4 0.7 1.0
INTEREST POINT 100 100 50
FONT STANDARD_8_BY_13
BACKGROUND COLOR 1.0 1.0 1.0 1.0 BOX
```

The following example shows a label that appears at any scale up to a scale of 5. It appers at one location (in the center of Maine) up to a scale of .01, and then gets moved closer and closer to the New Hampshire border as the scale reaches .01, then .05, and finally .50 (it disappears at a scale of 5). This example is used in an exhibit that zooms in from a large region containing several states to an area on the border between New Hampshire and Maine. At the last level, the label changes to "Kittery, Maine" (this last line does not appear in the exhibit).

```
BEGIN LABEL 1.0
N_RECORDS 3
SCALE 0 5
TEXT "Maine"
FONT HELVETICA_18
BACKGROUND SHADOW
COLOR 1.0 1.0 1.0
INTEREST POINT 200000 600000 10000
LOCAL_ORIGIN 285039.311421 4372264.503058 0
NEW_SCALE 0.01
INTEREST POINT 88000 430000 2000
NEW_SCALE 0.05
INTEREST POINT 78000 401000 500
NEW_SCALE 0.50
INTEREST POINT 76700 399000 100
TEXT "Kittery, Maine"
```

Vector Data (.vector) Format

The Vector file type is intended for representing data that should be displayed as vector fields.

The structure of the vector file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN VECTOR < *Version-Number*>	Currently, there is only one version (1.0).
{ **STATIC** }	"STATIC" specifies that vectors are not animated at a certain point in time.
SCALE < *scale* >	Specifies the amount to scale the vectors to make them visible when viewed as a group.

Example

The following example shows a curtain plot file for a curtain consisting of only 8 columns, each with at most 15 data points, and with data coded by an index. The curtain reference height is 500 meters below sea level.

```
BEGIN CURTAIN 1.0
CLUT
NCOLS 6
Z_STEP 10.0
LOCAL_ORIGIN 483785.00 5297590.00 -500.00
501.23 -5935.73 10.54 15 58605d55514f464948474948414741
510.66 -6029.69 13.33 15 575151514c484d46463f4340474242
520.39 -6126.67  8.67 14 5e69625c545146403e41433f3b43
529.82 -6220.62  8.30 10 d5f5b51564a4b4644424
540.22 -6323.12  9.97  8 666d674f494a4342
551.37 -6431.33 11.41  8 535b584947464240
562.51 -6539.54 12.73 12 625950484c424a4442474443
574.02 -6651.24 11.05 15 656b6c6863655e5855514e40434543
```

Vessel (.vessel) Format

The Vessel file type creates a crude, boxy-looking boat that has a tail and a course-made-good indicator line. The structure of the Vessel file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN VESSEL < *Version-Number*>	Currently, there is only one version (1.0).
BOW_SIZE < *Width*> < *Length*> < *Height*>	Specifies the dimensions of the bow (front) of the vessel. The Width and Height arguments indicate what the hull dimensions taper to Length meters from the front end of the hull.
HULL_SIZE < *Width*> < *Length*> < *Height*>	Specifies the dimensions of the hull (main body) of the vessel. All dimensions are given in meters. The hull is just a big rectangular solid that connects the bow, stern, and bridge together. The main reference point for the vessel is in the center of the bottom-rear of the hull, where the tail comes out.
STERN_SIZE < *Width*> < *Length*> < *Height*>	Specifies the dimensions of the stern (rear) of the vessel. The Width and Height arguments indicate what the hull dimensions taper to Length meters from the back end of the hull.
BRIDGE_SIZE < *Width*> < *Length*> < *Height*>	Specifies the dimensions of the bridge (box on top) of the vessel. All dimensions are given in meters. The bridge is a rectangular solid that sits on top of the hull, starting one quarter of the way up the hull.
SHADOW {**ON** | **OFF**}	Specifies whether or not a shadow should be shown below the vessel over the bathymetry underneath. NOT IMPLEMENTED YET.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The coordinates of the bottom center of the vessel, where the hull meets the stern.

Example

The following example shows a vessel with its keel 10 meters below the surface.

```
BEGIN VESSEL 1.0
BOW_SIZE 2.0 30.0 1.0
HULL_SIZE 25.0 180.0 18.0
STERN_SIZE 15.0 20.0 15.0
BRIDGE_SIZE 20.0 40.0 18.0
LOCAL_ORIGIN 355508.312009 4768062.760335 -10
```

GZX OBJECTS

XML Scene (.gzx) Format

The Scene file type is a special case that does not follow the regular format. It is not an object, but actually a collection of objects, textures, and scripts. It also allows for certain objects and scripts to be specified on-the-fly without pointing to another file. The format for the XML Scene file is as follows (bold words must appear *exactly* as they do here, including case).

<?xml version="1.0"?>	Must appear at the very beginning of the file. Identifies the XML version used.
<GZ>	Identifies this XML file as a GeoZui file.
{< *Scene-Item*>}*	Any number of Scene-Items may be present between the GZ statements. Scene-Items include Object-Items, Texture-Items, Script-Items, and Collections, as described in the next table.
</GZ>	Must appear at the end of the GeoZui-specific information. No other top-level tags can be present, and no more than one GZ section is read.

Scene Items

A < *Scene-Item*> within the XML Scene file format takes one of the following formats:

<Collection [**label=**< *label*>]**>** < *Scene-Item*>* **</Collection>**	A Collection allows object-items to be nested and given a collective name.
<Texture filename=< *filename*> [**label=**< *label*>] [**default=**< *value*>] **>** **</Texture>**	Causes GeoZui to load a texture of the given filename. A label can be given (otherwise the texture name is generated from the filename), and if the default tag is present, then the texture is designated as the default texture for objects.
<Object {**filename=**< *filename*> | **type=**< *type*> [< *tags*>} [**label=**< *label*>] [**texture=**< *texture*>] **>** [< *inline-object-info*>] **</Object>**	Causes GeoZui to either load an of the given filename or create an object of the given type. A label can be given (otherwise the object name is generated from the filename), and a texture can be given, either as the name of a texture already loaded, or as "default" (with quotes included).
<Script [**filename=**< *filename*>]**>** [< *inline-TCL-script*>] **</Script>**	Causes GeoZui to source a script of the given filename, if a filename is given. Also executes any text between the Script tags as TCL.

Example

The following example loads two textures and three objects. It runs a script called "testscript.tcl" between loading endeavor.gutm and r509.gutm, then runs an embedded script at the end.

```
<?xml version="1.0"?>
<GZ>
    <Texture filename="default.clut" label="default" default="true"></Texture>
    <Texture filename="finegrid.grid" label="finegrid"></Texture>
```

```
    <Object filename="nonan.gutm" texture="default"></Object>
    <Object filename="endeavor.gutm" texture="default"></Object>
    <Script filename="testscript.tcl"></Script>
    <Object filename="r509.gutm" texture="finegrid"></Object>
    <Script>puts "Here is an embeded script!\n"</Script>
</GZ>
```

Chart Object Format

A chart object displays a horizontal image at sea level, such as a nautical charts. When fused with a gutm, a hole will be created trough the image to display the gutm. Currently, only TIFF format images are supported for chart objects.

imagefile	Location of the TIFF file.
worldfile	Location of the corresponding *World Header File* as used in *ArcView* and *Arc/INFO*.

Example

```
<?xml version="1.0"?>
<GZ>
  <Object type="Chart" imagefile="portsmouth.tif" label="Porstmouth_chart" worldfile="portsmouth.tfw"></Object>
</GZ>
```

Tide Object Format

Tides are represented as a series of Gutms representing waterlevel hights above a datum. Each timestep is represented by a Timestep tag in the Object tag of type Tide.

time	Time in seconds since January 1st, 1970.
filename	Filename of the gutm for this timestep.

Example

```
<?xml version="1.0"?>
<GZ>
  <Object type="Tide" label="tide">
    <Timestep time="1023755397.12" filename="tidegrid37418.0208.txt.gutm"></Timestep>
    <Timestep time="1023759000" filename="tidegrid37418.0625.txt.gutm"></Timestep>
    <Timestep time="1023762602.88" filename="tidegrid37418.1042.txt.gutm"></Timestep>
    <Timestep time="1023766197.12" filename="tidegrid37418.1458.txt.gutm"></Timestep>
  </Object>
</GZ>
```

Time-Varying Gutm

A TVGutm is a surface that varies in time. Essentially, it is a collection of Gutms that are displayed one at a time, at their specified time.

time	Time in seconds since January 1st, 1970.
filename	Filename of the gutm for this timestep.

Example

```
<?xml version="1.0"?>
<GZ>
  <Object type="TVGutm" label="Land" texture="land">
    <Timestep time="946702800" filename="step0.gutm"></Timestep>
    <Timestep time="946703800" filename="step1.gutm"></Timestep>
    <Timestep time="946704800" filename="step2.gutm"></Timestep>
    <Timestep time="946705800" filename="step3.gutm"></Timestep>
  </Object>
</GZ>
```

NetCDF Object Formats

GeoZui3D has limited support for loading objects from netCDF files. A .gzx file with an Object type of NetCDF is used to describe how to interpret the data within the netCDF file.

ncfile	Filename of the netCDF file

A NetCDF Object tag may contain one or more of the following object types.

Flow

A flow tag describes how to read a collection of moving particles. The particles are rendered using an interpolated trail between particle positions to illustrate their flow.

Values may be interpreted from a netCDF file in various ways. The following codes are used to describe the source of some values:

d	Dimension
va	Variable attribute
ds	Dimension size

The follwoing values are used to determine what and how to read from the netCDF file.

var	Variable in the netCDF file containing the particle data
time	Dimension representing the times
timestep	Length of time bwtween steps
x	Which dimension and index points to the x values
y	Which dimension and index points to the y values
z	Which dimension and index points to the z values
count	How to determine the number of particles
headcolor	RGB color of the head of the particle streaks
tailcolor	RGB color of the tail of the particle streaks
linewidth	Width in pixels of the streaks.
antialias	If set to "true",
length	length in timesteps of the streaks.

A flow tag may also include the following tags:

BBox

A BBox tag represents a bounding box that colors all particles that start within.

east	East edge of the bounding box.
west	West edge of the bounding box.
north	North edge of the bounding box.
south	South edge of the bounding box.
top	Top of the bounding box.
bottom	Bottom of the bounding box.
headcolor	RGB color of the head of the particle streaks starting in the bounding box.
tailcolor	RGB color of the tail of the particle streaks starting in the bounding box.

Highlight

The highlight tag allows to highlight every nth particle trace.

step	Determines how many particles to skip between highlighted ones.
headcolor	RGB color of the head of the highlighted particle streaks
tailcolor	RGB color of the tail of the highlighted particle streaks

Example

```
<?xml version="1.0"?>
<GZ>
    <Object type="NetCDF" ncfile="nh_drogue/drogue.nc" label="NH_drogue"
    usetexture="false">
        <Flow var="Drogues" time="d Time" timestep="va Drogues Timestep"
            x="d Coordinate 0" y="d Coordinate 1" count="ds Count"
            headcolor="0.5 0.5 0.5" tailcolor="0.1 0.1 0.1" linewidth="1.75"
            antialias="false" length="3">
            <BBox name="A"
                east="348500" west="347500" north="4771500" south="4770500"
                headcolor="0.0 1.0 0.0" tailcolor="0.0 0.5 0.0">
            </BBox>
            <BBox name="B"
                east="362500" west="361500" north="4768750" south="4767250"
                headcolor="1.0 0.0 0.0" tailcolor="0.5 0.0 0.0">
            </BBox>
            <BBox name="C"
                east="352900" west="352400" north="4775750" south="4775250"
                headcolor="0.0 1.0 1.0" tailcolor="0.0 0.4 0.4">
            </BBox>
            <BBox name="D"
                east="369800" west="368800" north="4763000" south="4762000"
                headcolor="0.0 1.0 0.7" tailcolor="0.0 0.2 0.2">
            </BBox>
        </Flow>
    </Object>
</GZ>
```

Gutm

Not yet available

Tide

Not yer available

COMPOSITE OBJECT FILE FORMATS

General Composite Object Format

All composite object formats have a similar structure, since they are essentially objects that in some sense "manage" other objects. Elements in this format that act unchanged across composite objects only appear in this section (to reduce clutter and repetition). The structure is as follows:

[Optional Comments]	As normal, anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN ...	See the General file format description for details about the BEGIN keyword.
N_RECORDS < *Posable-Count*>	Specifies how many posables are to follow the LOCAL_ORIGIN keyword and be "managed" by this composite object.
[Header Contents]	Each *Data-Type* has its own additional keywords.
[**REVERSED_XY**]	See the General file format description for details about the REVERSED_XY keyword.
[**PROJECT** ...]	See the General file format description for details about the PROJECT keyword.
LOCAL_ORIGIN ...	See the General file format description for details about the LOCAL_ORIGIN keyword.
< *Posable**>	The lines following the LOCAL_ORIGIN keyword each contains the address of exactly one posable (which must already be loaded before the composite object is loaded). There should be as many of these lines as specified after N_RECORDS.
[Data]	Additional data may appear afterward, depending on the composite object.

AveragingOverview (.avov) Format

The AveragingOverview file type creates a composite object that tracks all managed posables. It optionally maintains a position that is at the center of the box bounding the managed posables, an orientation that is the average of the managed posables, and a scale that allows an attached window to contain all of the managed posable positions in it's central region. The structure of the AveragingOverview file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN AVERAGING_OVERVIEW < *Version-Number*>	Currently, there is only one version (1.0).
TRACKING {{H | P | R | S | T | X | Y | Z}*}	Specifies which components of the managed posables are to be tracked, in terms of maintaining the average composite value. Each letter is the first letter of the particular component: Heading, Pitch, Roll, Scale, Translation (XYZ), X-position, Y-position, and Z-position. Defaults to all except Roll being tracked (HPST).
AVERAGE [MEAN | MIN_MAX]	Specifies whether the average position should be calculated as a mean of all managed posables, or as a mean of the min/max extents of these posables. Defaults to MIN_MAX.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The value for the LOCAL_ORIGIN has little meaning, but for the sake of rendering accuracy, should be in the vicinity of where the managed posables are expected to be.
< *Posable**>	The list of managed posables.
	The AveragingOverview format has no data after the list of posables.

Example

This example creates an AveragingOverview that tracks the scale and position of two objects. When there are only two managed posables, there is no difference between AVERAGE MEAN and AVERAGE MIN_MAX.

```
BEGIN AVERAGING_OVERVIEW 1.0
N_RECORDS 2
TRACKING TS
AVERAGE MEAN
LOCAL_ORIGIN 493278.000000 5312620.000000 -2500.000000
Object.Sample1_sample
Object.Sample2_sample
```

Boundary (.boundary) Format

The Boundary file type creates a composite object that generates events whenever a managed posable crosses a planar boundary. The boundary is the plane passing through the position of the Boundary pose, where the plane is normal to the heading/pitch/roll of the Boundary pose. The boundary is rendered as a rectangle starting at the LOCAL_ORIGIN, with a width determined by the SCALE, and a height determined by the SCALE and HEIGHT_RATIO. The structure of the Boundary file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN AVERAGING_OVERVIEW < *Version-Number*>	Currently, there is only one version (1.0).
SCALE < *Scale*>	Specifies the width (and contributes to the height) of the rendered rectangle of the boundary. Defaults to 1.0.
HEIGHT_RATIO < *Ratio*>	Specifies the height of the rendered rectange of the boundary with respect to the scale. The height is determined by multiplying the scale by this value. Defaults to 0.0 (renders as a line segment).
YAW < *Angle*>	Specifies the heading of the normal vector to the boundary, pointing in the "inward" direction. Defaults to 0.
PITCH < *Angle*>	Specifies the pitch of the normal vector to the boundary, pointing in the "inward" direction. Defaults to 0.
ROLL < *Angle*>	Specifies the roll of the rendered rectangle with respect to the boundary's normal vector. Defaults to 0.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	Specifies the origin of the rendered rectangle.
< *Posable**>	The list of managed posables, for which events will be generated whenever they cross the boundary.
	The Boundary format has no data after the list of posables.

Example

This example creates a Boundary that faces nearly east, and is pitched down at 45 degrees. When rendered, the Boundary has width of 100m and a height of 80m. It generates a boundaryCrossedOut or boundaryCrossedIn event whenever the main window crosses the boundary (in on the east side, out on the west).

```
BEGIN BOUNDARY 1.0
N_RECORDS 1
SCALE 100
HEIGHT_RATIO 0.8
YAW 80
PITCH 45
LOCAL_ORIGIN 493278.000000 5312620.000000 -2000.000000
Window.main
```

ProximityHighlighter (.proximity) Format

The ProximityHighlighter file type creates a composite object that tracks the two closest of its managed posables. It optionally maintains a position that is at the center of the two closest posables and a scale that allows an attached window to contain both in it's central region. Events are generated as the two closest managed posables cross certain thresholds of proximity. Note that events are only generated for the nearest two, even if other posables come within proximity of each other. The structure of the ProximityHighlighter file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN PROXIMITY_HIGHLIGHTER < *Version-Number*>	Currently, there is only one version (1.0).
TRACKING {ON | OFF}	Specifies whether or not the position of the nearest two objects should be tracked (and maintained in the ProximityHighlighter's posable). Defaults to ON.
THRESHOLD < *distance*>	Specifies the distance at which the two nearest managed posables should cause the ProximityHighlighter to trigger and event. Triggers an inProximity event when two managed posables are within *distance* meters of each other, an inCloseProximity event when they are within half that distance, an outOfCloseProximity event when they were previously in close proximity and move out of 0.55 times the distance, and and outOfProximity event when they were in proximity and they move out of 1.1 times the distance. Defaults to 60 meters.
LOCAL_ORIGIN < *UTM-X*> < *UTM-Y*> < *Height*>	The value for the LOCAL_ORIGIN has little meaning, but for the sake of rendering accuracy, should be in the vicinity of where the managed posables are expected to be.
< *Posable**>	The list of managed posables.
	The ProximityHighlighter format has no data after the list of posables.

Example

This example creates a ProximityHighlighter that tracks the scale and position of the nearest of four objects, triggering events when any two of them are within 20 meters of each other.

```
BEGIN PROXIMITY_HIGHLIGHTER 1.0
N_RECORDS 4
THRESHOLD 20
LOCAL_ORIGIN 493278.000000 5312620.000000 -2500.000000
Object.Sample1_sample
Object.Sample2_sample
Object.Sample3_sample
Object.Sample4_sample
```

NON-OBJECT FORMATS

Legacy Scene (.sdv) Format

The Scene file type is a special case that does not follow the regular format. It is not an object, but actually a collection of objects and textures. It also allows for the automatic invocation of a script upon loading. The format for the Scene file is as follows (bold words must appear *exactly* as they do here, including case).

---

textures	Must appear at the very beginning of the file.
{< *Texture-Filename*> < *Texture-Label*>[< *Default-Indicator*>]}*	Any number of texture filename-label pairs may be present after the "textures" line. The Texture-Label is the name used to represent the texture loaded in from the file specified by Texture-Filename. Only one texture may have the Default-Indicator present, and there must not be any space between int and the Texture-Label. The Default-Indicator is the star character ("*").
objects	Must appear immediately after the last texture filename-name pair, without any intervening lines.
{< *Object-Filename*> < *Object-Label*> < *Texture-Label*>}*	Any number of object specification triples may be present after the "objects" line. The Object-Label is the name used to represent theobject loaded in from the file specified by Object-Filename. The Texture-Label specifies which texture should be mapped to this object when it is first rendered. If Texture-Label is "default", then the texture marked with the Default-Indicator is assigned.
end	Marks the end of required information (objects and textures). Must appear immediately after the last object specification triple, without any intervening lines.
[< *Script-Filename*>]*	Optional scripts to be run upon completion of loading all textures and objects. If present, the first line must appear immediately after the "end" line. Each script must be on a separate line without any intervening lines between them.

Example

The following example loads two textures and five objects. It runs a script called "tube.tcl" after everything has been loaded.

```
textures
default.clut default.clut*
myclut.clut myclut
objects
r509.gutm r509.gutm myclut
ROV4.tube ROV4.tube default
animals.point animals.point default
nonan.gutm nonan.gutm default
Sample1.sample Sample1_sample default
end
tube.tcl
```

Conversion Grid Format

The Conversion Grid file type is intended for bringing in regularly gridded surfaces that are not in UTM coordinates. It allows holes and oddly shaped boundaries, but uses a rectangular bounding grid area for its specification. This grid can then be converted to a GUTM grid using existing or user-provided projections and/or conversion methods. Conversion grids are not currently supported as their own file type, but are used for the GUTM "PROJECT" specification. The structure of the Conversion Grid file format is as follows:

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN CONVERSION_GRID < *Version-Number*>	Currently, there is only one version (1.0).
LAYOUT {**LITERAL** | **ROW_MAJOR** | **COLUMN_MAJOR**} [**INVERTED_ROWS**]	Specifies how the height data is layed out. LITERAL indicates that the data are arranged in rows and columns in the file (one row to a line). ROW_MAJOR and COLUMN_MAJOR indicate that the data is listed in row major or column major order, respectively, according to other parameters in the header. If INVERTED_ROWS is specified, then rows are expected to be given in the data from the northwest corner, down, rather than the usual southwest corner, up.
[**COLS_ROWS** < *Column-Count*> < *Row-Count*>]	Specifies the number of points in each column and row of the grid. COLS_ROWS should not appear if LAYOUT is LITERAL.
CELLSIZE { **IMPLICIT** | {{< *X-Size*> | **VARIABLE**} {< *Y-Size*> | **VARIABLE**} [**PER_RECORD**]}	Specifies the distance between grid points in X and Y directions. If IMPLICIT, then X and Y coordinates are expected for every point, always as an (x, y, z) triple. COLS_ROWS must be used and SHAPE must be FULLGRID if the IMPLICIT keyword is used here. If a number is given for X-size or Y-size, it specifies the width or height (respectiviely) of every cell. If VARIABLE is given for a dimension, it means that the individual coordinates will be given for each column and/or row, respectively. If PER_RECORD does not appear, the coordinates for each row and column are expected in sections delimited by the SECTION_DELIMITER. The non-major-ordered coordinates always appear in the data section first (column coordinates for row-major order, for instance) followed by the major-ordered coordinates (rows, for our example). If PER_RECORD appears, the major-ordered coordinates are expected at the beginning of each line record (whether this is a row or a column), and not in a separate section.
GRID_ORIGIN < *X-Origin*> < *Y-Origin*>	The coordinates of the Southwestern corner of the grid. This line is required if any numeric values are given for CELLSIZE. Values given here are ignored if the CELLSIZE for the corresponding dimension is given as VARIABLE.
[**SECTION_DELIMITER** < *Delimiter*>]	Specifies what ends a particular section of data. If no SECITON_DELIMITER is given, "END" is the default.
[**VALID_DATA** {**LESS_THAN** | **GREATER_THAN** | **BETWEEN**} < *value1*> [< *value2*>]]	Specifies what values constitue valid data (the rest are considered to be missing, or a "hole"). Any data that does not start with a digit or sign will also be considered missing. If VALID_DATA does not appear, all numeric data is considered valid.
[**SHAPE** { **FULLGRID** | **PARTIAL** }]	Specifies whether the data that follow specify the start and end of a line (PARTIAL) or not (FULLGRID). FULLGRID is the default if no SHAPE is given. SHAPE should not appear if LAYOUT is LITERAL.
END_HEADER	Specifies end of header.
< *Data*>	Specified from South to North and West to East, according to the parameters specified earlier in the header.

Color Look-Up Table (.clut) Format

The Clut file type is another special case that does not follow the regular format. It is not an object, but a specification for color assignment. Clut's can be used either as height-mapped textures or as lookup tables for data values, depending on the object they are applied to. The format for the Clut file is as follows (bold words must appear *exactly* as they do here, including case).

---

clut1.0	Must appear at the very beginning of the file.
{< *Value*> < *Red*> < *Green*> < *Blue*> < *Alpha*>}*	Any number of look-up table entries can be present after the "clut1.0" line. The Value is a floating point number, indicating the top of the range to have the specified color. The color is specified as levels of red, green, blue, and alpha; the levels are simple integers between 0 and 255 inclusive. If this Clut is used as a height-mapped texture, the color associated with the topmost value is used for all values above that value as well. For all other applications, values outside the range of the table entries result in a neutral color.
done	Must appear immediately after the last look-up table entry, without any intervening lines. Indicates the end of Clut information. Anything after this line is ignored.

Example

The following example specifies 11 different colors at values spaced 10.0 apart (when used as a height mapped texture, this would be 10.0 meters apart). Colors range from blue (-2190) to red (-2000).

```
clut1.0
-2190.000000 0 126 255 205
-2180.000000 0 169 255 205
-2170.000000 0 236 255 205
-2160.000000 0 255 126 205
-2150.000000 81 255 0 205
-2140.000000 185 255 0 205
-2130.000000 255 250 0 205
-2120.000000 255 232 0 205
-2110.000000 255 183 0 205
-2100.000000 255 104 0 205
-2000.000000 255 0 0 205
done
```

Draped Grid (.grid) Format

The Draped Grid file provides a specification for a grid pattern to be draped on a surface. Grids are intended to produce a "graph paper" look, and provide a convenient way of measuring distance on a horizontal surface. A grid is defined by specification of a single tile, which is replicated across the surface to which it is assigned. The tile is made up of an *image* for which width and height in pixels and a background color may be specified. A *maingrid* and up to 10 *subgrids* are drawn on this tile. The maingird determines the width and height in meters of the tile, and has an outline drawn around it of specified thickness and color. Thus a tile with only a maingrid looks like an outlined rectangle. Each subgrid is used to subdivide the maingrid, and consists of a number of vertical and horizontal dividing lines of specified thickness and color. When a draped grid texture tile is created, first the tile is filled in with the background image color. Then, drawing begins in order from the last subgrid specified in the file up to the first, and finally the maingrid outline is drawn. Thus, subgrids specified early in the file will draw over those specified later. The format for a Draped Grid file is as follows. Note, that lines need only be included in the file if values other than the default values are desired. The default grid contains only a main grid, but if subgrids are specified, the defaults for a subgrid apply unless overridden. Section headers (image, maingrid and subgrid) can appear in any order, and within a section order of the lines is unimportant.

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN GRID < *Version-Number*>	Currently, there is only one version (1.0).
< *Image-Section*>	This is the background specification for the grid tile. The format of the Image-Section is found in the first table below.
< *Maingrid-Section*>	This is the specification for the topmost level of the grid, which is essentially a border around the grid tile. The format of the Maingrid-Section is found in the second table below.
{< *Subgrid-Section*>}*	These are the specifications for the subdivisions within a grid tile. There may be up 10 Subgrid-Sections. The format of the Subgrid-Section is found in the third table below.

Grid Image-Section Format

< *Image-Section*> within the Grid file format takes the following form.

IMAGE	Indicates that general specifications for a background image tile follow.
WIDTH < *pixel-width*>	The width (east-west direction) in pixels of one tile. Default = 512 pixels.
HEIGHT < *pixel-height*>	The height (north-south direction) in pixels of one tile. Default = 512 pixels.
COLOR < *red*> < *green*> < *blue*> < *alpha*>	The tile's background color. Color components are specified as decimal fractions between 0 and 1. An alpha value of 0 indicates that the image background is fully transparent, 1 that it is fully opaque. Default = 0 0 0 0 (transparent black).

Grid Maingrid-Section Format

< *Maingrid-Section*> within the Grid file format takes the following form.

MAINGRID	Indicates that general specifications for the main grid drawn on an image tile follow.
HSIZE < *width*>	The horizontal size (east-west direction) in meters of the main grid. Default = 100 m.
VSIZE < *height*>	The vertical size (north-south direction) in meters of the main grid. Default = 100 m.
LINEWIDTH < *thickness*>	The thickness in pixels of the outline to be drawn around main grid tile. Default = 10 pixels.
COLOR < *red*> < *green*> < *blue*> < *alpha*>	The color of the outline to be drawn around main grid tile. Color components are specified as decimal fractions between 0 and 1. An alpha value of 0 indicates that the line is fully transparent, 1 that it is fully opaque. Default = 0.3 0.3 0.9 0.7.

Grid Subgrid-Section Format

< *Subgrid-Section*> within the Grid file format takes the following form.

SUBGRID	Indicates that general specifications for a subgrid drawn on an image tile follow.
HDIVS <*m*>	The number of horizontal divisions (east-west direction) across one tile of the main grid. Default = 10 divisions.
VDIVS <*n*>	The number of vertical divisions (north-south direction) across one tile of the main grid. Default = 10 divisions.
LINEWIDTH < *thickness*>	The thickness in pixels of the lines forming the subgrid. Default = 5 pixels.
COLOR < *red*> < *green*> < *blue*> < *alpha*>	The color of the lines forming the subgrid. Color components are specified as decimal fractions between 0 and 1. An alpha value of 0 indicates that the line is fully transparent, 1 that it is fully opaque. Default = 0.2 0.2 0.7 0.7.

Example

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A Grid that looks like a blue and red lined graph paper
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
begin grid 1.0

image
 width 512 % the base grid image tile will be 512x512 pixels
 height 512
 color 1 1 1 1 % the background of the grid will be opaque white

maingrid
 hsize 100 % the main grid cell is 100m x 100m
 vsize 100
```

```
 linewidth 8 % main grid lines are drawn 8 pixels wide
 color 0.4 0.4 1 1 % main grid lines are light blue, fully opaque

subgrid
 hdivs 2 % this subgrid divides the main grid in half (50 m)
 vdivs 2
 linewidth 4 % lines are 4 pixels wide
 color 0.9 0.35 0.35 1 % lines are light red

subgrid
 hdivs 10 % this subgrid divides the main grid in tenths (10m)
 vdivs 10
 linewidth 2 % lines are 2 pixels wide
 color 0.5 0.5 0.9 1 % lines are darker and grayer than main blue lines

subgrid
 hdivs 100 % this subgrid divides the main grid in hundreths (1m)
 vdivs 100
 linewidth 1 % lines are 1 pixel wide
 color 0.3 0.3 0.7 1 % lines are even darker and grayer than 10m lines
```

Georeferenced Draped Image (.jgw & .tfw) Formats

The georeferenced draped image file type is a special case that does not follow the regular format. It is not an object, but a specification for how an image is to be georeferenced before being draped on a horizontal surface. The format used is the *World Header File* format used in *ArcView* and *Arc/INFO*. Currently only *JPEG* and *TIFF* file formats are supported. The suffix of the file name determines which type of image file is to be loaded (.jgw = .jpg, .tfw = .tif). The body of the name of the file determines the body of the name of the image file to be loaded. For example, the file *neahbay.twf* will provide georeferencing information for the image file *neahbay.tif*. The image file must be located in the same directory. The file itself contains six lines which determine size, orientation, and geographic position of the image. The format is as follows.

< x cellsize>	The size of a single pixel of the georeferenced image in the east direction.
0	Must be zero (used in Arc/INFO to rectify rotation, but not supported in GeoZui3D)
0	Must be zero (used in Arc/INFO to rectify rotation, but not supported in GeoZui3D)
< y cellsize>	The size of a single pixel of the georeferenced image in the north direction.
<x>	The projected east position in meters (UTM) of the origin of the image to be draped. If this entry is positive, the origin of the image is on its west side. If it is negative the origin of the image is on its east side.
<y>	The projected north position in meters (UTM) of the origin of the image to be draped. If this entry is positive, the origin of the image is on its south side. If it is negative the origin of the image is on its north side.

Example

The following example indicates that the image is to be loaded with its origin in the north-west (upper left-hand) corner. This corner is geopositioned at (493278.0, 5313619.6) in UTM coordinates. Pixels of the image are square, with linear dimension 1.96 m.

```
1.96
0
0
-1.96
493278.000000
5313619.6000000
```

Draped Lattice Texture (.lat) Format

The Draped Lattice Texture file provides a very simple technique for producing a variety of both regular and irregular patterns with which to texture a surface. A lattice texture is defined by specification of a single tile, which is replicated across the surface to which it is assigned. The lattice texture tile is made up of an *image* for which width and height in pixels, a background color, and width and height in meters can be specified. In addition, a filtering operation can be specified, which is used to low-pass filter the image once it is created. Up to 10 *lattices* may be drawn on this tile. Each

lattice is a regular grid of lattice points (intersections of grid lines) at which a pattern element may be drawn. Associated with each lattice is a specification for how pattern elements are to be drawn into the tile. When a draped lattice texture tile is drawn, the tile is first filled in with the background image color. Then, drawing begins in order from the last lattice specified in the file up to the first. Thus, lattices specified early in the file will draw over those specified later. The format for a Draped Lattice file is as follows. Note, that lines need only be included if values other than the default values are desired. By default (i.e. if no lattices are explicitly specified in the file), only a single lattice is created with the default parameters.

[Optional Comments]	Anything up to the first line beginning with the keyword "BEGIN" is ignored, and is assumed to be a comment.
BEGIN LATTICE < *Version-Number*>	Currently, there is only one version (1.0).
< *Image-Section*>	This is the background specification for the lattice tile. The format of the Image-Section is found in the first table below.
{< *Lattice-Section*>}*	These are the specifications for the various texture elements to be generated within the lattice tile. There may be up to 10 Lattice-Sections. The format of the Lattice-Section is found in the second table below.

Lattice Image-Section Format

< *Image-Section*> within the Lattice file format takes the following form.

IMAGE	Indicates that general specifications for a background image tile follow.
WIDTH < *pixel-width*>	The width (east-west direction) in pixels of one tile. Default = 512 pixels.
HEIGHT < *pixel-height*>	The height (north-south direction) in pixels of one tile. Default = 512 pixels.
HSIZE < *width*>	The horizontal size (east-west direction) in meters of a single tile. Default = 100 m.
VSIZE < *height*>	The vertical size (north-south direction) in meters of a single tile. Default = 100 m.
COLOR < *red*> < *green*> < *blue*> < *alpha*>	The tile's background color. Color components are specified as decimal fractions between 0 and 1. An alpha value of 0 indicates that the image background is fully transparent, 1 that it is fully opaque. Default = 0 0 0 0 (transparent black).
FILTER [**NOFILTER** | [**GAUSS** | **BOX**] <*w*>]	Keyword NOFILTER indicates that no filtering is to be done, GAUSS that a gaussian filter is to be applied, and BOX that a box filter is to be applied. The filter kernel is of width *w*. Default = NOFILTER.

Lattice Lattice-Section Format

< *Lattice-Section*> within the Lattice file format takes the following form.

LATTICE	Indicates that general specifications for a lattice drawn on an image tile follow.
ROWS <*n*>	The number of lattice rows across one tile of the main grid. Default = 10 rows.
COLS <*n*>	The number of lattice columns across one tile of the main grid. Default = 10 columns.
PROBABILITY <*p*>	The probability that an individual lattice element will be drawn when the texture tile is created. *p* ranges between 0.0 and 1.0, where 0.0 = never drawn, 1.0 = always drawn. Default = 1.0
HJITTER [**CONSTANT** <*offset*>| [**NORMAL**	**UNIFORM**] <*offset*> <*range*>]	Horizontal (east-west) offset and jitter. Keyword CONSTANT indicates that no horizontal jitter is to be applied, NORMAL that a normally distributed jitter is to be applied, and UNIFORM that a uniformly distributed jitter is to be applied. The *offset* determines a fixed horizontal offset of the point from the lattice point. The *range* parameter indicates one standard deviation for a NORMAL distribution and indicates the maximum magnitude deviation for a UNIFORM distribution. All distances are specified in meters. Default = CONSTANT 0
VJITTER [**CONSTANT** <*offset*>| [**NORMAL**	**UNIFORM**] <*offset*> <*range*>]	Vertical (north-south) offset and jitter. Keywords and paramters are the same as for HJITTER. Default = CONSTANT 0
RJITTER [**CONSTANT** <*offset*>| [**NORMAL**	**UNIFORM**] <*offset*> <*range*>]	Rotational offset and jitter. Keyword CONSTANT indicates that no rotational jitter is to be applied, NORMAL that a normally distributed jitter is to be applied, and UNIFORM that a uniformly distributed jitter is to be applied. The *offset* determines a fixed rotational offset. The *range* parameter indicates one standard deviation for a NORMAL distribution and indicates the maximum magnitude deviation for a UNIFORM distribution. All angles are specified in degrees. Default = CONSTANT 0
STYLE [**DOT** <*diameter*>| [**LINE**	**GABOR**] <*angle*><*length*>< *thickness*>]	The style of the patterning element to be drawn at the lattice points. Keyword DOT signifies a filled circle with the indicated diameter. LINE signifies a straight line drawn at the indicated angle, and with the specified length and thickness. GABOR signifies a gabor function drawn with its main axis at the indicated angle, and with the specified length determining wavelength and thickness determining the ratio of gaussian size to wavelength. Default = DOT 0.1
COLOR <*red*> < *green*> < *blue*> < *alpha*>	The color to be used when drawing the patterning element. Color components are specified as decimal fractions between 0 and 1. An alpha value of 0 indicates that the element is fully transparent, 1 that it is fully opaque. Default = 0.2 0.2 0.7 0.7.

Examples

```
%-------------------------------------------------------------------------
% A sample regular lattice texture using both lines and dots
%-------------------------------------------------------------------------
begin lattice 1.0

image
  width 512  % the base image tile will be 512x512 pixels
  height 512
  color 1 1 1 1  % the background color is opaque white
  hsize 50  % one image tile is taken to be 50m x 50m
  vsize 50
  filter gauss 5 % 5x5 gaussian filter after drawing

% a 10 x 10 lattice of short angled red lines
% by default, all lines are drawn and there is no jitter or offset
lattice
  rows 10  % 10 x 10 lattice
  cols 10
  style line 45 2 0.4 % draw each line at 45 degrees, 2 m long, 0.4 m wide
  color 1 0.2 0.2 1 % lines are red

% a 5 x 5 lattice of green dots
% by default, all dots are drawn and there is no jitter or offset
lattice
  rows 5
  cols 5
  style dot 1.6  % draw each point as 1.6 m diameter dot
```

```
    color 0.2 1 0.2 1 % dots are green

  % a 1 x 50 lattice of vertical blue lines
  % the lines are rotated 90 degrees and are long enough so that when the
  % texture is tiled, they make unbroken vertical lines
  % by default, all lines are drawn and there is no jitter or offset
  lattice
    rows 1  % 1 row in lattice
    cols 50  % 50 cols in lattice
    style line 90 50 0.4 % draw each vertical line 50 m long, 0.4 m wide
    color 0.3 0.3 1 1 % lines are blue
  %-------------------------------------------------------------------


  %-------------------------------------------------------------------
  % a sample irregular lattice texture that creates a random dot pattern
  %-------------------------------------------------------------------
  begin lattice 1.0

  image
    width 512  % the base grid image will be 512x512 pixels
    height 512
    color 0 0 0 0  % the background color is transparent black
    hsize 50  % the total image is taken to be 50m x 50m
    vsize 50
    filter gauss 3 % 3x3 gaussian filter after drawing

  lattice
    rows 250  % row and col divisions are 0.1 meters
    cols 250
    probability 0.25 % 25% probability of lattice point being drawn
    hjitter uniform 0 0.1 % uniform dist. no offset, range +/- 0.1
    vjitter uniform 0 0.1
    style dot 0.4  % draw each point as 0.4 m diameter dot
    color 1 0.2 0.2 1 % dots are opaque red
  %-------------------------------------------------------------------
```

9/1/99

How to get perfect colour in X Mac 68k

Introduction:

Sick of staring at a red and black xterm? Wish your window managers colours weren't garbled? I'm not a programmer but I've worked out a way to get perfect colour in X Mac 68k.

X seems to add colours to a palette as they appear on the screen. For example, your window-manager appears first so its colours are added. Launch an application and its colours are added directly after the window managers and so on... On most platforms these colours display correctly, but on a 68k Mac we're stuck with the 256 colour table that Penguin provides us. X thinks its displaying the right colours when in fact its displaying the corresponding Penguin colour in the colour table.

OK, but how do I get the right colours?

It is possible to transfer the X colour table over to Penguin. This, however has 3 disadvantages:

1. All this does is 'trick' Penguin into displaying the right colours.

2. We only have 256 colours to work with; once the colour table is full colours start to go funny.

3. Applications have to be launched in the same order each time for them to display correctly.

DON'T LET THIS DETER YOU! It is easy to get a perfect looking window-manager with no fuss at all.

Here's how to do it:

1. Don't try doing this with AfterStep. AfterStep has a bad habit of filling up your entire colour table. I highly recommend using **WindowMaker**; apart from being one of the best managers, it can be configured to use as little as 4 colours (although I recommend using the standard 76) it also 'dithers' images to stay within the colour limit. This still leaves you with 180 colours to work with.

2. You'll need the following applications:

For Mac:

Penguin 16 **Must be version 16 or later**

Penguin Colors

ResEdit

GraphicConverter

Photoshop Clut Convertor

For Linux:

xwd - this should already be in /usr/X11R6/bin/

3. In Linux, make sure your window-manager is configured exactly as you want it. Restart X to clear the colour table.

4. Next launch the application you use most often or the most colour intensive (e.g GIMP)

5. In an xterm, type: *xwd -out screenshot.xwd* your pointer should turn into a cross-hair, click on the desktop and wait a couple of seconds. You just took a picture of your screen.

6. You should now have a 'screenshot.xwd' file in the current directory. Mount you Macintosh partition and copy the file into it.

7. Reboot

8. Open the screenshot.xwd file in GraphicConverter. WOW! So that's what it looks like! Save the file in 'Color Table' format (screenshot.PAL).

9. Launch 'Photoshop Clut Convertor', select the screenshot.PAL file and save it.

10. Open 'clut resource' file in ResEdit. Double click on 'clut', then '128 2056'. Select all then copy the colours.

11. Open the 'Penguin Colours' file in ResEdit. Double click on 'clut' then '8 2056'. Select all, clear the old colours then paste in the new ones.

12. Save the Penguin Colours file and make sure it's in the same directory as Penguin 16.

13. Boot Linux. Huh? All my console colours are funny? Remember: You are now using a palette optimised for X - not console.

14. Start X. Ta Daa! Hopefully you should be viewing your window-manager in glorious, perfect colour. Now launch your favourite app, it should be lookin' good too!

15. X is good but those console colours are bugging me! If you want the good 'ol black and white console. Change the following colours in ResEdit (I recommend you copy and paste them from an original copy of 'Penguin Colours' to avoid freezing up):

1 to black (background colour)

8 to white (text colour)

16 to white (cursor colour)

Note: You are sacrificing 3 important X-Colours by doing this.

This is by no means a solution to the problem - just a quick-fix way of making X more pleasant to look at. Enjoy!

Extra Notes:

* If you want to use WindowMaker, go to your nearest Debian mirror and download the 68k Binaries (Read the 'packages.txt' file to see what you need) You will also need the 'libwraster0_0.14.1-7.deb' file from /lib.

* I am currently using WindowMaker, xterm apps and GIMP with no trouble whatsoever. If I feel like playing Abuse, I load a different colour file at startup.

* Have a look at my screenshot!

Written by Anthony Super on the 9/1/99. E-Mail me at atsuper@iname.com

International Color Consortium ®

Specification ICC.1:2003-09

File Format for Color Profiles (Version 4.1.0)

[REVISION of ICC.1:2001-12]

Copyright notice

Licenses and trademarks

For additional information on the ICC

Visit the ICC Web site: **http://www.color.org**

0 Introduction

This specification describes the International Color Consortium® profile format. The intent of this format is to provide a cross-platform device profile format. Such device profiles can be used to translate color data created on one device into another device's native color space. The acceptance of this format by operating system vendors allows end users to transparently move profiles and images with embedded profiles between different operating systems. For example, this allows a printer manufacturer to create a single profile for multiple operating systems.

A large number of companies and individuals from a variety of industries participated in very extensive discussions on these issues. Many of these discussions occurred under the auspices of Forschungsgesellschaft Druck e.V. (FOGRA), the German graphic arts research institute, during 1993. The present specification evolved from these discussions and the ColorSync™ 1.0 profile format.

This is a very complex set of issues and the organization of this document strives to provide a clear, clean, and unambiguous explanation of the entire format. To accomplish this, the overall presentation is from a top-down perspective, beginning with a summary overview and continuing down into more detailed specifications to a byte stream description of format.

0.1 Intended audience

This specification is designed to provide developers and other interested parties a clear description of the profile format. A nominal understanding of color science is assumed, such as familiarity with the CIELAB color space, general knowledge of device characterizations, and familiarity with at least one operating system level color management system.

0.2 Organizational description of this specification

This specification is organized into a number of major clauses and annexes. Each clause and subclause is numbered for easy reference. A brief introduction is followed by a detailed summary of the issues involved in this document including: International Color Consortium, device profiles, profile element structure, embedded profiles, registration authority, and color model arbitration.

Clause 1 describes the scope of this specification.

Clause 2 provides the normative references for this specification.

Clause 3 describes the conformance requirements for this specification.

Clause 4 provides general definitions used within this specification.

Clause 5 provides descriptions of notations, symbols and abbreviations used in this specification.

Clause 6 describes the requirements of this specification. 6.1 describes the format header definition. 6.2 describes the tag table. 6.3 provides a top level view of what tags are required for each type of profile classification and a brief description of the algorithmic models associated with these classes. Four additional color transformation formats are also described: DeviceLink, color space conversion, abstract transformations, and named color transforms. 6.4 is a detailed algorithmic and intent description of all of the tagged elements described in the previous clauses. 6.5 provides a byte stream definition of the structures that make up the tags in 6.4.

Annex A describes the color spaces used in this specification. Annex B provides the necessary details to embed profiles into PICT, EPS, TIFF, and JPEG files. Annex C provides a general description of the Post-

Script Level 2 tags used in this specification. Annex D is a paper describing details of the profile connection space. Annex E provides details on the chromaticAdaptationTag. Annex F is a summary of the changes made in the last few revisions of the spec. The C language ICC header file in previous versions of the specification has been removed as an appendix. It is available on the ICC web site as file ICC.3.

0.3 International Color Consortium

Considering the potential impact of this specification on various industries, a consortium has been formed that will administer this specification and the registration of tag signatures and descriptions. The founding members of this consortium are: Adobe Systems Inc., Agfa-Gevaert N.V., Apple Computer, Inc., Eastman Kodak Company, FOGRA (Honorary), Microsoft Corporation, Silicon Graphics, Inc., Sun Microsystems, Inc., and Taligent, Inc. (resigned). These companies have committed to fully support this specification in their operating systems, platforms and applications. The consortium has since been expanded and now has over 60 members. See the ICC web site for information on how to become a member.

0.4 Device profiles

Device profiles provide color management systems with the information necessary to convert color data between native device color spaces and device independent color spaces. This specification divides color devices into three broad classifications: input devices, display devices and output devices. For each device class, a series of base algorithmic models are described which perform the transformation between color spaces. These models provide a range of color quality and performance results. Each of the base models provides different trade-offs in memory footprint, performance and image quality. The necessary parameter data to implement these models is described in the required portions on the appropriate device profile descriptions. This required data provides the information for the color management framework default color management module (CMM) to transform color information between native device color spaces. A representative architecture using these components is illustrated in Figure 1 below.

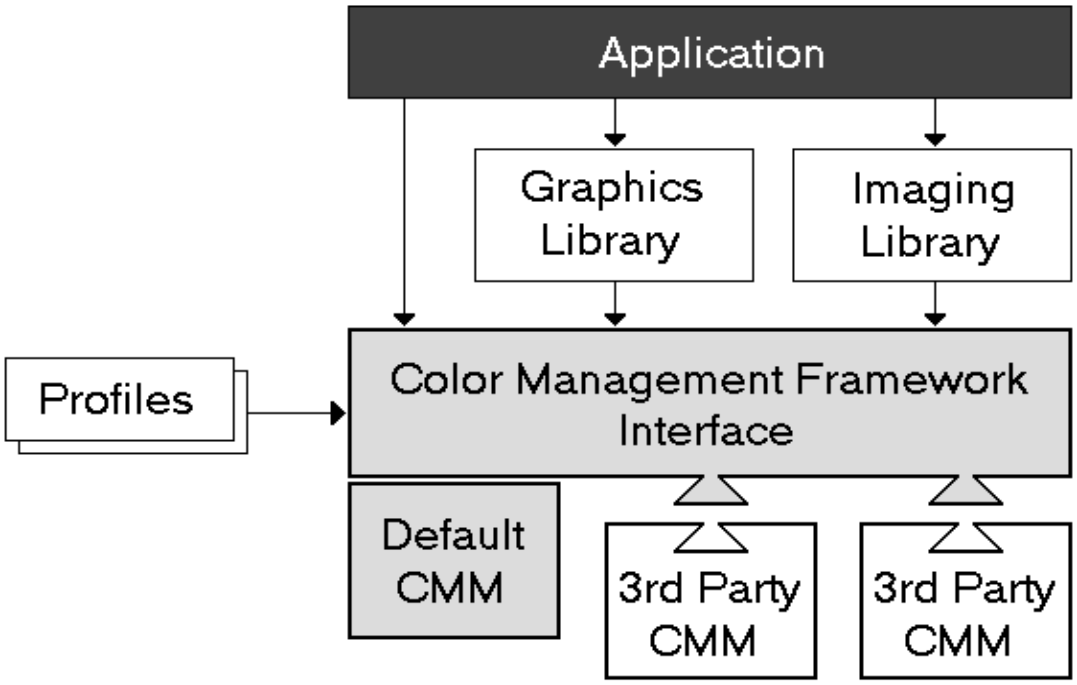


Figure 1 — Color management architecture

0.5 Profile element structure

The profile structure is defined as a header followed by a tag table followed by a series of tagged elements that can be accessed randomly and individually. This collection of tagged elements provides three levels of information for developers: required data, optional data and private data. An element tag table provides a table of contents for the tagging information in each individual profile. This table includes a tag signature, the beginning address offset and size of the data for each individual tagged element. Signatures in this specification are defined as a four-byte hexadecimal number. This tagging scheme allows developers to read in the element tag table and then randomly access and load into memory only the information necessary to their particular software application. Since some instances of profiles can be quite large, this provides significant savings in performance and memory. The detailed descriptions of the tags, along with their intent, are included later in this specification.

The required tags provide the complete set of information necessary for the default CMM to translate color information between the profile connection space and the native device space. Each profile class determines which combination of tags is required. For example, a multi-dimensional lookup table is required for output devices, but not for display devices.

In addition to the required tags for each device profile, a number of optional tags are defined that can be used for enhanced color transformations. Examples of these tags include calibration support, and others. In the case of required and optional tags, all of the signatures, an algorithmic description, and intent are registered with the International Color Consortium.

Private data tags allow CMM developers to add proprietary value to their profiles. By registering just the tag signature and tag type signature, developers are assured of maintaining their proprietary advantages while maintaining compatibility with this specification. However, the overall philosophy of this format is to maintain an open, cross-platform standard, therefore the use of private tags should be kept to an absolute minimum.

0.6 Embedded profiles

In addition to providing a cross-platform standard for the actual disk-based profile format, this specification also describes the convention for embedding these profiles within graphics documents and images. Embedded profiles allow users to transparently move color data between different computers, networks and even operating systems without having to worry if the necessary profiles are present on the destination systems. The intention of embedded profiles is to allow the interpretation of the associated color data. Embedding specifications are described in Annex B of this document.

0.7 Registration authority

This specification requires that signatures for CMM type, device manufacturer, device model, profile tags and profile tag types be registered to insure that all profile data is uniquely defined. The registration authority for these data is the ICC Technical Secretary. See the ICC Web Site (www.color.org) for contact information.

If and when this document becomes an International Standard this registration responsibility must be brought into conformance with ISO procedures. These procedures are being investigated on behalf of ICC and TC130.

0.8 Redundant data arbitration

There are several methods of color rendering described in the following structures that can function within a single CMM. If data for more than one method are included in the same profile, the following selection algorithm should be used by the software implementation.

For profile types input, display, output, or color space, the priority of the tag usage for each rendering intent is:

1. BToA0Tag, BToA1Tag, BToA2Tag, AToB0Tag, AToB1Tag, or AToB2Tag designated for the rendering intent
2. BToA0Tag or AToB0Tag
3. TRC's (redTRCTag, greenTRCTag, blueTRCTag, or grayTRCTag) and colorants (redMatrixColumnTag, greenMatrixColumnTag, blueMatrixColumnTag)

The available valid tag with the lowest number defines the transform.

1 Scope

This specification defines the data necessary to describe the color characteristics used to input, display, or output images, and an associated file format for the exchange of this data.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this specification. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of ISO and IEC maintain registers of currently valid International Standards.

CIE Publication 15.2-1986, "Colorimetry, Second Edition"

CIE Publ. 122-1996 The Relationship between Digital and Colorimetric Data for Computer-Controlled CRT Displays

EBU Tech. 3213-E: EBU standard for chromaticity tolerances for studio monitors

EC/CD 61966-2.1: Colour measurement and management in Multimedia systems and equipment - Part 2.1: Colour management in multimedia systems - Default RGB colour space - sRGB

IEC/CD 61966-3: Colour measurement and management in multimedia systems and equipment - Part 3: Equipment using cathode ray tubes

ISO 5-1:1984, "Photography -- Density measurements -- Part 1: Terms, symbols and notations"

ISO 5-2:1991, "Photography -- Density measurements -- Part 2: Geometric conditions for transmission density"

ISO 5-4:1995, "Photography -- Density measurements -- Part 4: Geometric conditions for reflection density"

ISO/IEC 646:1991, "Information technology -- ISO 7-bit coded character set for information interchange"

ISO 3664:1975, “Photography -- Illumination conditions for viewing colour transparencies and their reproductions”

ISO/IEC 8824-1:1995, “Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation”

ISO/IEC 10918-1:1994, “Information technology -- Digital compression and coding of continuous-tone still images: Requirements and guidelines”

ISO/DIS 12234, “Photography -- Electronic still picture cameras -- Removeable memory (TIFF/EP)”

ISO/FDIS 12639, “Graphic Technology -- Prepress digital data exchange -- Tag image file format for image technology (TIFF/IT)”

ISO 12641:1997, “Graphic technology -- Prepress digital data exchange -- Colour targets for input scanner calibration”

ISO 12642:1996, “Graphic technology -- Prepress digital data exchange -- Input data for characterization of 4-colour process printing”

ISO 13655:1996, “Graphic technology -- Spectral measurement and colorimetric computation for graphic arts images”

ITU-R BT.709-2, Parameter values for the HDTV standards for production and international programme exchange

PICT Standard Specifications, published by Apple Computer, Inc.

PostScript Language Reference Manual, Third Edition, Adobe Systems Inc.

SMPTE RP 145-1994: SMPTE C Color Monitor Colorimetry

TIFF 6.0 Specification, published by Adobe Systems Incorporated.

3 Conformance

Any color management system, application, utility or device driver that is in conformance with this specification shall have the ability to read the profiles as they are defined in this specification. Any profile-generating software and/or hardware that is in conformance with this specification shall have the ability to create profiles as they are defined in this specification. ICC conforming software will use the ICC profiles in an appropriate manner.

4 Definitions

For the purposes of this specification, the following definitions shall apply:

4.1
aligned
A data element is aligned with respect to a data type if the address of the data element is an integral multiple of the number of bytes in the data type.

4.2
ASCII string
A sequence of bytes, each containing a graphic character from ISO/IEC 646, the last character in the string being a NULL (character 0/0).

4.3
big-endian
Addressing the bytes within a 16, 32 or 64-bit value from the most significant to the least significant, as the byte address increases.

4.4
bit position
Bits are numbered such that bit 0 is the least significant bit.

4.5
byte
An 8-bit unsigned binary integer.

4.6
byte offset
The number of bytes from the beginning of a field.

4.7
fixed point representation
A method of encoding a real number into binary by putting an implied binary point at a fixed bit position. See Table 3 in 5.3.3 for an example.

Many of the tag types contain fixed point numbers. Several references can be found (MetaFonts, etc.) illustrating the preferability of fixed point representation to pure floating point representation in very structured circumstances.

4.8
NULL
The character coded in position 0/0 of ISO/IEC 646.

4.9
long word
A 32-bit quantity.

4.10
profile connection space (PCS)
An abstract color space used to connect the source and destination profiles. See A.1 for a full description.

4.11
rendering intent
A particular gamut mapping style or method of converting colors in one gamut to colors in another gamut. See Annex A for a more complete description of the rendering intents used in ICC profiles.

4.12
signature
An alphanumerical 4-byte value, registered with the ICC. Shorter values are padded at the end with 20h bytes.

5 Notation, symbols and abbreviations

5.1 Notations

All numeric values in this specification are expressed in decimal, unless otherwise indicated. A letter “h” is suffixed to denote a hexadecimal value.

Literal strings are denoted in this specification by enclosing them in double quotation marks.

5.2 Symbols and abbreviations

The following symbols and abbreviations are used within this specification with the meanings indicated:

ANSI	American National Standards Institute
CIE	*Commission Internationale de l’Éclairage* (International Commission on Illumination)
CLUT	color Lookup Table (multidimensional)
CMM	color Management Module
CMS	color Management System
CMY	Cyan, Magenta, Yellow
CMYK	Cyan, Magenta, Yellow, Key (black)
CRD	color Rendering Dictionary
CRT	Cathode-Ray Tube
EPS	Encapsulated PostScript
ICC	International Color Consortium
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
LCD	Liquid Crystal Display
LUT	Lookup Table
PCS	Profile Connection Space
PPD	PostScript Printer Description
RGB	Red, Green, Blue
TIFF	Tagged Image File Format
TRC	Tone Reproduction Curve

5.3 Basic numeric types

5.3.1 dateTimeNumber

A 12-byte value representation of the time and date. The actual values are encoded as 16-bit unsigned integers.

Table 1 — dateTimeNumber

Byte Offset	Content	Encoded as...
0..1	number of the year (actual year, e.g. 1994)	uInt16Number
2..3	number of the month (1-12)	uInt16Number
4..5	number of the day of the month (1-31)	uInt16Number
6..7	number of hours (0-23)	uInt16Number
8..9	number of minutes (0-59)	uInt16Number
10..11	number of seconds (0-59)	uInt16Number

All the dateTimeNumber values in a profile shall be in Coordinated Universal Time (UTC, also known as GMT or ZULU Time). Profile writers are required to convert local time to UTC when setting these values. Programs that display these values may show the dateTimeNumber as UTC, show the equivalent local time (at current locale), or display both UTC and local versions of the dateTimeNumber.

5.3.2 response16Number

This type is used to associate a normalized device code with a measurement value.

Table 2 — response16Number

Byte Offset	Content	Encoded as...
0..1	16-bit number encoding the interval [DeviceMin to DeviceMax] with DeviceMin encoded as 0000h and DeviceMax encoded as FFFFh	uInt16Number
2..3	reserved, must be zero	
4..7	measurement value	s15Fixed16Number

5.3.3 s15Fixed16Number

This type represents a fixed signed 4-byte/32-bit quantity which has 16 fractional bits. An example of this encoding is:

Table 3 — s15Fixed16Number

-32768,0	80000000h
0	00000000h
1,0	00010000h
32767 + (65535/65536)	7FFFFFFFh

5.3.4 u16Fixed16Number

This type represents a fixed unsigned 4-byte/32-bit quantity which has 16 fractional bits. An example of this encoding is:

Table 4 — u16Fixed16Number

0	00000000h
1,0	00010000h
65535 + (65535/65536)	FFFFFFFFh

5.3.5 u8Fixed8Number

This type represents a fixed unsigned 2-byte/16-bit quantity which has 8 fractional bits. An example of this encoding is:

Table 5 — u8Fixed8Number

0	0000h
1,0	0100h
255 + (255/256)	FFFFh

5.3.6 uInt16Number

This type represents a generic unsigned 2-byte/16-bit quantity.

5.3.7 uInt32Number

This type represents a generic unsigned 4-byte/32-bit quantity.

5.3.8 uInt64Number

This type represents a generic unsigned 8-byte/64-bit quantity.

5.3.9 uInt8Number

This type represents a generic unsigned 1-byte/8-bit quantity.

5.3.10 XYZNumber

This type represents a set of three fixed signed 4-byte/32-bit quantities used to encode CIEXYZ tristimulus values where byte usage is assigned as follows:

Table 6 — XYZNumber

Byte Offset	Content	Encoded as...
0..3	CIE X	s15Fixed16Number
4..7	CIE Y	s15Fixed16Number
8..11	CIE Z	s15Fixed16Number

All XYZNumbers (other than those specifying luminance) are scaled such that Y is specified over the range of 0 to 1,0. Tristimulus values must be non-negative.

5.3.11 Seven-bit ASCII

Table 7 — Hexadecimal

Hexadecimal															
00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	“	23	#	24	$	25	%	26	&	27	‘
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

Table 8 — Decimal

Decimal															
0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	“	35	#	36	$	37	%	38	&	39	‘
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	del

6 Requirements

An ICC profile shall include the following elements, in the order shown below in Figure 2, as a single file.

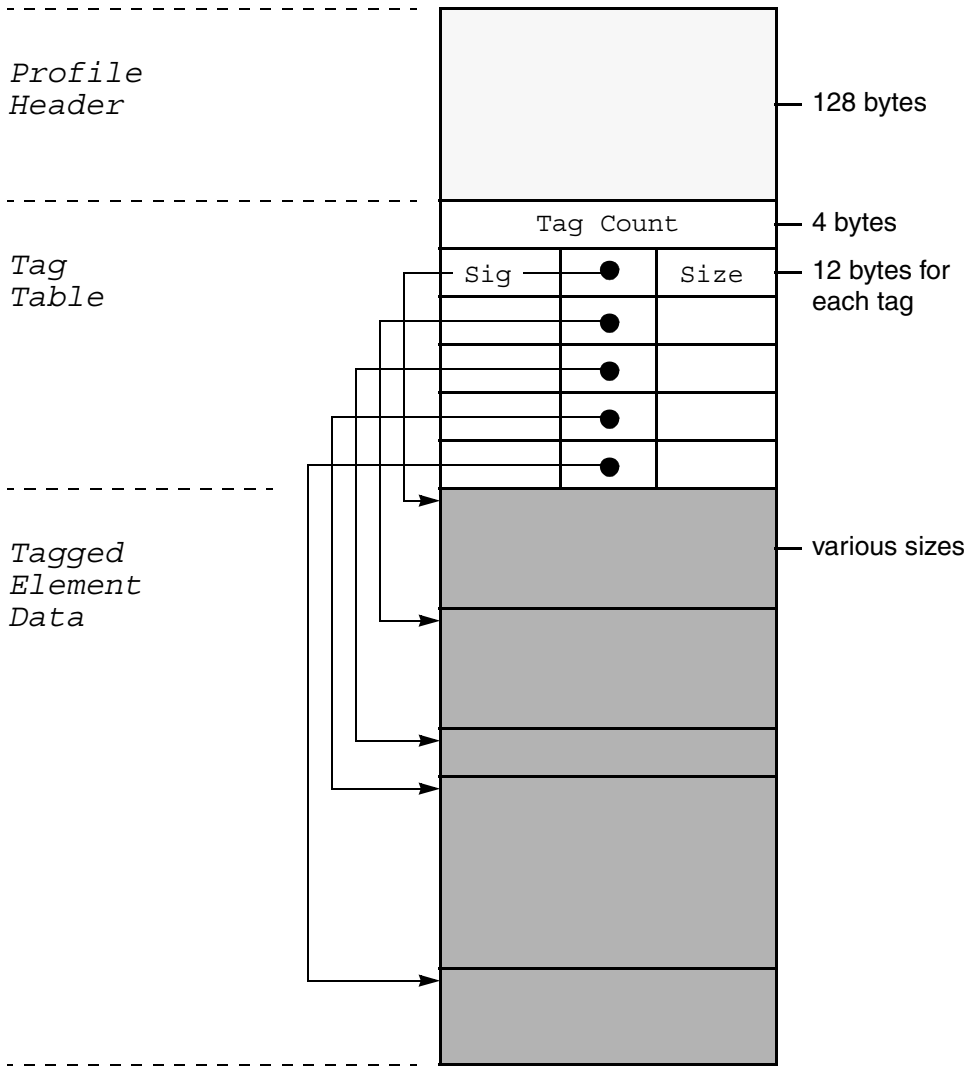


Figure 2 — Profile Map

First, the 128-byte file header as defined in 6.1.

Second, the tag table as defined in 6.2.

Third, the tagged element data in accordance with the requirements of 6.3, 6.4 and 6.5.

There are additional requirements on the structure of the profile.

1) The first tagged element data must immediately follow the tag table.
2) All tagged element data, including the last, must be padded by no more than three following pad bytes to reach a 4-byte boundary.
3) All pad bytes must be NULL.

4) The profile size value in the header of the profile must be the exact size obtained by combining the profile header, the tag table, and the tagged element data, including the pad bytes for the last tag. This implies that the length must be a multiple of four.

The above restrictions result in two key benefits. First, the likelihood of two profiles which contain the same tag data yet have different checksum values is reduced. Second, all profiles are reduced to a minimum size.

The information necessary to understand and create the tagged element data is arranged within this specification as follows. Each class, and subclass, of device (e.g.: input, RGB) requires the use of specific tags and allows other optional tags. These relationships are described in 6.3. Tags themselves are described in 6.4. However tag descriptions draw upon a series of commonly used “tag types” which are defined in 6.5. The definition of the basic number types used for data encoding are found in 5.3.

All profile data must be encoded as big-endian.

All color spaces used in this specification shall be in accordance with Annex A.

6.1 Header description

The profile header provides the necessary information to allow a receiving system to properly search and sort ICC profiles. Table 9 gives the byte position, content and encoding of the profile header.

This header provides a set of parameters at the beginning of the profile format. For color space conversion and abstract profiles, the device profile dependent fields are set to zero if irrelevant. Having a fixed length header allows for performance enhancements in the profile searching and sorting operations.

Table 9 — Header (Part 1 of 2)

Byte Offset	Content	Encoded as...
0..3	Profile size	uInt32Number
4..7	CMM Type signature	see below
8..11	Profile version number	see below
12..15	Profile/Device Class signature	see below
16..19	Color space of data (possibly a derived space) [i.e. “the canonical input space”]	see below
20..23	Profile Connection Space (PCS) [i.e. “the canonical output space”]	see below
24..35	Date and time this profile was first created	dateTimeNumber
36..39	‘acsp’ (61637370h) profile file signature	
40..43	Primary Platform signature	see below
44..47	Flags to indicate various options for the CMM such as distributed processing and caching options	see below
48..51	Device manufacturer of the device for which this profile is created	see below
52..55	Device model of the device for which this profile is created	see below

Table 9 — Header (Part 2 of 2)

56..63	Device attributes unique to the particular device setup such as media type	see below
64..67	Rendering Intent	see below
68..79	The XYZ values of the illuminant of the Profile Connection Space. This must correspond to D50. It is explained in more detail in A.1.	XYZNumber
80..83	Profile Creator signature	see below
84..99	Profile ID	see below
100..127	28 bytes reserved for future expansion - must be set to zeros	

6.1.1 Profile size

The total size of the profile in bytes.

6.1.2 CMM Type signature

Identifies the preferred CMM to be used. The signatures must be registered in order to avoid conflicts (see 0.7). If no CMM is preferred, this field should be set to zero.

6.1.3 Profile version

Profile version number where the first 8 bits identify the major revision and the next 8 bits identify the minor revision and bug fix revision. The major and minor revision are set by the International Color Consortium and will match up with editions of this specification.

The current profile version number is "4.1.0" (encoded as 04000000h).

The encoding is such that:

Table 10 — Profile version

Byte Offset	Content
0	Major Revision in Binary-Coded Decimal
1	Minor Revision & Bug Fix Revision in each nibble in Binary-Coded Decimal
2	reserved, must be set to 0
3	reserved, must be set to 0

A major revision change will only happen when a profile specification change requires that all CMMs be upgraded in order to correctly use the profile. For example, the addition of new required tags would cause the major revision to change. A minor version change will occur when new profiles can still be used by existing CMMs. For example, the addition of new optional tags would cause the minor revision to change, because existing CMMs will be able to process the profiles correctly while ignoring the new tags.

6.1.4 Profile/Device Class signature

There are three basic classes of device profiles: Input, Display and Output profiles.

Within each of these classes there can be a variety of subclasses, such as RGB scanners, CMYK scanners and many others. These basic classes have the following signatures:

Table 11 — Device class

Device Class	Signature	Hex Encoding
Input Device profile	‘scnr’	73636E72h
Display Device profile	‘mntr’	6D6E7472h
Output Device profile	‘prtr’	70727472h

In addition to the three basic device profile classes, four additional color processing profiles are defined. These profiles provide a standard implementation for use by the CMM in general color processing or for the convenience of CMMs which may use these types to store calculated transforms. These four profile classes are: DeviceLink, color space conversion, abstract, and named color profiles.

DeviceLink profiles provide a mechanism in which to save and store a series of device profiles and non-device profiles in a concatenated format as long as the series begins and ends with a device profile. This is useful for workflows where a combination of device profiles and non-device profiles are used repeatedly.

ColorSpace Conversion profiles are used as a method for CMMs to convert between different non-device color spaces.

The Abstract color profiles provide a generic method for users to make subjective color changes to images or graphic objects by transforming the color data within the PCS.

Named Color profiles can be thought of as sibling profiles to device profiles. For a given device there would be one or more device profiles to handle process color conversions and one or more named color profiles to handle named colors. There might be multiple named color profiles to account for different consumables or multiple named color vendors.

These profiles have the following signatures:

Table 12 — Profile class

Profile Class	Signature	Hex Encoding
DeviceLink profile	‘link’	6C696E6Bh
ColorSpace Conversion profile	‘spac’	73706163h
Abstract profile	‘abst’	61627374h
Named Color profile	‘nmcl’	6E6D636Ch

6.1.5 Color Space signature

The encoding is such that:

Table 13 — Color space signature

Color Space	Signature	Hex Encoding
XYZData	‘XYZ ’	58595A20h
labData	‘Lab ’	4C616220h
luvData	‘Luv ’	4C757620h
YCbCrData	‘YCbr ’	59436272h
YxyData	‘Yxy ’	59787920h
rgbData	‘RGB ’	52474220h
grayData	‘GRAY’	47524159h
hsvData	‘HSV ’	48535620h
hlsData	‘HLS ’	484C5320h
cmykData	‘CMYK’	434D594Bh
cmyData	‘CMY ’	434D5920h
2colorData	‘2CLR’	32434C52h
3colorData (if not listed above)	‘3CLR’	33434C52h
4colorData (if not listed above)	‘4CLR’	34434C52h
5colorData	‘5CLR’	35434C52h
6colorData	‘6CLR’	36434C52h
7colorData	‘7CLR’	37434C52h
8colorData	‘8CLR’	38434C52h
9colorData	‘9CLR’	39434C52h
10colorData	‘ACLR’	41434C52h
11colorData	‘BCLR’	42434C52h
12colorData	‘CCLR’	43434C52h
13colorData	‘DCLR’	44434C52h
14colorData	‘ECLR’	45434C52h
15colorData	‘FCLR’	46434C52h

6.1.6 Profile Connection Space signature

The encoding is such that:

Table 14 — Profile connection space signature

Profile Connection Color Space	Signature	Hex Encoding
XYZData	‘XYZ ’	58595A20h
labData	‘Lab ’	4C616220h

When the profile is a DeviceLink profile, the Profile Connection Space Signature can be any of the signatures in the Color Space Signatures table. (See 6.1.5)

6.1.7 Primary Platform signature

Signature to indicate the primary platform/operating system framework for which the profile was created.

The encoding is such that:

Table 15 — Primary platform signature

Primary Platform	Signature	Hex Encoding
Apple Computer, Inc.	‘APPL’	4150504Ch
Microsoft Corporation	‘MSFT’	4D534654h
Silicon Graphics, Inc.	‘SGI ’	53474920h
Sun Microsystems, Inc.	‘SUNW’	53554E57h
Taligent, Inc.	‘TGNT’	54474E54h

If there is no primary platform, this field should be set to zero.

6.1.8 Profile flags

Flags to indicate various hints for the CMM such as distributed processing and caching options. The least-significant 16 bits are reserved for the ICC.

The encoding is such that:

Table 16 — Profile flags

Flags	Bit Position
Embedded Profile (0 if not embedded, 1 if embedded in file)	0
Profile cannot be used independently from the embedded color data (set to 1 if true, 0 if false)	1

6.1.9 Device manufacturer and model signatures

The signatures for various manufacturers and models are listed in a separate document (ICC Signatures). New signatures must be registered with the ICC (see 0.7).

6.1.10 Attributes

Attributes unique to the particular device setup such as media type. The least-significant 32 bits of this 64-bit value are reserved for the ICC.

The encoding is such that (with “on” having value 1 and “off” having value 0):

Table 17 — Header attributes

Attribute	Bit Position
Reflective (off) or Transparency (on)	0
Glossy (off) or Matte (on)	1
Positive (off) or negative (on) media	2
Color (off) or black & white (on) media	3

Notice that bits 0, 1, 2, and 3 describe the media, not the device. For example, a profile for a color scanner that has been loaded with black & white film will have bit 3 set on, regardless of the colorspace of the scanner (see 6.1.5).

If the media is not inherently "color" or "black & white" (such as the paper in an inkjet printer), the reproduction takes on the property of the device. Thus, an inkjet printer loaded with a color ink cartridge can be thought to have "color" media.

6.1.11 Rendering intent

Perceptual, media-relative colorimetric, saturation and ICC-absolute colorimetric are the four intents required to be supported. The least-significant 16 bits are reserved for the ICC.

The encoding is such that:

Table 18 — Header rendering intents

Rendering Intent	Value
Perceptual	0
Media-Relative Colorimetric	1
Saturation	2
ICC-Absolute Colorimetric	3

The rendering intent specifies the style of reproduction which should be used (or, in the case of a DeviceLink profile, was used) when this profile is (was) combined with another profile. In a sequence of profiles, it applies to the combination of this profile and the next profile in the sequence and not to the entire sequence. Typically, the user or application will set the rendering intent dynamically at runtime or embedding time. Therefore, this flag may not have any meaning until the profile is used in some context, e.g in a DeviceLink or an embedded source profile.

The field is a uInt32Number in which the least-significant 16 bits are used to encode the rendering intent. The most significant 16 bits should be set to 0.

6.1.12 Profile Creator signature

Identifies the creator of the profile. The signatures are from the group of signatures used for the device manufacturer field.

6.1.13 Profile ID

This field is optional but should be used to record a checksum value which if used shall be generated using the MD5 fingerprinting method as defined in RFC 1321. The entire profile, based on the size field in the header, shall be used to calculate the ID after the values in the 6.1.8 Profile flags field (Bytes 44 to 47), 6.1.11 Rendering intent field (Bytes 64 to 67), and 6.1.13 Profile ID field (Bytes 84 to 99) fields in the profile header have been temporarily replaced with zeros. If a profile ID has not been calculated the value of the field shall be set to zero.

6.2 Tag table definition

The tag table acts as a table of contents for the tags and tag element data in the profiles. The first four bytes contain a count of the number of tags in the table itself. The tags within the table are not required to be in any particular order.

Each tag signature in the tag table must be unique; a profile cannot contain more than one tag with the same signature.

Individual tag structures within the Tag Table:

Table 19 — Tag table structure

Byte Offset	Content	Encoded as...
0..3	Tag Signature	
4..7	Offset to beginning of tag data	uInt32Number
8..11	Element Size	uInt32Number

6.2.1 Tag signature

A four-byte value registered with the ICC (see 0.7).

6.2.2 Offset

The address of the tag data element. This is the number of bytes from the beginning of the profile data stream (i.e. the offset to the first byte in the profile header is 0). For profiles that are not embedded in images, this number is the same as the file offset.

All tag data is required to start on a 4-byte boundary (relative to the start of the profile data stream) so that a tag starting with a 32-bit value will be properly aligned without the tag handler needing to know the contents of the tag. This means that the least-significant two bits of each offset must be zero.

6.2.3 Element size

The number of bytes in the tag data element. The element size must be for the actual data and must not include any padding at the end of the tag data. An element may have any size (up to the limit imposed by the 32-bit offsets).

6.3 Required tags for profiles

This clause provides a top level view of what tags are required for each type of profile classification and a brief description of the algorithmic models associated with these classes. A general description for each tag is included in this clause.

Note that these descriptions assume two things; every profile contains a header, and may include additional tags beyond those listed as required in this clause. The explicitly listed tags are those which are required in order to comprise a legal profile of each type.

In general, multi-dimensional tables refer to lookup tables with more than one input component.

The intent of requiring tags with profiles is to provide a common base level of functionality. If a custom CMM is not present, then the default CMM will have enough information to perform the requested color transformations. The particular models implied by the required data are also described below. While this data might not provide the highest level of quality obtainable with optional data and private data, the data provided is adequate for sophisticated device modeling.

In the following tables, the term "undefined" means that the use of the tag in that situation is not specified by the ICC. The ICC recommends that such tags not be included in profiles. If the tag is present, its use is implementation dependent. In general, the BToAxTags represent the inverse operation of the AToBxTags.

Note that AToB1Tag and BToA1Tag are used to provide both of the colorimetric intents.

Note that tags may reference the same tag data but that this may not be suitable in many cases.

If an optional transformation tag is not present, see Section 0.8.

The interpretation of some tags are context dependent. This dependency is described below in Table 20

Table 20 — Profile type/profile tag and defined rendering intents

Profile Class	**AToB0Tag**	**AToB1Tag**	**AToB2Tag**	**TRC/ matrix**	**BToA0Tag**	**BToA1Tag**	**BToA2Tag**
Input	Device to PCS: perceptual	Device to PCS: colorimetric	Device to PCS: saturation	colorimetric	PCS to Device: perceptual	PCS to Device: colorimetric	PCS to Device: saturation
Display	Device to PCS: perceptual	Device to PCS: colorimetric	Device to PCS: saturation	colorimetric	PCS to Device: perceptual	PCS to Device: colorimetric	PCS to Device: saturation
Output	Device to PCS: perceptual	Device to PCS: colorimetric	Device to PCS: saturation	colorimetric	PCS to Device: perceptual	PCS to Device: colorimetric	PCS to Device: saturation
Color-Space	ColorSpace to PCS: perceptual	ColorSpace to PCS: colorimetric	ColorSpace to PCS: saturation	undefined	PCS to Color-Space: perceptual	PCS to Color-Space: colorimetric	PCS to Color-Space: saturation
Abstract	PCS to PCS	undefined	undefined	undefined	undefined	undefined	undefined
DeviceLink	Device1 to Device2 rendering intent defined according to Table 9	undefined	undefined	undefined	undefined	undefined	undefined
Named Color	undefined	undefined	undefined	undefined	undefined	undefined	undefined

6.3.1 Input Profile

This profile represents input devices such as scanners and digital cameras.

6.3.1.1 Monochrome Input Profiles

Table 21 — Monochrome input profile required tags

Tag Name	**General Description**
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve (TRC)
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The mathematical model implied by this data is:

$$connection = \text{grayTRC}[device] \quad (1)$$

This represents a simple tone reproduction curve adequate for most monochrome input devices. The *connection* values in this equation should represent the achromatic channel of the profile connection space in the range of 0 to 1,0 where 0 represents black and 1,0 represents white. The PCS value is derived by

multiplying the D50 white point by the normalized TRC value between 0 and 1,0. If the inverse of this is desired, then the following equation is used:

$$device = \text{grayTRC}^{-1}[connection] \tag{2}$$

AToB0Tag, AToB1Tag, AToB2Tag, BToA0Tag, BToA1Tag, BToA2Tag may be included in monochrome profiles. If these are present, their usage shall be as defined in Table 20.

6.3.1.2 Three-component Matrix-based Input Profiles

This profile type is often used with devices whose nominal color space is RGB. .

Table 22 — Three-component matrix-based input profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
redMatrixColumnTag	The first column in the matrix used in TRC/matrix transforms (This column is combined with the linear red channel during the matrix multiplication)
greenMatrixColumnTag	The second column in the matrix used in TRC/matrix transforms (This column is combined with the linear green channel during the matrix multiplication)
blueMatrixColumnTag	The third column in the matrix used in TRC/matrix transforms (This column is combined with the linear blue channel during the matrix multiplication)
redTRCTag	Red channel tone reproduction curve
greenTRCTag	Green channel tone reproduction curve
blueTRCTag	Blue channel tone reproduction curve
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

This model describes transformation from device color space to PCS (see Annex A for PCS description). The transformation is based on three non-interdependent per-channel tone reproduction curves to convert between non-linear and linear RGB values and a 3x3 matrix to convert between linear RGB values and relative XYZ values. The mathematical model implied by this data is:

$$\begin{aligned} linear_r &= \text{redTRC}[device_r] \\ linear_g &= \text{greenTRC}[device_g] \\ linear_b &= \text{blueTRC}[device_b] \end{aligned} \tag{3}$$

$$\begin{bmatrix} connection_X \\ connection_Y \\ connection_Z \end{bmatrix} = \begin{bmatrix} redMatrixColumn_X & greenMatrixColumn_X & blueMatrixColumn_X \\ redMatrixColumn_Y & greenMatrixColumn_Y & blueMatrixColumn_Y \\ redMatrixColumn_Z & greenMatrixColumn_Z & blueMatrixColumn_Z \end{bmatrix} \begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix} \quad (4)$$

This represents a simple linearization followed by a linear mixing model. The three tone reproduction curves linearize the raw values with respect to the luminance (Y) dimension of the CIEXYZ encoding of the profile connection space. The 3x3 matrix converts these linearized values into XYZ values for the CIEXYZ encoding of the profile connection space. The inverse model is given by the following equations:

$$\begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix} = \begin{bmatrix} redMatrixColumn_X & greenMatrixColumn_X & blueMatrixColumn_X \\ redMatrixColumn_Y & greenMatrixColumn_Y & blueMatrixColumn_Y \\ redMatrixColumn_Z & greenMatrixColumn_Z & blueMatrixColumn_Z \end{bmatrix}^{-1} \begin{bmatrix} connection_X \\ connection_Y \\ connection_Z \end{bmatrix} \quad (5)$$

$$\begin{aligned} device_r &= \text{redTRC}^{-1}[1](linear_r > 1) \\ device_r &= \text{redTRC}^{-1}[linear_r](0 \le linear_r \le 1) \\ device_r &= \text{redTRC}^{-1}[0](linear_r < 0) \end{aligned} \quad (6)$$

$$\begin{aligned} device_g &= \text{greenTRC}^{-1}[1](linear_g > 1) \\ device_g &= \text{greenTRC}^{-1}[linear_g](0 \le linear_g \le 1) \\ device_g &= \text{greenTRC}^{-1}[0](linear_g < 0) \end{aligned} \quad (7)$$

$$\begin{aligned} device_b &= \text{blueTRC}^{-1}[1](linear_b > 1) \\ device_b &= \text{blueTRC}^{-1}[linear_b](0 \le linear_b \le 1) \\ device_b &= \text{blueTRC}^{-1}[0](linear_b < 0) \end{aligned} \quad (8)$$

Only the CIEXYZ encoding of the profile connection space can be used with matrix/TRC models. This profile may be used for any device which has a three-component color space suitably related to XYZ by this model. An AToB0Tag must be included if the CIELAB encoding of the profile connection space is to be used.

Additional multidimensional tags (AToB0Tag, AToB1Tag, AToB2Tag, BToA0Tag, BToA1Tag, BToA2Tag) may also be included. If these are present, their usage shall be as defined in Table 20.

An additional multidimensional gamut tag (gamutTag) may be included. The usage of this tag is identical as in output profiles (Section 6.3.3.2).

6.3.1.3 N-component LUT-based Input Profiles

Table 23 — N-component LUT-based input profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Device to PCS: 8-bit or 16-bit data
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The AToB0Tag represents a device model that can use a multi-dimensional lookup table. The model is described in detail in the tag descriptions and the general lookup table tag element structures.

Additional multidimensional tags (AToB1Tag, AToB2Tag, BToA0Tag, BToA1Tag, BToA2Tag) may also be included. If these are present, their usage shall be as defined in Table 20.

In addition, a gamutTag may be included. The usage of this tag is identical as in output profiles (Section 6.3.3.2).

6.3.2 Display Profile

This profile represents display devices such as monitors.

6.3.2.1 Monochrome Display Profiles

Table 24 — Monochrome display profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The mathematical model implied by this data is:

$$connection = \text{grayTRC}[device] \tag{9}$$

This represents a simple tone reproduction curve adequate for most monochrome display devices. The *connection* values in this equation should represent the achromatic channel of the profile connection space in the range of 0 to 1,0 where 0 represents black and 1,0 represents white. The PCS value is derived by multiplying the D50 white point by the normalized TRC value between 0 and 1,0. If the inverse of this is desired, then the following equation is used:

$$device = \text{grayTRC}^{-1}[connection] \tag{10}$$

AToB0Tag, AToB1Tag, AToB2Tag, BToA0Tag, BToA1Tag, BToA2Tag may be included in monochrome profiles. If these are present, their usage shall be as defined in Table 20.

6.3.2.2 Three-component Matrix-based Display Profiles

This profile type is often used with devices whose nominal color space is RGB. .

Table 25 — Three-component matrix-based display profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
redMatrixColumnTag	The first column in the matrix used in TRC/matrix transforms (This column is combined with the linear red channel during the matrix multiplication)
greenMatrixColumnTag	The second column in the matrix used in TRC/matrix transforms (This column is combined with the linear green channel during the matrix multiplication)
blueMatrixColumnTag	The third column in the matrix used in TRC/matrix transforms (This column is combined with the linear blue channel during the matrix multiplication)
redTRCTag	Red channel tone reproduction curve
greenTRCTag	Green channel tone reproduction curve
blueTRCTag	Blue channel tone reproduction curve
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

This model describes transformation from device color space to PCS (see Annex A for PCS description). The transformation is based on three non-interdependent per-channel tone reproduction curves to convert between non-linear and linear RGB values and a 3x3 matrix to convert between linear RGB values and relative XYZ values. The mathematical model implied by this data is:

$$\begin{aligned} linear_r &= \text{redTRC}[device_r] \\ linear_g &= \text{greenTRC}[device_g] \\ linear_b &= \text{blueTRC}[device_b] \end{aligned} \tag{11}$$

$$\begin{bmatrix} connection_X \\ connection_Y \\ connection_Z \end{bmatrix} = \begin{bmatrix} redMatrixColumn_X & greenMatrixColumn_X & blueMatrixColumn_X \\ redMatrixColumn_Y & greenMatrixColumn_Y & blueMatrixColumn_Y \\ redMatrixColumn_Z & greenMatrixColumn_Z & blueMatrixColumn_Z \end{bmatrix} \begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix} \tag{12}$$

This represents a simple linearization followed by a linear mixing model. The three tone reproduction curves linearize the raw values with respect to the luminance (Y) dimension of the CIEXYZ encoding of the profile connection space. The 3x3 matrix converts these linearized values into XYZ values for the CIEXYZ encoding of the profile connection space. The inverse model is given by the following equations:

$$\begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix} = \begin{bmatrix} redMatrixColumn_X & greenMatrixColumn_X & blueMatrixColumn_X \\ redMatrixColumn_Y & greenMatrixColumn_Y & blueMatrixColumn_Y \\ redMatrixColumn_Z & greenMatrixColumn_Z & blueMatrixColumn_Z \end{bmatrix}^{-1} \begin{bmatrix} connection_X \\ connection_Y \\ connection_Z \end{bmatrix} \tag{13}$$

$$\begin{aligned} device_r &= \text{redTRC}^{-1}[1](linear_r > 1) \\ device_r &= \text{redTRC}^{-1}[linear_r](0 \le linear_r \le 1) \\ device_r &= \text{redTRC}^{-1}[0](linear_r < 0) \end{aligned} \tag{14}$$

$$\begin{aligned} device_g &= \text{greenTRC}^{-1}[1](linear_g > 1) \\ device_g &= \text{greenTRC}^{-1}[linear_g](0 \le linear_g \le 1) \\ device_g &= \text{greenTRC}^{-1}[0](linear_g < 0) \end{aligned} \tag{15}$$

$$
\begin{aligned}
device_b &= \text{blueTRC}^{-1}[1](linear_b > 1) \\
device_b &= \text{blueTRC}^{-1}[linear_b](0 \le linear_b \le 1) \\
device_b &= \text{blueTRC}^{-1}[0](linear_b < 0)
\end{aligned}
\tag{16}
$$

Only the CIEXYZ encoding of the profile connection space can be used with matrix/TRC models. This profile may be used for any device which has a three-component color space suitably related to XYZ by this model. An AToB0Tag must be included if the CIELAB encoding of the profile connection space is to be used.

Additional multidimensional tags (AToB0Tag, AToB1Tag, AToB2Tag, BToA0Tag, BToA1Tag, BToA2Tag) may also be included. If these are present, their usage shall be as defined in Table 20.

An additional multidimensional gamut tag (gamutTag) may be included. The usage of this tag is identical as in output profiles (Section 6.3.3.2).

6.3.2.3 N-Component LUT-Based Display Profiles

Table 26 — N-component LUT-based display profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Device to PCS: 8-bit or 16-bit data: intent of 0
BToA0Tag	PCS to Device space: 8-bit or 16-bit data: intent of 0
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The AToB0Tag and the BToA0Tag represent a device model that can use a multi-dimensional lookup table. The model is described in detail in the tag descriptions and the general lookup table tag element structures.

Additional multidimensional tags (AToB1Tag, AToB2Tag, BToA1Tag, BToA2Tag) may also be included. If these are present, their usage shall be as defined in Table 20.

An additional multidimensional gamut tag (gamutTag) may be included. The usage of this tag is identical as in output profiles (Section 6.3.3.2).

6.3.3 Output Profile

This profile represents output devices such as printers and film recorders.

6.3.3.1 Monochrome Output Profiles

Table 27 — Monochrome output profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The mathematical model implied by this data is:

$$connection = \text{grayTRC}[device] \tag{17}$$

This represents a simple tone reproduction curve adequate for most monochrome output devices. The *connection* values in this equation should represent the achromatic channel of the profile connection space in the range of 0 to 1,0 where 0 represents black and 1,0 represents white. The PCS value is derived by multiplying the D50 white point by the normalized TRC value between 0 and 1,0. If the inverse of this is desired, then the following equation is used:

$$device = \text{grayTRC}^{-1}[connection] \tag{18}$$

AToB0Tag, AToB1Tag, AToB2Tag, BToA0Tag, BToA1Tag, BToA2Tag may be included in monochrome profiles. If these are present, their usage shall be as defined in Table 20.

NOTE The output values are the control values and not the "K" (black) values.

6.3.3.2 Color Output Profiles

Table 28 — Color output profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Device to PCS: 8-bit or 16-bit data: intent of 0
BToA0Tag	PCS to Device space: 8-bit or 16-bit data: intent of 0
gamutTag	Out of Gamut: 8-bit or 16-bit data
AToB1Tag	Device to PCS: 8-bit or 16-bit data: intent of 1
BToA1Tag	PCS to Device space: 8-bit or 16-bit data: intent of 1
AToB2Tag	Device to PCS: 8-bit or 16-bit data: intent of 2
BToA2Tag	PCS to Device space: 8-bit or 16-bit data: intent of 2
mediaWhitePointTag	Media XYZ white point
colorantTableTag	Colorants used in the profile, required if Color Space Signature is xCLR (e.g., 3CLR)
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The intent values used in Table 28 directly correlate to the rendering intent values defined in Table 18.

Each of the first three intents in Table 18 are associated with a specific tag. The fourth intent, ICC-absolute colorimetry, is obtained by using the media-relative colorimetric intent tag and the mediaWhitePointTag as described in Annex A.

It is permissible to reference the same tag data for all of these intents and to use the media-relative colorimetric intent tag when ICC-absolute colorimetry is specified.

The AToB0Tag, BToA0Tag, AToB1Tag, BToA1Tag, AToB2Tag, and BToA2Tag represent a device model that can use a multi-dimensional lookup table. The model is described in detail in the tag descriptions and the general lookup table tag element structures.

The colorantTableTag (Section 6.4.14) is a required tag to specify the names and XYZ or L*a*b* values of the colorants for the xCLR multi-dimensional color spaces, as these names are not otherwise implicit in the choice of the color space.

6.3.4 Additional Profile Formats

6.3.4.1 DeviceLink Profile

This profile represents a one-way link or connection between devices. It does not represent any device model nor can it be embedded into images.

Table 29 — DeviceLink profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Actual transformation parameter structure; 8-bit or 16-bit data
profileSequenceDescTag	An array of descriptions of the profile sequence
colorantTableTag	Colorants used in the profile, required if Color Space Signature is xCLR (e.g., 3CLR)
copyrightTag	Profile copyright information

The single AToB0Tag may contain data for any one of the four possible rendering intents. The rendering intent used is indicated in the header of the profile.

The AToB0Tag represents a device model that can use a multi-dimensional lookup table. The model is described in detail in the tag descriptions and the general lookup table tag element structures.

The DeviceLink profile is a pre-evaluated transform that cannot be undone.

The color space of data in the DeviceLink profile will be the same as the color space of the data of the first profile in the sequence. The profile connection space will be the same as the color space of the data of the last profile in the sequence.

The colorantTableTag (Section 6.4.14) is a required tag to specify the names and XYZ or L*a*b* values of the colorants for the xCLR multi-dimensional color spaces, as these names are not otherwise implicit in the choice of the color space.

6.3.4.2 ColorSpace Conversion Profile

This profile provides the relevant information to perform a color space transformation between the non-device color spaces and the PCS. It does not represent any device model. ColorSpace Conversion profiles may be embedded in images.

Table 30 — ColorSpace conversion profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
BToA0Tag	Inverse transformation parameter structure; 8-bit or 16-bit data
AToB0Tag	Actual transformation parameter structure; 8-bit or 16-bit data
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The AToB0Tag and BToA0Tag represent a model that can use a multi-dimensional lookup table. The model is described in detail in the tag descriptions and the general lookup table tag element structures.

For color transformation profiles, the device profile dependent fields are set to zero if irrelevant.

Additional multidimensional tags (AToB1Tag, AToB2Tag., BToA1Tag, BToA2Tag) may also be included. If these are present, their usage shall be as defined in Table 20.

In addition, a gamutTag may be included. The usage of this tag is identical as in output profiles (Section 6.3.3.2).

6.3.4.3 Abstract Profile

This profile represents abstract transforms and does not represent any device model. Color transformations using abstract profiles are performed from PCS to PCS. Abstract profiles cannot be embedded in images.

Table 31 — Abstract profile required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Actual transformation parameter structure; 8-bit or 16-bit data
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The AToB0Tag represents a PCS to PCS model that can use a multi-dimensional lookup table. The model is described in detail in the tag descriptions and the general lookup table tag element structures.

6.3.4.4 Named Color Profile

Named color profiles can be thought of as sibling profiles to device profiles. For a given device there would be one or more device profiles to handle process color conversions and one or more named color profiles to handle named colors. There might be multiple named color profiles to account for different consumables or multiple named color vendors. This profile provides a PCS and optional device representation for a list of named colors. Named color profiles are device-specific in that their data is shaped for a particular device.

Table 32 — Named color required tags

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
namedColor2Tag	PCS and optional device representation for named colors
mediaWhitePointTag	Media XYZ white point
copyrightTag	Profile copyright information
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.

The namedColor2Tag provides a PCS and optional device representation for each named color in a list of named colors. The PCS representation is provided to support general color management functionality. It is very useful for display and emulation of the named colors.

When using a named color profile with the device for which it is intended, the device representation of the color specifies the exact device coordinates for each named color. The PCS representation in conjunction with the device's output profile can provide an approximation of these exact coordinates. The exactness of this approximation is a function of the accuracy of the output profile and the color management system performing the transformations.

The combination of the PCS and device representations provides for flexibility with respect to accuracy and portability.

6.4 Tag descriptions

This clause specifies the individual tags used to create all possible portable profiles in the ICC Profile Format. The appropriate tag typing is indicated with each individual tag description. Note that the signature indicates only the type of data and does not imply anything about the use or purpose for which the data is intended.

Any of the tags in Table 33 can be used as optional tags if they are not used in the required set for a particular profile and are not specifically excluded in a profile definition.

The tag types are described in Section 6.5.

Table 33 — Tag list (Part 1 of 2)

Tag Name	General Description
AToB0Tag	Multidimensional transformation structure
AToB1Tag	Multidimensional transformation structure
AToB2Tag	Multidimensional transformation structure
blueMatrixColumnTag	Relative XYZ values of blue phosphor or colorant
blueTRCTag	The third column in the matrix used in TRC/matrix transforms (This column is combined with the linear blue channel during the matrix multiplication)
BToA0Tag	Multidimensional transformation structure
BToA1Tag	Multidimensional transformation structure
BToA2Tag	Multidimensional transformation structure
calibrationDateTimeTag	Profile calibration date and time
charTargetTag	Characterization target such as IT8/7.2
chromaticAdaptationTag	Converts XYZ color from the actual illumination source to PCS illuminant. Required only if the actual illumination source is not D50.
chromaticityTag	Set of phosphor/colorant chromaticity
colorantOrderTag	Identifies the laydown order of colorants
colorantTableTag	Identifies the colorants used in the profile
copyrightTag	Profile copyright information
deviceMfgDescTag	Displayable description of device manufacturer
deviceModelDescTag	Displayable description of device model
gamutTag	Out of gamut: 8-bit or 16-bit data

Table 33 — Tag list (Part 2 of 2)

grayTRCTag	Gray tone reproduction curve
greenMatrixColumnTag	The second column in the matrix used in TRC/matrix transforms (This column is combined with the linear green channel during the matrix multiplication)
greenTRCTag	Green channel tone reproduction curve
luminanceTag	Absolute luminance for emissive device
measurementTag	Alternative measurement specification information
mediaBlackPointTag	Media XYZ black point
mediaWhitePointTag	Media XYZ white point
namedColor2Tag	PCS and optional device representation for named colors
outputResponseTag	Description of the desired device response
preview0Tag	Preview transformation: 8-bit or 16-bit data
preview1Tag	Preview transformation: 8-bit or 16-bit data
preview2Tag	Preview transformation: 8-bit or 16-bit data
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for displays
profileSequenceDescTag	An array of descriptions of the profile sequence
redMatrixColumnTag	The first column in the matrix used in TRC/matrix transforms (This column is combined with the linear red channel during the matrix multiplication)
redTRCTag	Red channel tone reproduction curve
technologyTag	Device technology information such as LCD, CRT, Dye Sublimation, etc.
viewingCondDescTag	Viewing condition description
viewingConditionsTag	Viewing condition parameters

6.4.1 AToB0Tag

Tag Type: lut8Type or lut16Type or lutAtoBType
Tag Signature: ‘A2B0’ (41324230h)

This tag defines a color transform from Device to PCS using lookup table tag element structures. The processing mechanisms are described in lut8Type or lut16Type or lutAtoBType.

6.4.2 AToB1Tag

Tag Type: lut8Type or lut16Type or lutAtoBType
Tag Signature: ‘A2B1’ (41324231h)

This tag defines a color transform from Device to PCS using lookup table tag element structures. The processing mechanisms are described in lut8Type or lut16Type or lutAtoBType.

6.4.3 AToB2Tag

Tag Type: lut8Type or lut16Type or lutAtoBType
Tag Signature: 'A2B2' (41324232h)

This tag defines a color transform from Device to PCS using lookup table tag element structures. The processing mechanisms are described in lut8Type or lut16Type or lutAtoBType.

6.4.4 blueMatrixColumnTag

Tag Type: XYZType
Tag Signature: 'bXYZ' (6258595Ah)

The third column in the matrix used in TRC/matrix transforms.

6.4.5 blueTRCTag

Tag Type: curveType or parametricCurveType
Tag Signature: 'bTRC' (62545243h)

Blue channel tone reproduction curve. The first element represents no colorant (white) or phosphors (black) and the last element represents 100 percent colorant (blue) or 100 percent phosphor (blue).

6.4.6 BToA0Tag

Tag Type: lut8Type or lut16Type or lutBtoAType
Tag Signature: 'B2A0' (42324130h)

This tag defines a color transform from PCS to Device using the lookup table tag element structures. The processing mechanisms are described in lut8Type or lut16Type or lutBtoAType.

6.4.7 BToA1Tag

Tag Type: lut8Type or lut16Type or lutBtoAType
Tag Signature: 'B2A1' (42324131h)

This tag defines a color transform from PCS to Device using the lookup table tag element structures. The processing mechanisms are described in lut8Type or lut16Type or lutBtoAType.

6.4.8 BToA2Tag

Tag Type: lut8Type or lut16Type or lutBtoAType
Tag Signature: 'B2A2' (42324132h)

This tag defines a color transform from PCS to Device using the lookup table tag element structures. The processing mechanisms are described in lut8Type or lut16Type or lutBtoAType.

6.4.9 calibrationDateTimeTag

Tag Type: dateTimeType
Tag Signature: 'calt' (63616C74h)

Profile calibration date and time. Initially, this tag matches the contents of the profile header's creation date/time field. This allows applications and utilities to verify if this profile matches a vendor's profile and how recently calibration has been performed.

6.4.10 charTargetTag

Tag Type: textType
Tag Signature: 'targ' (74617267h)

This tag contains the name of the registered characterization data set, or it contains the measurement data for a characterization target. This tag is provided so that distributed utilities can identify the underlying characterization data, create transforms "on the fly" or check the current performance against the original device performance.

The first seven characters of the text shall identify the nature of the characterization data.

If the first seven characters are "ICCHDAT", then the remainder of the text shall be a single space followed by the Reference Name of a characterization data set in the ICC Characterization Data Registry and terminated with a NULL byte (00h). The Reference Name in the text must match exactly (including case) the Reference Name in the registry.

If the first seven characters match one of the identifiers defined in an ANSI or ISO standard, then the tag embeds the exact data file format defined in that standard. Each of these file formats contains an identifying character string as the first seven characters of the format, allowing an external parser to determine which data file format is being used. This provides the facilities to include a wide range of targets using a variety of measurement specifications in a standard manner.

NOTE: It is highly recommended that the profileDescriptionTag also include an identification of the characterization data that was used in the creation of the profile (e.g. "Based on CGATS TR 001").

6.4.11 chromaticAdaptationTag

Tag Type: s15Fixed16ArrayType
Tag Signature: 'chad' (63686164h)

This tag converts an XYZ color, measured at a device's specific illumination conditions, to an XYZ color in the PCS illumination conditions after complete adaptation.

The tag reflects a survey of the currently used methods of conversion, all of which can be formulated as a matrix transformation (see Annex E). Such a 3 by 3 chromatic adaptation matrix is organized as a 9-element array of signed 15.16 numbers (s15Fixed16ArrayType tag). Similarly as in the other occurrences of a 3 by 3 matrix in the ICC tags, the dimension corresponding to the matrix rows varies least rapidly while the one corresponding to the matrix columns varies most rapidly.

$$array = \begin{bmatrix} a0 & a1 & a2 & a3 & a4 & a5 & a6 & a7 & a8 \end{bmatrix} \tag{19}$$

$$\begin{bmatrix} Xpcs \\ Ypcs \\ Zpcs \end{bmatrix} = \begin{bmatrix} a0 & a1 & a2 \\ a3 & a4 & a5 \\ a6 & a7 & a8 \end{bmatrix} \begin{bmatrix} Xsrc \\ Ysrc \\ Zsrc \end{bmatrix} \tag{20}$$

Where *XYZsrc* represents the measured value in the actual device viewing condition and *XYZpcs* represents the chromatically adapted value in the PCS.

The chromatic adaptation matrix is a combination of three separate conversions:

1) Conversion of source CIE XYZ tristimulus values to cone response tristimulus values.

2) Adjustment of the cone response values for an observer's chromatic adaptation.

3) Conversion of the adjusted cone response tristimulus back to CIE XYZ values.

6.4.12 chromaticityTag

Tag Type: chromaticityType
Tag Signature: 'chrm' (6368726Dh)

The data and type of phosphor/colorant chromaticity set.

6.4.13 colorantOrderTag

Tag Type: colorantOrderType
Tag Signature: 'clro' (636C726Fh)

This tag specifies the laydown order of colorants.

6.4.14 colorantTableTag

Tag Type: colorantTableType
Tag Signature: 'clrt' (636C7274h)

This tag identifies the colorants used in the profile by a unique name and an XYZ or L*a*b* value.

This is a required tag for profiles where the color space defined in the header is xCLR, where x is one of the allowed numbers from 2 through Fh, per Table 13. See Section 6.3.3.2, Section 6.3.4.1.

6.4.15 copyrightTag

Tag Type: multiLocalizedUnicodeType
Tag Signature: 'cprt' (63707274h)

This tag contains the text copyright information for the profile.

6.4.16 deviceMfgDescTag

Tag Type: multiLocalizedUnicodeType
Tag Signature: 'dmnd' (646D6E64h)

Structure containing invariant and localizable versions of the device manufacturer for display. The content of this structure is described in 6.5.12.

6.4.17 deviceModelDescTag

Tag Type: multiLocalizedUnicodeType
Tag Signature: 'dmdd' (646D6464h)

Structure containing invariant and localizable versions of the device model for display. The content of this structure is described in 6.5.12.

6.4.18 gamutTag

Tag Type: lut8Type or lut16Type or lutBtoAType
Tag Signature: 'gamt' (67616D74h)

Out of gamut tag. The processing mechanisms are described in lut8Type or lut16Type or lutBtoAType.

This tag takes PCS values as its input and produces a single channel of output. If the output value is 0, the PCS color is in-gamut. If the output is non-zero, the PCS color is out-of-gamut, with the output value "n+1" being at least as far out of gamut as the output value "n".

6.4.19 grayTRCTag

Tag Type: curveType or parametricCurveType
Tag Signature: 'kTRC' (6B545243h)

Gray tone reproduction curve. The tone reproduction curve provides the necessary information to convert between a single device channel and the CIEXYZ encoding of the profile connection space. The first element represents black and the last element represents white.

6.4.20 greenMatrixColumnTag

Tag Type: XYZType
Tag Signature: 'gXYZ' (6758595Ah)

The second column in the matrix used in TRC/matrix transforms.

6.4.21 greenTRCTag

Tag Type: curveType or parametricCurveType
Tag Signature: 'gTRC' (67545243h)

Green channel tone reproduction curve. The first element represents no colorant (white) or phosphors (black) and the last element represents 100 percent colorant (green) or 100 percent phosphor (green).

6.4.22 luminanceTag

Tag Type: XYZType
Tag Signature: 'lumi' (6C756D69h)

Absolute luminance of emissive devices in candelas per square meter as described by the Y channel. The X and Z channels are ignored in all cases.

6.4.23 measurementTag

Tag Type: measurementType
Tag Signature: 'meas' (6D656173h)

Alternative measurement specification such as a D65 illuminant instead of the default D50.

6.4.24 mediaBlackPointTag

Tag Type: XYZType
Tag Signature: 'bkpt' (626B7074h)

This tag specifies the media black point and contains the CIE 1931 XYZ colorimetry of the black point of the actual medium.

NOTE Previous revisions of this specification contained an error indicating that this tag is used to calculate ICC-absolute colorimetry. This is not the case.

6.4.25 mediaWhitePointTag

Tag Type: XYZType
Tag Signature: 'wtpt' (77747074h)

This tag, which is used for generating ICC-absolute colorimetric intent, specifies the XYZ tristimulus values of the media white point. If the media is measured under an illumination source which has a chromaticity other than D50, the measured values must be adjusted to D50 using the chromaticAdaptationTag matrix before recording in the tag. For reflecting and transmitting media, the tag values are specified relative to the perfect diffuser (which is normalized to a Y value of 1,0) for illuminant D50. For displays, the values specified must be those of D50 (i.e. 0,9642, 1,0 0,8249) normalized such that Y = 1,0.

See Annex A for a more complete description of the use of the media white point.

6.4.26 namedColor2Tag

Tag Type: namedColor2Type
Tag Signature: 'ncl2' (6E636C32h)

Named color information providing a PCS and optional device representation for a list of named colors.

6.4.27 outputResponseTag

Tag Type: responseCurveSet16Type
Tag Signature: 'resp' (72657370h)

Structure containing a description of the device response for which the profile is intended. The content of this structure is described in 6.5.16.

NOTE The user's attention is called to the possibility that the use of this tag for device calibration may require use of an invention covered by patent rights. By publication of this specification, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. The patent holder has, however, filed a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. Details may be obtained from the publisher.

6.4.28 preview0Tag

Tag Type: lut8Type or lut16Type or lutBtoAType
Tag Signature: 'pre0' (70726530h)

Preview transformation from PCS to device space and back to the PCS. The processing mechanisms are described in lut8Type or lut16Type or lutBtoAType.

This tag contains the combination of tag B2A0 and tag A2B1.

6.4.29 preview1Tag

Tag Type: lut8Type or lut16Type or lutBtoAType
Tag Signature: 'pre1' (70726531h)

Preview transformation from the PCS to device space and back to the PCS. The processing mechanisms are described in lut8Type or lut16Type or lutBtoAType.

This tag contains the combination of tag B2A1 and tag A2B1.

6.4.30 preview2Tag

Tag Type: lut8Type or lut16Type or lutBtoAType
Tag Signature: 'pre2' (70726532h)

Preview transformation from PCS to device space and back to the PCS. The processing mechanisms are described in lut8Type or lut16Type or lutBtoAType.

This tag contains the combination of tag B2A2 and tag A2B1.

6.4.31 profileDescriptionTag

Tag Type: multiLocalizedUnicodeType
Tag Signature: 'desc' (64657363h)

Structure containing invariant and localizable versions of the profile description for display. The content of this structure is described in 6.5.12. This invariant description has no fixed relationship to the actual profile disk file name.

6.4.32 profileSequenceDescTag

Tag Type: profileSequenceDescType
Tag Signature: 'pseq' (70736571h)

Structure containing a description of the profile sequence from source to destination, typically used with the DeviceLink profile. The content of this structure is described in 6.5.15.

6.4.33 redMatrixColumnTag

Tag Type: XYZType
Tag Signature: 'rXYZ' (7258595Ah)

The first column in the matrix used in TRC/matrix transforms.

6.4.34 redTRCTag

Tag Type: curveType or parametricCurveType
Tag Signature: 'rTRC' (72545243h)

Red channel tone reproduction curve. The first element represents no colorant (white) or phosphors (black) and the last element represents 100 percent colorant (red) or 100 percent phosphor (red).

6.4.35 technologyTag

Tag Type: signatureType
Tag Signature: 'tech' (74656368h)

Device technology information such as CRT, Dye Sublimation, etc. The encoding is such that:

Table 34 — Technology signatures

Technology	Signature	Hex Encoding
Film Scanner	'fscn'	6673636Eh
Digital Camera	'dcam'	6463616Dh
Reflective Scanner	'rscn'	7273636Eh
Ink Jet Printer	'ijet'	696A6574h
Thermal Wax Printer	'twax'	74776178h
Electrophotographic Printer	'epho'	6570686Fh
Electrostatic Printer	'esta'	65737461h
Dye Sublimation Printer	'dsub'	64737562h
Photographic Paper Printer	'rpho'	7270686Fh
Film Writer	'fprn'	6670726Eh
Video Monitor	'vidm'	7669646Dh
Video Camera	'vidc'	76696463h
Projection Television	'pjtv'	706A7476h
Cathode Ray Tube Display	'CRT '	43525420h
Passive Matrix Display	'PMD '	504D4420h
Active Matrix Display	'AMD '	414D4420h
Photo CD	'KPCD'	4B504344h
PhotoImageSetter	'imgs'	696D6773h
Gravure	'grav'	67726176h
Offset Lithography	'offs'	6F666673h
Silkscreen	'silk'	73696C6Bh
Flexography	'flex'	666C6578h

6.4.36 viewingCondDescTag

Tag Type: multiLocalizedUnicodeType
Tag Signature: 'vued' (76756564h)

Structure containing invariant and localizable versions of the viewing conditions. The content of this structure is described in 6.5.12.

6.4.37 viewingConditionsTag

Tag Type: viewingConditionsType
Tag Signature: ‘view’ (76696577h)

Viewing conditions parameters. The content of this structure is described in 6.5.25.

6.5 Tag type definitions

This clause specifies the type and structure definitions used to create all of the individual tagged elements in the ICC Profile Format. The data type description identifiers are indicated at the right margin of each data or structure definition. An effort was made to make sure one-byte, two-byte and four-byte data lies on one-byte, two-byte and four-byte boundaries respectively. This required occasionally including extra spaces indicated with “reserved for padding” in some tag type definitions. Value 0 is defined to be of “unknown value” for all enumerated data structures.

All tags, including private tags, have as their first four bytes (0..3) a tag signature (a 4-byte sequence) to identify to profile readers what kind of data is contained within a tag. This encourages tag type reuse and allows profile parsers to reuse code when tags use common tag types. The second four bytes (4..7) are reserved for future expansion and must be set to 0 in this version of the specification. Each new tag signature and tag type signature must be registered with the International Color Consortium (see 0.7) in order to prevent signature collisions.

Where not specified otherwise, the least-significant 16 bits of all 32-bit flags in the type descriptions below are reserved for use by the International Color Consortium.

When 7-bit ASCII text representation is specified in types below, each individual character is encoded in 8 bits with the most-significant bit set to zero. The details are presented in 5.3.11.

6.5.1 chromaticityType

The chromaticityType information provides basic chromaticity data and type of phosphors or colorants of a monitor to applications and utilities. The byte stream is given below..

Table 35 — chromaticityType encoding

Byte Offset	Content	Encoded as...
0..3	’chrm’ (6368726Dh) type signature	
4..7	reserved, must be set to 0	
8..9	Number of Device Channels	uInt16Number
10..11	encoded value of phosphor or colorant type	see below
12..19	CIE xy coordinate values of channel 1	u16Fixed16Number[2]
20..27	CIE xy coordinate values of channel 2	u16Fixed16Number[2]
28..35	CIE xy coordinate values of channel 3	u16Fixed16Number[2]
36..end	CIE xy coordinate values of other channels (if needed)	u16Fixed16Number[...]

When using this type, it is necessary to assign each color space component to device channel. Table 43 in 6.5.7 shows these assignments.

The encoding for the phosphor or colorant type field is such that:.

Table 36 — Phosphor or colorant encoding

Phosphor or Colorant Type	Encoded Value	Channel 1 x,y	Channel 2 x,y	Channel 3 x,y
unknown	0000h	any	any	any
ITU-R BT.709	0001h	(0,640, 0,330)	(0,300, 0,600)	(0,150, 0,060)
SMPTE RP145-1994	0002h	(0,630, 0,340)	(0,310, 0,595)	(0,155, 0,070)
EBU Tech.3213-E	0003h	(0,64 0,33)	(0,29, 0,60)	(0,15, 0,06)
P22	0004h	(0,625, 0,340)	(0,280, 0,605)	(0,155, 0,070)

When the encoded value is 0000h, the actual set of chromaticity values shall be described. Otherwise, the chromaticity values shall match the table values for the given phosphor type.

6.5.2 colorantOrderType

Table 37 — colorantOrderType encoding

Byte Offset	Content	Encoded as...
0..3	‘clro’ (636c726fh) type signature	
4..7	reserved, must be set to 0	
8..11	Count of colorants	uInt32Number
12	One-based number of the colorant to be printed first.	
13..n	The remaining count-1 colorants are described in a manner consistent with the first colorant	

This is an optional tag which specifies the laydown order in which colorants will be printed on an n-colorant device. The laydown order may be the same as the channel generation order listed in the colorantTableTag or the channel order of a color space such as CMYK, in which case this tag is not needed.

When this is not the case (for example, ink-towers sometimes use the order KCMY), this tag may be used to specify the laydown order of the colorants.

The size of the array is the same as the number of colorants. The first position in the array contains the number of the first colorant to be laid down, the second position contains the number of the second colorant to be laid down, and so on, until all colorants are listed. The colorant numbers are one-based.

When this tag is used, the "count of colorants" must be in agreement with the color space signature of Section 6.1.5.

6.5.3 colorantTableType

Table 38 — colorantTableType encoding

Byte Offset	Content	Encoded as...
0..3	'clrt' (636c7274h) type signature	
4..7	reserved, must be set to 0	
8..11	Count of colorants	uInt32Number
12..43	First colorant name (32 byte field, null terminated, unused bytes must be set to zero)	7-bit ASCII
44..49	First colorant coordinates in the PCS colorspace of the profile as described in Section 6.1.6 (the Profile Connection Space Signature in the header). Only 16-bit PCS coordinates are allowed.	uInt16Number
50..n	The remaining colorants, if count > 1, described using the format of bytes 12-49 of the first colorant	

The purpose of this tag is to identify the colorants used in the profile by a unique name and an XYZ or L*a*b* value to give the colorant an unambiguous value. The first colorant listed is the colorant of the first device channel of a lut tag. The second colorant listed is the colorant of the second device channel of a lut tag, and so on.

The XYZ or L*a*b* value may also be used to derive the visual density of the colorant, which trapping algorithms may then use to determine overlay values. The visual density is calculated using the formula:

$$\text{Visual Density} = -\log_{10}(Y) \quad \text{if the PCS is in XYZ} \tag{21}$$

If the PCS is in L*a*b*, convert L* to Y using:

$$Y = L^* / 903{,}3 \quad \text{if } L^* < 8 \tag{22}$$

$$Y = ((L^* + 16) / 116)^3 \quad \text{if } L^* >= 8 \tag{23}$$

The colorant coordinates are provided only for convenience and, for many profile classes, should be populated by processing the individual colorants through the AToB1Tag of the profile if this tag exists, otherwise the user must supply the coordinates if this tag is to be used. An individual colorant has the maximum value in the channel corresponding to that colorant and the minimum value in all other channels.

For example, using a 3CLR 8-bit profile, the individual colorant for the first channel would be (255, 0, 0). Processing this color through the AToB1Tag would produce the first colorant coordinates listed in bytes 44-49.

This tag is required for output profiles and named color profiles where the device colorspace (Section 6.1.5, Color Space Signature) is one of the xCLR color spaces. When this tag is used, the "count of colorants" must be in agreement with the color space signature of Section 6.1.5.

6.5.4 curveType

The curveType contains a 4-byte count value and a one-dimensional table of 2-byte values. The byte stream is given below.

Table 39 — curveType encoding

Byte Offset	Content	Encoded as...
0..3	'curv' (63757276h) type signature	
4..7	reserved, must be set to 0	
8..11	count value specifying number of entries that follow	uInt32Number
12..end	actual curve values starting with the zeroth entry and ending with the entry *count*-1.	uInt16Number[...]

The count value specifies the number of entries in the curve table except as follows:

when *count* is 0, then a linear response (slope equal to 1,0) is assumed,

when *count* is 1, then the data entry is interpreted as a simple gamma value encoded as a u8Fixed8Number. Gamma is interpreted canonically and not as an inverse.

when *count* is 2, then the data entries shall be set so that the correct results are obtained when linear interpolation is used to generate intermediate values.

6.5.5 dataType

The dataType is a simple data structure that contains either 7-bit ASCII or binary data, i.e. textType data or transparent 8-bit bytes. The length of the string is obtained by subtracting 12 from the element size portion of the tag itself. If this type is used for ASCII data, it must be terminated with a 00h byte.

Table 40 — dataType encoding

Byte Offset	Content
0..3	'data' (64617461h) type signature
4..7	reserved, must be set to 0
8..11	data flag, 00000000h represents ASCII data, 00000001h represents binary data, other values are reserved for future use
12..end	a string of (element size - 12) ASCII characters or (element size - 12) bytes

6.5.6 dateTimeType

This dateTimeType is a 12-byte value representation of the time and date. The actual values are encoded as a dateTimeNumber described in 5.3.1.

Table 41 — dateTimeType encoding

Byte Offset	Content	Encoded as...
0..3	'dtim' (6474696Dh) type signature	
4..7	reserved, must be set to 0	
8..19	date and time	dateTimeNumber

6.5.7 lut16Type

This structure converts an input color into an output color using tables with 16-bit precision. This type contains four processing elements: a 3 by 3 matrix (only used when the input color space is XYZ), a set of one dimensional input lookup tables, a multidimensional lookup table, and a set of one dimensional output tables. Data is processed using these elements via the following sequence:

(matrix) ⇒ (1d input tables) ⇒ (multidimensional lookup table) ⇒ (1d output tables).

Table 42 — lut16Type encoding

Byte Offset	Content	Encoded as...
0..3	'mft2' (6D667432h) [multi-function table with 2-byte precision] type signature	
4..7	reserved, must be set to 0	
8	Number of Input Channels	uInt8Number
9	Number of Output Channels	uInt8Number
10	Number of CLUT grid points (identical for each side)	uInt8Number
11	Reserved for padding (required to be 00h)	
12..15	Encoded e00 parameter	s15Fixed16Number
16..19	Encoded e01 parameter	s15Fixed16Number
20..23	Encoded e02 parameter	s15Fixed16Number
24..27	Encoded e10 parameter	s15Fixed16Number
28..31	Encoded e11 parameter	s15Fixed16Number
32..35	Encoded e12 parameter	s15Fixed16Number
36..39	Encoded e20 parameter	s15Fixed16Number
40..43	Encoded e21 parameter	s15Fixed16Number
44..47	Encoded e22 parameter	s15Fixed16Number
48..49	Number of input table entries	uInt16Number
50..51	Number of output table entries	uInt16Number
52..n	Input tables	uInt16Number[...]
n+1..m	CLUT values	uInt16Number[...]
m+1..o	Output tables	uInt16Number[...]

The input, output, and grid tables contained in a lut16Type each embodies a one- or multi-dimensional function which maps an input value in the "domain" of the function to an output value in the "range" of the function.

The domain of each of these tables is defined to consist of all real numbers between 0,0 and 65535,0, inclusive. The first entry is located at 0,0, the last entry at 65535,0, and intermediate entries are uniformly spaced using an increment of 65535,0/(M-1). For the input and output tables, M is the number of entries in the table. For the CLUT, M is the number of grid points along each dimension. Note that since the increment of 65535,0/(M-1) is not necessarily an integer, the domain is specified to be over the real numbers rather than restricting it to the integers only.

The range of a function used to generate the contents of a table is likewise defined to be all real numbers between 0,0 and 65535,0, inclusive. Because the contents of a table are encoded using 16 bits of precision, it is necessary to round each real value to the nearest 16-bit integer.

This means that both the domain and range of the functions represented by the elements of the lut16Type as a whole are all real numbers between 0,0 and 65535,0, inclusive. In many situations it is necessary to convert between these 16-bit values and some other bit precision.

See Annex A for additional guidance on this topic.

The matrix is organized as a 3 by 3 array. The dimension corresponding to the matrix rows varies least rapidly and the dimension corresponding to the matrix columns varies most rapidly and is shown in matrix form below.

$$\begin{bmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{bmatrix}$$

When using the matrix of an output profile, and the input data is XYZ, we have

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (24)$$

Each input X, Y or Z is an unsigned 1.15 number and each matrix entry is a signed 15.16 number. Therefore, each multiplication in the matrix multiply is 1.15 * s15.16 = s16.31 and the final sum is also s16.31 (48 bits). From this sum we take bits 31..16 as the unsigned integer result for X', Y', or Z'. These are then used as the inputs to the input tables of the multidimensional LUT. This normalization is used since the number of fractional bits in the input data must be maintained by the matrix operation.

The matrix is mandated to be an identity matrix unless the input is in the XYZ color space.

The input tables are arrays of 16-bit unsigned values. Each input table consists of a minimum of two and a maximum of 4096 two-byte integers. Each input table entry is appropriately normalized to the range 0-65535. The inputTable is of size (InputChannels * inputTableEntries * 2) bytes. When stored in this tag, the one-dimensional lookup tables are assumed to be packed one after another in the order described below.

The CLUT is organized as an n-dimensional array with a given number of grid points in each dimension, where n is the number of input channels (input tables) in the transform. The dimension corresponding to the first input channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value contains m two-byte integers, where m is the number of output channels. The first sequential two-byte integer of the entry contains the function value for the first output function, the second sequential two-byte integer of the entry contains the function value for the second output function, and so on until all the output functions have been supplied. Each two-byte integer in the CLUT is appropriately normalized to the range of 0 - 65535. The equation for computing the byte size of the CLUT is:

$$CLUTSize = (GridPoints^{InputChannels} * OutputChannels * 2) \text{ bytes} \quad (25)$$

The output tables are arrays of 16-bit unsigned values. Each output table consists of a minimum of two and a maximum of 4096 two-byte integers. Each output table entry is appropriately normalized to the range 0 - 65535. The outputTable is of size (OutputChannels * outputTableEntries * 2) bytes. When stored in this tag, the one-dimensional lookup tables are assumed to be packed one after another in the order described in the following paragraph.

If the number of data points in a one-dimensional table, or in a particular dimension of the CLUT, is two, the data for those points shall be set so that the correct results are obtained when linear interpolation is used to generate intermediate values.

When using this type, it is necessary to assign each color space component to an input and output channel. The following table shows these assignments. The channels are numbered according to the order in which their table occurs. Note that additional color spaces can be added simply by defining the signature, channel assignments, and creating the tables.

Table 43 — lut16Type channel encodings

Color Space	Channel 1	Channel 2	Channel 3	Channel 4
‘XYZ ’	X	Y	Z	
‘Lab ’	L	a	b	
‘Luv ’	L	u	v	
‘YCbr’	Y	Cb	Cr	
‘Yxy ’	Y	x	y	
‘RGB ’	R	G	B	
‘GRAY’	K			
‘HSV ’	H	S	V	
‘HLS ’	H	L	S	
‘CMYK’	C	M	Y	K
‘CMY ’	C	M	Y	
‘2CLR’	Ch. 1	Ch. 2		
‘3CLR’	Ch. 1	Ch. 2	Ch. 3	
‘4CLR’	Ch. 1	Ch. 2	Ch. 3	Ch. 4

The color space used on the PCS side of a lut16Type tag (which may be either the input or output space, or both in the case of an abstract profile) is identified by the Profile Connection Space field in the profile header (see 6.1.6). This field does not distinguish between 8-bit and 16-bit PCS encodings. For the lut16Type tag, the 'Lab' signature is defined to specify a legacy 16-bit CIELAB encoding and the 'XYZ ' signature is defined to specify the 16-bit XYZ encoding. Note that this definition only applies to the encoding used at the Profile Connection Space side of the tag. It does NOT apply when these signatures are used on the "Color Space of Data" field in the profile header (see 6.1.5), except in the case of an abstract profile.

For color values that are in the Lab color space on the PCS side of the tag, this tag uses a legacy 16-bit Lab encoding, not the 16-bit CIELAB PCS encoding that is defined in Annex A.2. This encoding is retained for backwards compatibility with profile version 2.

To convert color values from this tag's legacy 16-bit Lab encoding to the 16-bit CIELAB PCS encoding defined in Annex A.2, multiply all values with 65535/65280 (that is, FFFFh/FF00h). Any color values that are in the value range of legacy 16-bit PCS Lab, but not in the 16-bit CIELAB PCS encoding defined in Annex A.2, shall be clipped on a per-component basis when transforming from legacy 16-bit PCS Lab to the 16-bit CIELAB PCS encoding defined in Annex A.2. To convert color values from the 16-bit CIELAB PCS encoding defined in Annex A.2 to this tag's legacy 16-bit Lab encoding, divide all values with 65535/65280.

The L* values have a different encoding than the a* and b* values. The L* encoding is:

Table 44 — L* encoding

Value (L*)	16 bit
0	0000h
100,0	FF00h
100 + (25500/65280)	FFFFh

Although the 16-bit encoding can represent values slightly greater than 100,0, these are not valid PCS L* values and they should not be used.

The a* and b* encoding is:

Table 45 — a* or b* encoding

Value (a* or b*)	16-bit
-128,0	0000h
0	8000h
127,0	FF00h
127 + (255/256)	FFFFh

Note that the 16-bit encoding can represent values slightly greater than 127,0. Since a* and b* have no defined limits, these are valid PCS values.

6.5.8 lut8Type

This structure converts an input color into an output color using tables of 8-bit precision. This type contains four processing elements: a 3 by 3 matrix (only used when the input color space is XYZ), a set of one dimensional input lookup tables, a multidimensional lookup table, and a set of one dimensional output tables. Data is processed using these elements via the following sequence:

(matrix) ⇒ (1d input tables) ⇒ (multidimensional lookup table) ⇒ (1d output tables).

Table 46 — lut8Type encoding (Part 1 of 2)

Byte Offset	Content	Encoded as...
0..3	‘mft1’ (6D667431h) [multi-function table with 1-byte precision] type signature	
4..7	reserved, must be set to 0	
8	Number of Input Channels	uInt8Number
9	Number of Output Channels	uInt8Number
10	Number of CLUT grid points (identical for each side)	uInt8Number
11	Reserved for padding (fill with 00h)	
12..15	Encoded e00 parameter	s15Fixed16Number
16..19	Encoded e01 parameter	s15Fixed16Number

Table 46 — lut8Type encoding (Part 2 of 2)

20..23	Encoded e02 parameter	s15Fixed16Number
24..27	Encoded e10 parameter	s15Fixed16Number
28..31	Encoded e11 parameter	s15Fixed16Number
32..35	Encoded e12 parameter	s15Fixed16Number
36..39	Encoded e20 parameter	s15Fixed16Number
40..43	Encoded e21 parameter	s15Fixed16Number
44..47	Encoded e22 parameter	s15Fixed16Number
48..m	Input tables	uInt8Number[...]
m+1..n	CLUT values	uInt8Number[...]
n+1..o	Output tables	uInt8Number[...]

The input, output, and grid tables contained in a lut8Type each embodies a one- or multi-dimensional function which maps an input value in the "domain" of the function to an output value in the "range" of the function.

The domain of each of these tables is defined to consist of all real numbers between 0,0 and 255,0, inclusive. The first entry is located at 0,0, the last entry at 255,0, and intermediate entries are uniformly spaced using an increment of 255,0/(M-1). For the input and output tables, M is 256. For the CLUT, M is the number of grid points along each dimension. Note that since the increment of 255,0/(M-1) is not necessarily an integer, the domain is specified to be over the real numbers rather than restricting it to the integers only. The range of a function used to generate the contents of a table is likewise defined to be all real numbers between 0,0 and 255,0, inclusive.

Because the contents of a table are encoded using 8 bits of precision, it is necessary to round each real value to the nearest 8-bit integer. This means that both the domain and range of the functions represented by the elements of the lut8Type as a whole are all real numbers between 0,0 and 255,0, inclusive. In many situations it is necessary to convert between these 8-bit values and some other bit precision.

See Annex A for additional guidance on this topic.

The color space used on the PCS side of a lut8Type tag (which may be either the input or output space, or both in the case of an abstract profile) is identified by the Profile Connection Space field in the profile header (see 6.1.6). This field does not distinguish between 8-bit and 16-bit PCS encodings. For the lut8Type tag, the 'Lab' signature is defined to specify the 8-bit CIELAB encoding. Note that this definition only applies to the encoding used as the Profile Connection Space side of the tag. It does NOT apply when these signatures are used on the "Color Space of Data" field in the profile header (see 6.1.5), except in the case of an abstract profile.

An 8-bit XYZ PCS has not been defined, so the interpretation of a lut8Type in a profile that uses the XYZ PCS is implementation specific. Because of the resulting ambiguity and because an 8-bit linear quantization of XYZ results in poor quality, it is recommended that the lut8Type tag not be used in profiles that employ the XYZ PCS.

The matrix is organized as a 3 by 3 array. The dimension corresponding to the matrix rows varies least rapidly and the dimension corresponding to the matrix columns varies most rapidly and is shown in matrix form below.

$$\begin{bmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{bmatrix}$$

When using the matrix of an output profile, and the input data is XYZ, we have

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (26)$$

Each input X, Y or Z is an unsigned 1.15 number and each matrix entry is a signed 15.16 number. Therefore, each multiplication in the matrix multiply is 1.15 * s15.16 = s16.31 and the final sum is also s16.31 (48 bits). From this sum we take bits 31..16 as the unsigned integer result for X', Y', or Z'. These are then scaled to the range 0-255 and used as the inputs to the input tables of the multidimensional LUT. This normalization is used since the number of fractional bits in the input data must be maintained by the matrix operation.

The matrix is mandated to be an identity matrix unless the input is in the XYZ color space.

The input tables are arrays of 8-bit unsigned values. Each input table consists of 256 one-byte integers. Each input table entry is appropriately normalized to the range 0-255. The inputTable is of size (InputChannels * 256) bytes. When stored in this tag, the one-dimensional lookup tables are assumed to be packed one after another in the order described below.

The CLUT is organized as an n-dimensional array with a given number of grid points in each dimension, where n is the number of input channels (input tables) in the transform. The dimension corresponding to the first input channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is an m-byte array, where m is the number of output channels. The first sequential byte of the entry contains the function value for the first output function, the second sequential byte of the entry contains the function value for the second output function, and so on until all the output functions have been supplied. Each byte in the CLUT is appropriately normalized to the range 0 - 255. The equation for computing the byte size of the CLUT is:

$$CLUTSize = (GridPoints^{InputChannels} * OutputChannels) \text{ bytes} \quad (27)$$

The output tables are arrays of 8-bit unsigned values. Each output table consists of 256 one-byte integers. Each output table entry is appropriately normalized to the range 0 - 255. The outputTable is of size (OutputChannels * 256) bytes. When stored in this tag, the one-dimensional lookup tables are assumed to be packed one after another in the order described in the following paragraph.

If the number of data points in a one-dimensional table, or in a particular dimension of the CLUT, is two, the data for those points shall be set so that the correct results are obtained when linear interpolation is used to generate intermediate values.

When using this type, it is necessary to assign each color space component to an input and output channel. The following table shows these assignments. The channels are numbered according to the order in

which their table occurs. Note that additional color spaces can be added simply by defining the signature, channel assignments, and creating the tables.

Table 47 — lut8Type channel encodings

Color Space	Channel 1	Channel 2	Channel 3	Channel 4
‘XYZ ’	X	Y	Z	
‘Lab ’	L	a	b	
‘Luv ’	L	u	v	
‘Yxy ’	Y	x	y	
‘YCbr’	Y	Cb	Cr	
‘RGB ’	R	G	B	
‘GRAY’	K			
‘HSV ’	H	S	V	
‘HLS ’	H	L	S	
‘CMYK’	C	M	Y	K
‘CMY ’	C	M	Y	
‘2CLR’	Ch. 1	Ch. 2		
‘3CLR’	Ch. 1	Ch. 2	Ch. 3	
‘4CLR’	Ch. 1	Ch. 2	Ch. 3	Ch. 4

6.5.9 lutAtoBType

This structure converts an input color value to an output color value. The type contains up to five processing elements: a set of one dimensional curves, a 3 by 3 matrix with offset terms, a set of one dimensional curves, a multidimensional lookup table, and a set of one dimensional output curves. Data are processed using these elements via the following sequence:

("A" curves)-> (multidimensional lookup table)->("M" curves)-> (matrix) ->("B" curves).

NOTE The processing elements are not in this order in the tag to allow for simplified reading and writing of profiles.

It is possible to use any or all of these processing elements. At least one processing element must be included. Some processing elements are not allowed in certain circumstances.

The following combinations are allowed for the lutAtoBType:

B
M - Matrix - B
A - CLUT - B
A - CLUT - M - Matrix - B

Other combinations may be achieved by setting processing element values to identity transforms.

When using this type, it is necessary to assign each color space component to an input and output channel. This assignment is specified in Table 43.

Table 48 — lutAtoBType encoding

Byte Offset	Content	Encoded as...
0..3	‘mAB ’ (6D414220h) [multi-function A-to-B table] type signature	
4..7	reserved, must be set to 0	
8	Number of Input Channels	uInt8Number
9	Number of Output Channels	uInt8Number
10..11	Reserved for padding, must be set to 0	
12..15	Offset to first "B" curve*	uInt32Number
16..19	Offset to matrix	uInt32Number
20..23	Offset to first "M" curve*	uInt32Number
24..27	Offset to CLUT	uInt32Number
28..31	Offset to first "A" curve*	uInt32Number
32..n	First data entry	

Each curve and processing element must start on a 4-byte boundary. To achieve this, each item may be followed by up to three 00h pad bytes as needed.

The offset entries (bytes 12-31) point to the various processing elements found in the tag. The offsets indicate the number of bytes from the beginning of the tag to the desired data. If any of the offsets are zero, that is an indication that processing element is not present and the operation is not performed.

Either the XYZ or Lab PCS may be used with this tag type.

6.5.9.1 "A" Curves

There are the same number of "A" curves as there are input channels. The "A" curves may only be used when the CLUT is used. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "A" curve is stored as an embedded curveType or a parametricCurveType. The length is as indicated by the convention of the respective curve type. Note that the entire tag type, including the tag type signature and reserved bytes, is included for each curve.

6.5.9.2 CLUT

The CLUT appears as an n-dimensional array, with each dimension having a number of entries corresponding to the number of grid points.

The CLUT values are arrays of 8 bit or 16 unsigned values, normalized to the range of 0-255 or 65535.

The CLUT is organized as an n-dimensional array with a variable number of grid points in each dimension, where n is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is an m-integer array, where m is the number of output channels. The first sequential

integer of the entry contains the function value for the first output function, the second sequential integer of the entry contains the function value for the second output function and so on until all of the output functions have been supplied. The equation for computing the byte size of the CLUT is defined below.

nGrid1 * nGrid2 *... * nGridN * number of output channels * sizeof (channel component) (28)

Table 49 — lutAtoBType CLUT encoding

Byte Offset	Content	Encoded as...
0..15	Number of grid points in each dimension. Only the first n entries are used, where n is the number of input channels. Unused entries shall be set to 00h.	uInt8Number[16]
16	Precision of data elements in bytes. Must be either 01h or 02h.	uInt8Number
17..19	Reserved for padding, must be set to 0	
20..n	CLUT data points (arranged as described in the text).	uInt8Number[...] or uInt16Number[...]

If the number of input channels does not equal the number of output channels, the CLUT must be present.

If the number of grid points in a one-dimensional table, or in a particular dimension of the CLUT, is two, the data for those points shall be set so that the correct results are obtained when linear interpolation is used to generate intermediate values.

6.5.9.3 "M" Curves

There are the same number of "M" curves as there are output channels. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "M' curve is stored as an embedded curveType or a parametricCurveType. The length is as indicated by the convention of the respective curve type. Note that the entire tag type, including the tag type signature and reserved bytes, is included for each curve. The "M" curves may only be used when the matrix is used.

6.5.9.4 Matrix

The matrix is organized as a 3x4 array. The elements appear in order from e1-e12. The matrix elements are each s15Fixed16Numbers.

array = $[e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}]$

The matrix is used to convert data to a different color space, according to the following equation:

$$\begin{bmatrix} Y1 \\ Y2 \\ Y3 \end{bmatrix} = \begin{bmatrix} e1 & e2 & e3 \\ e4 & e5 & e6 \\ e7 & e8 & e9 \end{bmatrix} \bullet \begin{bmatrix} X1 \\ X2 \\ X3 \end{bmatrix} + \begin{bmatrix} e10 \\ e11 \\ e12 \end{bmatrix} \quad (29)$$

Each input X1, X2, or X3 is an u16Fixed16Number and each matrix entry is a s15Fixed16Number. Therefore, each multiplication in the matrix multiply is 1.15*s15.16=s16.31 and the final sum is also s16.31(48

bits). From this sum we take bits 31-16 as the unsigned integer result for Yl, Y2, or Y3. These are used as the inputs to the "B" curves.

The matrix is allowed only if the number of output channels, or "M" curves, is 3.

6.5.9.5 "B" Curves

There are the same number of "B" curves as there are output channels. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "B" curve is stored as an embedded curveType or a parametricCurveType. The length is as indicated by the convention of the respective curve type. Note that the entire tag type, including the tag type signature and reserved bytes, are included for each curve.

6.5.10 lutBtoAType

This structure converts an input color value to an output color value. The type contains up to five processing elements: a set of one dimensional curves, a 3 by 3 matrix with offset terms, a set of one dimensional curves, a multidimensional lookup table, and a set of one dimensional output curves. Data are processed using these elements via the sequence defined below.

("B" curves)->(matrix)->("M" curves)->(multidimensional lookup table)->("A" curves).

It is possible to use any or all of these processing elements. At least one processing element must be included. Some processing elements are not allowed in certain circumstances.

The following combinations are allowed for the lutBtoAType:

B
B - Matrix - M
B - CLUT - A
B - Matrix - M - CLUT - A

Other combinations may be achieved by setting processing element values to identity transforms.

When using this type, it is necessary to assign each color space component to an input and output channel. This assignment is specified in Table 43.

Table 50 — lutBtoAType encoding

Byte Offset	Content	Encoded as...
0..3	‘mBA ’ (6D424120h) [multi-function B-to-A table] type signature	
4..7	reserved, must be set to 0	
8	Number of Input Channels	uInt8Number
9	Number of Output Channels	uInt8Number
10-11	Reserved for padding, must be set to 0	
12..15	Offset to first "B" curve*	uInt32Number
16..19	Offset to matrix	uInt32Number
20..23	Offset to first "M" curve*	uInt32Number
24..27	Offset to CLUT	uInt32Number
28..31	Offset to first "A" curve*	uInt32Number
32..n	First data entry	

Each curve and processing element must start on a 4-byte boundary. To achieve this, each item may be followed by up to three 00h pad bytes as needed.

The offset entries (bytes 12-31) point to the various processing elements found in the tag. The offsets indicate the number of bytes from the beginning of the tag to the desired data. If any of the offsets are zero, that processing element is not present and the operation is treated as an identity transformation.

Either the XYZ or Lab PCS may be used with this tag type.

6.5.10.1 "B" Curves

There are the same number of "B" curves as there are input channels. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "B" curve is stored as an embedded curveType tag or a parametricCurveType. The length is as indicated by the convention of the proper curve type. Note that the entire tag type, including the tag type signature and reserved bytes, is included for each curve.

6.5.10.2 Matrix

The matrix is organized as a 3x4 array. The elements of the matrix appear in the type in order from e1-e12. The matrix elements are each s15Fixed16Numbers.

$$\text{array} = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}]$$

The matrix is used to convert data to a different color space, according to the following equation:

$$\begin{bmatrix} Y1 \\ Y2 \\ Y3 \end{bmatrix} = \begin{bmatrix} e1 & e2 & e3 \\ e4 & e5 & e6 \\ e7 & e8 & e9 \end{bmatrix} \bullet \begin{bmatrix} X1 \\ X2 \\ X3 \end{bmatrix} + \begin{bmatrix} e10 \\ e11 \\ e12 \end{bmatrix} \quad (30)$$

Each input X1, X2, or X3 is an u16Fixed16Number and each matrix entry is a s15Fixed16Number. Therefore, each multiplication in the matrix multiply is 1.15*s15.16=s16.31 and the final sum is also s16.31(48 bits). From this sum we take bits 31-16 as the unsigned integer result for YI, Y2, or Y3. These are used as the inputs to the "M" curves.

The matrix is allowed only if the number of input channels, or "B" curves, is 3.

6.5.10.3 "M" Curves

There are the same number of "M" curves as there are input channels. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each 'M' curve is stored as an embedded curveType or a parametricCurveType. The length is as indicated by the convention of the proper curve type. Note that the entire tag type, including the tag type signature and reserved bytes, are included for each curve. The "M" curves may only be used when the matrix is used.

6.5.10.4 CLUT

The CLUT appears as an n-dimensional array, with each dimension having a number of entries corresponding to the number of grid points.

The CLUT values are arrays of 8 bit or 16 unsigned values, normalized to the range of 0-255 or 65535.

The CLUT is organized as an n-dimensional array with a variable number of grid points in each dimension, where n is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is an m-integer array, where m is the number of output channels. The first sequential integer of the entry contains the function value for the first output function, the second sequential integer of the entry contains the function value for the second output function and so on until all of the output functions have been supplied. The equation for computing the byte size of the CLUT is:

nGrid1 * nGrid2 *...* nGridN * number of output channels * sizeof (channel component) (31)

Table 51 — lutBtoAType CLUT encoding

Byte Offset	Content	Encoded as...
0..15	Number of grid points in each dimension. Only the first n entries are used, where n is the number of input channels. Unused entries shall be set to 00h.	uInt8Number[16]
16	Precision of data elements in bytes. Must be either 01h or 02h.	uInt8Number
17..19	Reserved for padding.	
20..n	CLUT data points (arranged as described in the text).	uInt8Number[...] or uInt16Number[...]

If the number of grid points in a one-dimensional table, or in a particular dimension of the CLUT, is two, the data for those points shall be set so that the correct results are obtained when linear interpolation is used to generate intermediate values.

If the number of input channels does not equal the number of output channels, the CLUT must be present.

6.5.10.5 "A" Curves

There are the same number of "A" curves as there are output channels. The "A" curves may only be used when the CLUT is used. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "A" curve is stored as an embedded curveType or a parametricCurveType. The length is as indicated by the convention of the proper curve type. Note that the entire tag type, including the tag type signature and reserved bytes, is included for each curve.

6.5.11 measurementType

The measurementType information refers only to the internal profile data and is meant to provide profile makers an alternative to the default measurement specifications.

Table 52 — measurementType structure

Byte Offset	Content	Encoded as...
0..3	'meas' (6D656173h) type signature	
4..7	reserved, must be set to 0	
8..11	encoded value for standard observer	see below
12..23	XYZ tristimulus values for measurement backing	XYZNumber
24..27	encoded value for measurement geometry	see below
28..31	encoded value for measurement flare	see below
32..35	encoded value for standard illuminant	see below

The encoding for the standard observer field is such that:

Table 53 — Standard observer encodings

Standard Observer	Encoded Value
unknown	00000000h
CIE 1931 standard colorimetric observer	00000001h
CIE 1964 standard colorimetric observer	00000002h

The encoding for the measurement geometry field is such that:

Table 54 — Measurement geometry encodings

Geometry	Encoded Value
unknown	00000000h
0/45 or 45/0	00000001h
0/d or d/0	00000002h

The encoding for the measurement flare value is shown below and is equivalent to the basic numeric type u16Fixed16Number in 5.3.4.

Table 55 — Measurement flare encodings

Flare	Encoded Value
0 (0%)	00000000h
1,0 (or 100%)	00010000h

The encoding for the standard illuminant field is such that:

Table 56 — Standard illuminant encodings

Standard Illuminant	Encoded Value
unknown	00000000h
D50	00000001h
D65	00000002h
D93	00000003h
F2	00000004h
D55	00000005h
A	00000006h
Equi-Power (E)	00000007h
F8	00000008h

6.5.12 multiLocalizedUnicodeType

This tag structure contains a set of multilingual Unicode strings associated with a profile. Each string in the set is stored in a separate record with the information about what language and region the string is for.

Table 57 — multiLocalizedUnicodeType

Byte Offset	Content	Encoded as...
0..3	'mluc' (0x6D6C7563) type signature	
4..7	reserved, must be set to 0	
8..11	number of names: the number of name records that follow.	uInt32Number
12..15	name record size: the length in bytes of each name record that follows. Each name record currently consists of the fields first name language code to first name offset.	uInt32Number
16..17	first name language code: language code from ISO-639	uInt16Number
18..19	first name country code: region code from ISO-3166	uInt16Number
20-23	first name length: the length in bytes of the string	uInt32Number
24..27	first name offset: the offset from the start of the tag in bytes	uInt32Number
28..28+12n-1	if more than one name record, store them here	
28+12n...end	Storage area of Unicode characters	

Note that the third field of this tag, the name record size should, for the time being, contain the value 12, which corresponds to the size in bytes of each name record. Any code that needs to access the nth name record should determine the record's offset by multiplying n by the contents of this size field and adding 16. This minor extra effort allows for future expansion of name records, should the need arise, without having to define yet another new tag type.

For the specification of Unicode, see The Unicode Standard published by The Unicode Consortium or visit their website at http://www.unicode.org. For the definition of language code and region codes, see ISO-639 and ISO-3166. The Unicode strings in storage are encoded as 16-bit big-endian, UTF-16BE, and should not be NULL terminated.

If the specific string for the desired region is not stored in the tag, the string with the same language code should be used. If the specific string for the desired language is not stored in the tag, the first string in the tag is used if no other user preference is available.

6.5.13 namedColor2Type

The namedColor2Type is a count value and array of structures that provide color coordinates for 7-bit ASCII color names. For each named color, a PCS and optional device representation of the color are given. Both representations are 16-bit values. The device representation corresponds to the header's "color space of data" field. This representation should be consistent with the "number of device components" field in the namedColor2Type. If this field is 0, device coordinates are not provided. The PCS representation corresponds to the header's PCS field. The PCS representation is always provided. Color names are fixed-length, 32-byte fields including null termination. In order to maintain maximum portability, it is strongly recommended that special characters of the 7-bit ASCII set not be used.

Table 58 — namedColor2Type encoding

Byte Offset	Content	Encoded as...
0..3	'ncl2' (6E636C32h) type signature	
4..7	reserved, must be set to 0	
8..11	vendor specific flag (least-significant 16 bits reserved for ICC use)	
12..15	count of named colors	uInt32Number
16..19	number of device coordinates for each named color	uInt32Number
20..51	prefix for each color name (32-byte field including null termination)	7-bit ASCII
52..83	suffix for each color name (32-byte field including null termination)	7-bit ASCII
84..115	first color root name (32-byte field including null termination)	7-bit ASCII
116..121	first named color's PCS coordinates. The encoding is the same as the encodings for the PCS color spaces as described in Annex A. Only 16-bit L*a*b*, encoded using legacy 16-bit PCS Lab encoding, and XYZ are allowed.	uInt16Number[3]
122..y	first named color's device coordinates. For each coordinate, 0000h represents the minimum value for the device coordinate and FFFFh represents the maximum value for the device coordinate. The number of coordinates is given by the "number of device coordinates" field. If the "number of device coordinates" field is 0, this field is not given.	uInt16Number[...]
y+1..z	if *count* > 1 the remaining *count*-1 colors are described in a manner consistent with the first named color, see byte offsets 84..y.	

For color values that are in the Lab color space on the PCS side of the tag, this tag uses a legacy 16-bit Lab encoding, not the 16-bit CIELAB PCS encoding that is defined in Annex A.2. This encoding is retained for backwards compatibility with profile version 2.

To convert color values from this tag's legacy 16-bit Lab encoding to the 16-bit CIELAB PCS encoding defined in Annex A.2, multiply all values with 65535/65280 (that is, FFFFh/FF00h). Any color values that are in the value range of legacy 16-bit PCS Lab, but not in the 16-bit CIELAB PCS encoding defined in Annex A.2, shall be clipped on a per-component basis when transforming from legacy 16-bit PCS Lab to the 16-bit CIELAB PCS encoding defined in Annex A.2. To convert color values from the 16-bit CIELAB PCS encoding defined in Annex A.2 to this tag's legacy 16-bit Lab encoding, divide all values with 65535/65280.

The L* values have a different encoding than the a* and b* values. The L* encoding is:

Table 59 — L* encoding

Value (L*)	16 bit
0	0000h
100,0	FF00h
100 + (25500/65280)	FFFFh

Although the 16-bit encoding can represent values slightly greater than 100,0, these are not valid PCS L* values and they should not be used.

The a* and b* encoding is:

Table 60 — a* or b* encoding

Value (a* or b*)	16-bit
-128,0	0000h
0	8000h
127,0	FF00h
127 + (255/256)	FFFFh

Note that the 16-bit encoding can represent values slightly greater than 127,0. Since a* and b* have no defined limits, these are valid PCS values.

6.5.14 parametricCurveType

The parametricCurveType describes a one-dimensional curve by specifying one of a predefined set of functions using the parameters. The byte stream is as follows:

Table 61 — parametricCurveType encoding

Byte Offset	Content	Encoded as...
0..3	'para' (70617261h) type signature	
4..7	reserved, must be set to 0	
8..9	function type	uInt16Number (see below)
10..11	reserved, must be set to 0	
12..end	one or more parameters (see below)	s15Fixed16Number [...]

The encoding for the function type field and the parameters are such that:

Table 62 — parametricCurveType function type encoding

Function type	Encoded value	Parameters	Note
$Y = X^{\gamma}$	0000h	γ	
$Y = (aX + b)^{\gamma} \quad (X \geq -b/a)$ $Y = 0 \quad (X < -b/a)$	0001h	$\gamma\ a\ b$	CIE 122-1966
$Y = (aX + b)^{\gamma} + c \ (X \geq -b/a)$ $Y = c \quad (X < -b/a)$	0002h	$\gamma\ a\ b\ c$	IEC 61966-3
$Y = (aX + b)^{\gamma} \quad (X \geq d)$ $Y = cX \quad (X < d)$	0003h	$\gamma\ a\ b\ c\ d$	IEC 61966-2.1 (sRGB)
$Y = (aX + b)^{\gamma} + e \ (X \geq d)$ $Y = (cX + f) \quad (X < d)$	0004h	$\gamma\ a\ b\ c\ d\ e\ f$	

NOTE More functions can be added as necessary.

The order of the parameters in the tag data, Table 61, follows the left-to-right order of the parameters in Table 62.

The domain and range of each function shall be [0,0 1,0]. Any function value outside the range shall be clipped to the range of the function. When unsigned integer data is supplied as input, it shall be converted to the domain be dividing it by a factor of $(2^N) -1$, where N is the number of bits used to represent the input data. When unsigned integer data is required as output, it shall be converted from the range by multiplying it by a factor of $(2^M) -1$, where M is the number of bits used to represent the output data.

6.5.15 profileSequenceDescType

This type is an array of structures, each of which contains information from the header fields and tags from the original profiles which were combined to create the final profile. The order of the structures is the order

in which the profiles were combined and includes a structure for the final profile. This provides a description of the profile sequence from source to destination, typically used with the DeviceLink profile.

Table 63 — profileSequenceDescType structure

Byte Offset	Content
0..3	'pseq' (70736571h) type signature
4..7	reserved, must be set to 0
8..11	count value specifying number of description structures in the array
12..end	*count* profile description structures

Each profile description structure has the format:

Table 64 — Profile Description structure

Byte Offset	Content
0..3	Device manufacturer signature (from corresponding profile's header)
4..7	Device model signature (from corresponding profile's header)
8..15	Device attributes (from corresponding profile's header)
16..19	Device technology information such as CRT, Dye Sublimation, etc. (corresponding profile's technology signature)
20..m	displayable description of device manufacturer (corresponding profile's deviceMfgDescTag)
m+1..n	displayable description of device model (corresponding profile's deviceModelDescTag)

If the deviceMfgDescTag and/or deviceModelDescTag is not present in a component profile, then a "placeholder" tag should be inserted. This tag should have a 0 in the number of names field in the multiLocalizedUnicodeType structure with no name record or strings.

Also note that the entire tag, including the tag type, should be stored.

If the technologyTag is not present, bytes 16..19 should be 00000000h.

6.5.16 responseCurveSet16Type

ICC profiles for display and output devices will produce the desired color only while the device has a particular relationship between normalized device codes and physical colorant amount (the reference response). If the response of the device changes (the current response), the profile will no longer produce the correct result. In many cases it is impractical to produce a new profile for the current response, but the change can be compensated for by modifying the single channel device codes.

The purpose of this tag type is to provide a mechanism to relate physical colorant amounts with the normalized device codes produced by lut8Type or lut16Type tags so that corrections can be made for variation in the device without having to produce a new profile. The mechanism can be used by applications to allow users with relatively inexpensive and readily available instrumentation to apply corrections to individual output color channels in order to achieve consistent results.

Two pieces of information are necessary for this compensation: the reference response and the current response. This tag type provides a mechanism that allows applications that create profiles to specify the reference response. The way in which applications determine and make use of the current response is not specified at this time.

The measurements are of the standard variety used in the photographic, graphic arts, and television industries for process control. The measurements are intended to represent colorant amounts and so different measurement techniques are appropriate for different device types.

It is the job of the profile creator to provide reference response data in as many measurement units as practical and appropriate so that applications may select the same units that are measured by the user's instrument. Since it is not possible in general to translate between measurement units, and since most instruments only measure in one unit, providing a wide range of measurement units is vital. The profile originator must decide which measurement units are appropriate for the device.

Here are some examples of suitable measurement units: For process colors, density should be reported. Red-filter density should be reported for the cyan channel, green-filter for the magenta channel, blue-filter for the yellow channel, and visual for the black channel. For other colorants, such as Spot colors or Hi-Fi colors, it is the responsibility of the profile creator to select the appropriate units of measure for the system being profiled. Several different density standards are used around the world, so it is important that profile creators report in as many different density units as possible. Examples of suitable density measurements are: Status T, Status E, Status I and DIN.

This structure relates normalized device codes that would result from a lut16Type tag with density measurements of the resulting colorant amount. normalized device codes resulting from a lut8Type tag should first be multiplied by 257 (101h).

For those fields that have been structured in arrays of channel data, the channels are ordered as specified for the appropriate color space in Table 43.

Table 65 — responseCurveSet16Type structure

Byte Offset	Content	Encoded as...
0..3	'rcs2' (72637332h) [response curve set with 2-byte precision] type signature	
4..7	reserved, must be set to 0	
8..9	number of channels	uInt16Number
10..11	count of measurement types	uInt16Number
12..m	*count* relative offsets from byte 0 of this structure. Each will point to the response data for the measurement unit.	uInt32Number[...]
m+1..n	*count* response curve structures	see below

Each response curve structure has the format:

Table 66 — Curve structure

Byte Offset	Content	Encoded as...
0..3	measurement unit signature	see below
4..m	number of measurements for each channel: 4 = count of chan. 1 measurements, 8 = count of chan. 2 measurements, m-3 = count of final channel measurements	uInt32Number[...]
m+1..n	*number-of-channels* measurements of patch with the maximum colorant value	XYZNumber[...]
n+1..p	*number-of-channels* response arrays. Each array contains *number-of-measurements* response16Numbers appropriate to the channel	response16Number[...]

NOTE The XYZ values are CIE XYZ tristimulus values as described in 5.3.10. The response arrays must be ordered with normalized device code elements increasing.

The measurement unit is encoded as follows:

Table 67 — Curve measurement encodings

Measurement Unit	Signature	Hex Encoding
Status A: ANSI PH2.18 densitometer response. This is the accepted standard for reflection densitometers for measuring photographic color prints.	‘StaA’	53746141h
Status E: A densitometer response which is the accepted standard in Europe for color reflection densitometers.	‘StaE’	53746145h
Status I: A densitometer response commonly referred to as narrow band or interference-type response.	‘StaI’	53746149h
Status T: Wide band color reflection densitometer response which is the accepted standard in the United States for color reflection densitometers.	‘StaT’	53746154h
Status M: Standard densitometer response for measuring negatives.	‘StaM’	5374614Dh
DIN: Measurement according to DIN standard with no polarizing filter.	‘DN ’	444E2020h
DIN with Polarizing Filter	‘DN P’	444E2050h
Narrow band DIN	‘DNN ’	444E4E20h
Narrow band DIN with Polarizing filter	‘DNNP’	444E4E50h

6.5.17 s15Fixed16ArrayType

This type represents an array of generic 4-byte/32-bit fixed point quantity. The number of values is determined from the size of the tag.

Table 68 — s16Fixed16ArrayType encoding

Byte Offset	Content
0..3	‘sf32’ (73663332h) type signature
4..7	reserved, must be set to 0
8..end	an array of s15Fixed16Number values

6.5.18 signatureType

The signatureType contains a four-byte sequence used for signatures. Typically this type is used for tags that need to be registered and can be displayed on many development systems as a sequence of four characters. Sequences of less than four characters are padded at the end with spaces, 20h.

Table 69 — signatureType encoding

Byte Offset	Content
0..3	‘sig ’ (73696720h) type signature
4..7	reserved, must be set to 0
8..11	four-byte signature

6.5.19 textType

The textType is a simple text structure that contains a 7-bit ASCII text string. The length of the string is obtained by subtracting 8 from the element size portion of the tag itself. This string must be terminated with a 00h byte.

Table 70 — textType encoding

Byte Offset	Content
0..3	‘text’ (74657874h) type signature
4..7	reserved, must be set to 0
8..end	a string of (element size - 8) 7-bit ASCII characters

6.5.20 u16Fixed16ArrayType

This type represents an array of generic 4-byte/32-bit quantity. The number of values is determined from the size of the tag.

Table 71 — u16Fixed16ArrayType encoding

Byte Offset	Content
0..3	‘uf32’ (75663332h) type signature
4..7	reserved, must be set to 0
8..end	an array of u16Fixed16Number values

6.5.21 uInt16ArrayType

This type represents an array of generic 2-byte/16-bit quantity. The number of values is determined from the size of the tag.

Table 72 — uInt16ArrayType encoding

Byte Offset	Content
0..3	‘ui16’ (75693136h) type signature
4..7	reserved, must be set to 0
8..end	an array of unsigned 16-bit integers

6.5.22 uInt32ArrayType

This type represents an array of generic 4-byte/32-bit quantity. The number of values is determined from the size of the tag.

Table 73 — uInt32ArrayType encoding

Byte Offset	Content
0..3	‘ui32’ (75693332h) type signature
4..7	reserved, must be set to 0
8..end	an array of unsigned 32-bit integers

6.5.23 uInt64ArrayType

This type represents an array of generic 8-byte/64-bit quantity. The number of values is determined from the size of the tag.

Table 74 — uInt64ArrayType encoding

Byte Offset	Content
0..3	‘ui64’ (75693634h) type signature
4..7	reserved, must be set to 0
8..end	an array of unsigned 64-bit integers

6.5.24 uInt8ArrayType

This type represents an array of generic 1-byte/8-bit quantity. The number of values is determined from the size of the tag.

Table 75 — uInt8ArrayType encoding

Byte Offset	Content
0..3	‘ui08’ (75693038h) type signature
4..7	reserved, must be set to 0
8..end	an array of unsigned 8-bit integers

6.5.25 viewingConditionsType

This type represents a set of viewing condition parameters including: CIE ’absolute’ illuminant white point tristimulus values and CIE ’absolute’ surround tristimulus values.

Table 76 — viewingConditionsType encoding

Byte Offset	Content	Encoded as...
0..3	‘view’ (76696577h) type signature	
4..7	reserved, must be set to 0	
8..19	CIE ’absolute’ XYZ values for illuminant (in which Y is in cd/m^2)	XYZNumber
20..31	CIE ’absolute’ XYZ values for surround (in which Y is in cd/m^2)	XYZNumber
32..35	illuminant type	as described in measurementType

The viewing condition described in this tag is the actual viewing condition assumed for the media for which the profile is defined, specified in CIE absolute units. Note that the luminanceTag must be the same as the Y value given in this tag.

6.5.26 XYZType

The XYZType contains an array of three encoded values for the XYZ tristimulus values. The number of sets of values is determined from the size of the tag. The byte stream is given below. Tristimulus values must be non-negative. The signed encoding allows for implementation optimizations by minimizing the number of fixed formats.

.

Table 77 — XYZType encoding

Byte Offset	Content	Encoded as...
0..3	'XYZ ' (58595A20h) type signature	
4..7	reserved, must be set to 0	
8..end	an array of XYZ numbers	XYZNumber

Annex A
Color spaces

The International Color Profile Format supports a variety of both device-dependent and device-independent color spaces divided into three basic families: 1) CIEXYZ based, 2) RGB based, and 3) CMY based.

The CIE color spaces are defined in CIE publication 15.2 on Colorimetry. A subset of the CIEXYZ based spaces are also defined as connection spaces. The device dependent spaces below are only representative and other device dependent color spaces may be used without needing to update the profile format specification or the software that uses it.

Table 78 — CIE color spaces

Base Space	Description	Derivative Space
CIEXYZ	base CIE device-independent color space	CIELAB
GRAY	monochrome device-dependent color space	
RGB	base additive device-dependent color space	HLS, HSV
CMY	base subtractive device-dependent color space	CMYK

A.1 Profile Connection Spaces

A key component of these profiles is a well-defined profile connection space. This space is the interface which provides an unambiguous connection between the input and output profiles as illustrated in the diagram below. The profile connection space is based on the CIE 1931 standard colorimetric observer. This experimentally derived standard observer provides a very good representation of the human visual system color matching capabilities. Unlike device dependent color spaces, if two colors have the same CIE colorimetry they will match if viewed under the same conditions. Because the imagery is typically produced for a wide variety of viewing environments, it is necessary to go beyond simple application of the CIE system.

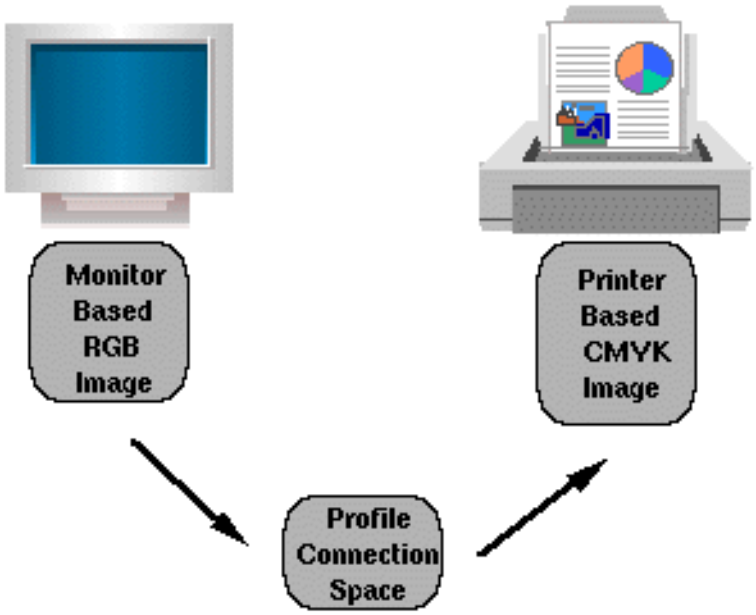


Figure 3 — Profile connection space illustration

The profile connection space is defined as the CIE colorimetry which, in the case of the perceptual rendering intent, will produce the desired color appearance if rendered on a reference imaging media and viewed in a reference viewing environment. This reference corresponds to an ideal reflection print viewed in an ISO P2 standard viewing booth.

The default measurement parameters for the profile connection space and all other color spaces defined in this specification are based on the ISO 13655 standard, “Graphic technology - Spectral measurement and colorimetric computation for graphic arts images.” Essentially this defines a standard illuminant of D50, the 1931 CIE standard colorimetric observer, and $0^o/45^o$ or $45^o/0^o$ reflectance measurement geometry. The reference viewing condition shall be ISO 3664 viewing condition P2 using the recommended 20% surround reflectance. This is a graphics arts and photography print comparison environment using a D50 illuminant at an illumination level of 500 lux.

One of the first steps in profile building involves determining the colorimetry of a set of colors from some imaging media or display. If the imaging media or viewing environment differ from the reference, it will be necessary to adapt the measured colorimetry to that appropriate for the profile connection space. These adaptations account for such differences as white point chromaticity and luminance relative to an ideal reflector, maximum density, viewing surround, viewing illuminant, and flare. Currently, it is the responsibility of the profile builder to do this adaptation.

However, the possibility of allowing a variable illuminant in the PCS is under active consideration by the International Color Consortium. For this reason, a PCS illuminant field is in the profile header, but must be set to the CIE Illuminant D50 [X=0,9642 Y=1,0000 Z=0,8249]. Note that the PCS illuminant field should not be confused with the viewing conditions tag defined in clause 6.4.36:“viewingCondDescTag” and clause 6.4.37:“viewingConditionsTag”.

The PCS is based on media-relative colorimetry. This is in comparison to ICC-absolute colorimetry. In ICC-absolute colorimetry colors are represented with respect to the illuminant, for example D50, and a perfect diffuser for reflecting and transmitting media. In media-relative colorimetry, colors are represented with respect to a combination of the illuminant and the media’s white, e.g. unprinted paper. The translation from media-relative colorimetry XYZ data, XYZ_r to ICC-absolute colorimetric data, XYZ_a, is given by

$$X_a = \left(\frac{X_{mw}}{X_i}\right) \cdot X_r \qquad \text{A.1.0.0.0-0}$$

$$Y_a = \left(\frac{Y_{mw}}{Y_i}\right) \cdot Y_r \qquad \text{A.1.0.0.0-1}$$

$$Z_a = \left(\frac{Z_{mw}}{Z_i}\right) \cdot Z_r \qquad \text{A.1.0.0.0-2}$$

where XYZ_{mw} represents the chromatically adapted media white point as found in the mediaWhitePointTag and XYZ_i represents the PCS illuminant white (which is D50).

The actual media and actual viewing conditions will typically differ from the reference conditions. The profile specification defines tags which provide information about the actual white point and black point of a given media or display. These tags may be used by a CMM to provide functionality beyond that of the default. For example, an advanced CMM could use the tags to adjust colorimetry based on the Dmin of a specific media. A tag is also provided to describe the viewing environment. This information is useful in choosing a profile appropriate for the intended viewing method.

A.2 PCS Encodings

There are many ways of encoding CIE colorimetry. This specification provides two methods in order to satisfy conflicting requirements for accuracy and storage space. These encodings, a CIELAB encoding and a 16-bit/component CIEXYZ encoding, are described below. The CIEXYZ space represents a linear transformation of the average color matching data, obtained by mixing red, green and blue lights to match all spectral colors, derived experimentally in the 1920s. The CIELAB space represents a transformation of the CIEXYZ space into one that is nearly perceptually uniform. This uniformity allows color errors to be equally weighted throughout its domain. While supporting multiple CIE encodings increases the complexity of color management, it provides immense flexibility in addressing different user requirements such as color accuracy and memory footprint.

The relationship between PCS CIEXYZ and PCS CIELAB is given by the usual set of equations as defined in ISO 13655:1996 but where the media white point (rather than the illuminant) is used as the relevant white point. Thus:

X/Xn is replaced by Xr/Xi (or Xa/Xmw) A.2.0.0.0-1

Y/Yn is replaced by Yr/Yi (or Ya/Ymw) A.2.0.0.0-2

Z/Zn is replaced by Zr/Zi (or Za/Zmw) A.2.0.0.0-3

where X/Xn, Y/Yn, and Z/Zn are defined in ISO 13655:1996, Annex B, section B.1.

NOTE The equations are as follows:

$L^* = 116[f(Y/Y_n)] - 16$ A.2.0.0.0-4

$a^* = 500[f(X/X_n) - f(Y/Y_n)]$ A.2.0.0.0-5

$b^* = 200[f(Y/Y_n) - f(Z/Z_n)]$ A.2.0.0.0-6

for: $X/X_n > 0{,}008856$, $f(X/X_n) = (X/Xn)^{1/3}$ A.2.0.0.0-7

$Y/Y_n > 0{,}008856$, $f(Y/Y_n) = (Y/Yn)^{1/3}$ A.2.0.0.0-8

$Z/Z_n > 0{,}008856$, $f(Z/Z_n) = (Z/Zn)^{1/3}$ A.2.0.0.0-9

for: $X/X_n <= 0{,}008856$, $f(X/X_n) = 7{,}7870(X/X_n)+16/116$ A.2.0.0.0-10

$Y/Y_n <= 0{,}008856$, $f(Y/Y_n) = 7{,}7870(Y/Y_n)+16/116$ A.2.0.0.0-11

$Z/Z_n <= 0{,}008856$, $f(Z/Z_n) = 7{,}7870(Z/Z_n)+16/116$ A.2.0.0.0-12

where $7{,}7870 = [29 / (6 * \mathrm{sqrt}(3))]^2$

It is important to understand that the PCS encodings do not represent a quantization of the connection space. The purpose of the encodings is to allow points within the space to be specified. Since the processing models benefit from interpolation between table entries, the interpolated AToB results should be used as the inputs to the BToA transforms. The AToB results should not be rounded to the nearest encoding value.

For the CIEXYZ encoding, each component (X, Y, and Z) is encoded as a fixed unsigned 16-bit quantity which has 15 fractional bits (u1.15).

An example of this encoding is:

Table 79 — CIEXYZ encoding

0	0000h
1,0	8000h
1 + (32767/32768)	FFFFh

The largest valid XYZ values are those of the PCS illuminant (specified in the profile header).[1)] This encoding was chosen to allow for PCS illuminants that have an X or Z greater than 1,0.

NOTE: CIE specifies that for reflecting and transmitting media Y should be normalized such that it has the value 100 for the perfect reflector or transmitter. In this specification, for reasons of coding efficiency, Y is specified such that it has the value 1 for the perfect reflector or transmitter.

For the CIELAB PCS encodings, the L* values have a different encoding than the a* and b* values. The L* encoding is:

Table 80 — CIELAB L* encoding

Value (L*)	**8-bit**	**16-bit**
0	00h	0000h
100,0	FFh	FFFFh

The a* and b* encoding is:

Table 81 — CIELAB a* or b* encoding

Value (a* or b*)	**8-bit**	**16-bit**
-128,0	00h	0000h
0	80h	8080h
127,0	FFh	FFFFh

Note that this is not "two's complement" encoding, but a linear scaling after an offset of 128. This encoding was chosen to prevent discontinuities in CLUTs when going from negative to positive values.

You can convert between the 8-bit and 16-bit encodings by multiplying or dividing by 257.

Because of the way that PCS encodings map to input tables, the BToAn tags must be able to handle invalid PCS values. However, the results of sending invalid values to these tags is up to the creator of the profile.

[1)] For a D50 illuminant, the largest valid XYZ values are [0,9642 1,0 0,8249], or [7B6Bh, 8000h, 6996h] in encoded form. Note that the PCS illuminant values are stored in s15.16 format, so you must translate them to u1.15 format to find the encoded PCS limits.

An important point to be made is that the PCS is not necessarily intended for the storage of images. A separate series of “interchange color spaces” may be defined in a future version of this specification for this purpose. The design choices made for these spaces (colorimetric encoding, reference media, viewing conditions, etc.) might be different than that of the PCS.

A.3 External and internal conversions

CMMs or other applications that use ICC tags to perform color transformations typically need to perform two types of data processing in addition to table interpolation. First, because the color values being processed (such as image pixels) may not match the native precision of an ICC tag (such as a lut16Type or lut8Type), it may be necessary to alter the precision of the input to (or results from) these transforms. Second, because there is more than one PCS encoding, it may be necessary to convert the output from a first transform before applying it to the input of a second transform. These two types of additional processing may be thought of as primarily affecting the **external** and **internal** interfaces of ICC processing, respectively.

In the first (external) case, the appropriate conversion method is to multiply each color value by $(2^M-1)/(2^N-1)$, where N is the starting number of bits and M is the required number of bits. This converts a number with values from 0 to (2^N-1) to a number with values from 0 to (2^M-1). For example, to prepare an 8-bit image value for input to a lut16Type tag the scale factor is $(2^{16}-1)/(2^8-1)$ = 65535,0/255,0 = 257,0. Note that the colors represented by the scaled numbers (be they device coordinates or some other color space) are not altered by the change in precision. For example, if a particular image value represents an L* of 31,0, then the scaled value also represents an L* of 31,0. Additionally, if an integer value is required from the scaling operation, it should be obtained via rounding rather than truncation.

In the second (internal) case, the appropriate conversion uses the CIE equations to convert between CIEXYZ and CIELAB.

Any colors in PCS XYZ encoding range that are outside of the PCS LAB encoding range shall be clipped on a per-component basis to the outside limits of the range of PCS LAB when transforming from XYZ into LAB. Conversely, any colors that occur in PCS LAB encoding range that are outside of the encoding range of PCS XYZ shall be clipped on a per-component basis to the PCS XYZ range when transforming from LAB into XYZ

A.4 Rendering Intents

In general, actual device gamuts will not be large enough to reproduce the desired color appearances communicated by the PCS values. Four rendering intents (gamut mapping styles) are provided to address this problem. Each one represents a different compromise. The colorimetric rendering intents operate on measurement-based colorimetric values as chromatically adapted to the PCS illuminant D50. This adaptation will usually be indicated in the chromaticAdaptationTag. The other rendering intents operate on colorimetric values which are corrected in an as-needed fashion to account for any differences between devices, media, and viewing conditions.

A.4.1 Colorimetric Intents

The colorimetric intents preserve the relationships between in-gamut colors at the expense of out-of-gamut colors. Mapping of out-of-gamut colors is not specified but should be consistent with the intended use of the transform.

A.4.1.1 MediaWhitePoint Tag

The mediaWhitePointTag contains CIE 1931 XYZ colorimetry of the white point of the actual medium. If the viewer completely adapts to the white point of the medium (as is often the case with monitors) this tag should be set to Xi, Yi, Zi (the PCS white point). If chromatic adaptation is being applied to the PCS values, the adaptation should be applied to the mediaWhitePointTag values as well.

A.4.1.2 Media-Relative Colorimetric Intent

This intent re-scales the in-gamut, chromatically adapted tristimulus values such that the white point of the actual medium is mapped to the PCS white point (for either input or output). It is useful for colors that have already been mapped to a medium with a smaller gamut than the reference medium (and therefore need no further compression).

A.4.1.3 ICC-Absolute Colorimetric Intent

For this intent, the chromatically adapted tristimulus values of the in-gamut colors are unchanged. It is useful for spot colors and when simulating one medium on another (proofing). Note that this definition of ICC-absolute colorimetry is actually called “relative colorimetry” in CIE terminology, since the data has been normalized relative to the perfect diffuser viewed under the same illumination source as the sample.

A.4.1.4 Applying the ICC-Absolute Colorimetric Intent

Profiles do not contain a separate transform for the ICC-absolute colorimetric intent. When this intent is needed, it is generated using the media-relative colorimetric intent and scaling the PCS values by the ratio of the destination profile mediaWhitePointTag to the source profile mediaWhitePointTag (as described in this annex). This applies to profiles using the TRC/matrix model, which is defined to use the media-relative colorimetric intent.

A.4.2 Perceptual Intent

The exact gamut mapping of the perceptual intent is vendor specific and involves compromises such as trading off preservation of contrast in order to preserve detail throughout the tonal range. It is useful for general reproduction of images, particularly pictorial or photographic-type images.

A.4.3 Saturation Intent

The exact gamut mapping of the saturation intent is vendor specific and involves compromises such as trading off preservation of hue in order to preserve the vividness of pure colors. It is useful for images which contain objects such as charts or diagrams.

Annex B
Embedding Profiles

This annex details the requirements and options for embedding device profiles within PICT, EPS, TIFF, JFIF, and GIF image files. All profiles except Abstract and DeviceLink profiles can be embedded. The complete profile must be embedded with all tags intact and unchanged.

B.1 Embedding ICC profiles in PICT files

Apple has defined a QuickDraw picture comment type for embedded ICC profiles. The picture comment value of 224 is followed by a 4-byte selector that describes the type of data in the comment. Using a selector allows the flexibility to embed more CMM related information in the future. The following selectors are currently defined:

Table 82 — PICT selectors

Selector	Description	
0	Beginning of an ICC profile.	Profile data to follow.
1	Continuation of ICC profile data.	Profile data to follow.
2	End of ICC profile data.	No profile data follows.

Because the dataSize parameter of the PicComment procedure is a signed 16-bit value, the maximum amount of profile data that can be embedded in a single picture comment is 32763 bytes (32767 - 4 bytes for the selector). You can embed a larger profile by using multiple picture comments of selector type 1. The profile data must be embedded in consecutive order, and the last piece of profile data must be followed by a picture comment of selector type 2.

All embedded ICC profiles, including those that fit within a single picture comment, must be followed by the end-of-profile picture comment (selector 2), as shown in the following examples.

EXAMPLE 1: Embedding a 20K profile.

```
PicComment kind = 224, dataSize = 20K + 4, selector = 0, profile data = 20K

PicComment kind = 224, dataSize = 4, selector = 2
```

EXAMPLE 2: Embedding a 50K profile.

```
PicComment kind = 224, dataSize = 32K, selector = 0, profile data = 32K - 4 PicComment
kind = 224, dataSize = 18K + 8, selector = 1, profile data = 18K + 4

PicComment kind = 224, dataSize = 4, selector = 2
```

In ColorSync 1.0, picture comment types `CMBeginProfile` (220) and `CMEndProfile` (221) are used to begin and end a picture comment. The `CMBeginProfile` comment is not supported for ICC profiles; however, the `CMEndProfile` comment can be used to end the current profile and begin using the System Profile for both ColorSync 1.0 and 2.0.

The `CMEnableMatching` (222) and `CMDisableMatching` (223) picture comments are used to begin and end color matching in both ColorSync 1.0 and 2.0

See “Advanced Color Imaging on the Mac OS”, Apple Computer 1995, for more information about picture comments.

B.2 Embedding ICC profiles in EPS files

There are two places within EPS files that embedding International Color Consortium (ICC) profiles are appropriate. 1) Associated with a screen preview. 2) Associated with the page description. Embedding ICC profiles within a screen preview is necessary so that applications using this screen preview to display a representation of the EPS page description can do so with accurate colors. Embedding ICC profiles within a page description is necessary so that sophisticated applications, such as OPI server software, can perform color conversions along with image replacement. For general information concerning PostScript's Document Structuring Conventions (DSC), the EPS file format, or specific PostScript operators, see the PostScript Language Reference Manual, second edition.

1) There are a variety of different methods of storing a screen preview within an EPS file depending on the intended environment. For cross platform applications with embedded ICC profiles, TIFF screen previews are recommended. The TIFF format has been extended to support the embedding of ICC profiles. ICC profiles can also be embedded in a platform specific manner. For example on the Macintosh, Apple has defined a method for embedding ICC profiles in PICT files (see B.1).

Note that a given page description may use multiple distinct color spaces. In such cases, color conversions must be performed to a single color space to associate with the screen preview.

2) ICC profiles can also be embedded in the page description portion of an EPS file using the `%%BeginICCProfile:` / `%%EndICCProfile` comments. This convention is defined as follows.

```
%%BeginICCProfile:  <profileid> <numberof> [<type> [<bytesorlines>]]
<profileid> ::= <text> (Profile ID)
<numberof> ::= <int> (Lines or physical bytes)
<type> ::= Hex | ASCII (Type of data)
<bytesorlines> ::= Bytes | Lines  (Read in bytes or lines)
%%EndICCProfile (no keywords)
```

These comments are designed to provide information about embedded ICC profiles. If the type argument is missing, ASCII data is assumed. ASCII refers to an ASCII base-85 representation of the data. If the `bytesorlines` argument is missing, `<numberof>` shall be considered to indicate bytes of data. If `<numberof>` = -1, the number of bytes of data are unknown. In this case, to skip over the profile one must read data until the encountering the `%%EndICCProfile` comment.

`<profileID>` provides the profile's ID in order to synchronize it with PostScript's setcolorspace and findcolorrendering operators and associated operands (see below). Note that `<numberof>` indicates the bytes of physical data, which vary from the bytes of virtual data in some cases. With hex, each byte of virtual data is represented by two ASCII characters (two bytes of physical data). Although the PostScript interpreter ignores white space and percent signs in hex and ASCII data, these count toward the byte count.

Each line of profile data shall begin with a single percent sign followed by a space (%). This makes the entire profile section a PostScript language comment so the file can be sent directly to a printer without modification. The space avoids confusion with the open extension mechanism associated with DSC comments.

ICC profiles can be embedded within EPS files to allow sophisticated applications, such as OPI server software, to extract the profiles, and to perform color processing based on these profiles. In such situations it is desirable to locate the page description's color space and rendering intent, since this color space and rendering intent may need to be modified based on any color processing. The `%%BeginSetColorSpace:`

/ %%EndSetColorSpace and %%BeginRenderingIntent: / %%EndRenderingIntent comments are used to delimit the color space and rendering intent respectively.

```
%%BeginSetColorSpace: <profileid>
<profileid> ::= <text> (ICC Profile ID)
%%EndSetColorSpace (no keywords)
```

<profileid> provides the ICC profile's ID corresponding to this color space. The ICC profile with this profile must have occurred in the PostScript job using the %%BeginICCProfile: / %%EndICCProfile comment convention prior to this particular %%BeginSetColorSpace: comment.

An example usage is shown here for CIE 1931 (XYZ)-space with D65 white point that refers to the ICC profile with <profileid> = XYZProfile.

```
%%BeginSetColorSpace: XYZProfile
[/CIEBasedABC <<
/WhitePoint [0.9505 1 1.0890]
/RangeABC [0 0.9505 0 1 0 1.0890]
/RangeLMN [0 0.9505 0 1 0 1.0890]
>>] setcolorspace
%%EndSetColorSpace
```

Note that the setcolorspace command is included within the comments. The PostScript enclosed in these comments shall not perform any other operations other than setting the color space and shall have no side effects.

```
%%BeginRenderingIntent: <profileid>
<profileid> ::= <text>            (ICC Profile ID)
%%EndRenderingIntent (no keywords)
```

<profileid> provides the ICC profile's ID corresponding to this rendering intent. The ICC profile with this profile must have occurred in the PostScript job using the %%BeginICCProfile: / %%EndICCProfile comment convention prior to invocation of this particular %%BeginRenderingIntent: comment.

An example usage is shown here for the Perceptual rendering intent that refers to the ICC profile with <profileid> = RGBProfile.

```
%%BeginRenderingIntent: RGBProfile
/Perceptual findcolorrendering pop
/ColorRendering findresource setcolorrendering
%%EndRenderingIntent
```

Note that the setcolorrendering command is included within the comments. The PostScript enclosed in these comments shall not perform any other operations other than setting the rendering intent and shall have no side effects.

B.3 Embedding ICC profiles in TIFF files

The discussion below assumes some familiarity with TIFF internal structure. It is beyond the scope of this document to detail the TIFF format, and readers are referred to the “TIFF™ Revision 6.0” specification, which is available from Adobe Systems Incorporated.

The International Color Consortium has been assigned a private TIFF tag for purposes of embedding ICC device profiles within TIFF image files. This is not a required TIFF tag, and Baseline TIFF readers are not currently required to read it. It is, however, strongly recommended that this tag be honored.

An ICC device profile is embedded, in its entirety, as a single TIFF field or Image File Directory (IFD) entry in the IFD containing the corresponding image data. An IFD should contain no more than one embedded profile. A TIFF file may contain more than one image, and so, more than one IFD. Each IFD may have its own embedded profile. Note, however, that Baseline TIFF readers are not required to read any IFDs beyond the first one.

The structure of the ICC Profile IFD Entry is as follows.

Table 83 — ICC profile IFD entry structure

Byte Offset	Content
0..1	The TIFFTag that identifies the field = 34675(8773.H)
2..3	The field Type = 7 = UNDEFINED (treated as 8-bit bytes).
4..7	The Count of values = the size of the embedded ICC profile in bytes.
8..11	The Value Offset = the file offset, in bytes, to the beginning of the ICC profile.

Like all IFD entry values, the embedded profile must begin on a 2-byte boundary, so the Value Offset will always be an even number.

A TIFF reader should have no knowledge of the internal structure of an embedded ICC profile and should extract the profile intact.

B.4 Embedding ICC profiles in JPEG files

The JPEG standard (ISO/IEC 10918-1) supports application specific data segments. These segments may be used for tagging images with ICC profiles. The APP2 marker is used to introduce the tag. Given that there are only 15 supported APP markers, there is a chance of many applications using the same marker. ICC tags are thus identified by beginning the data with a special null terminated byte sequence, "ICC_PROFILE".

The length field of a JPEG marker is only two bytes long; the length of the length field is included in the total. Hence, the values 0 and 1 are not legal lengths. This would limit maximum data length to 65533. The identification sequence would lower this even further. As it is quite possible for an ICC profile to be longer than this, a mechanism must exist to break the profile into chunks and place each chunk in a separate marker. A mechanism to identify each chunk in sequence order would thus be useful.

The identifier sequence is followed by one byte indicating the sequence number of the chunk (counting starts at 1) and one byte indicating the total number of chunks. All chunks in the sequence must indicate the same total number of chunks. The one-byte chunk count limits the size of embeddable profiles to 16707345 bytes.

B.5 Embedding ICC profiles in GIF files

The GIF89a image file format supports Application Extension blocks, which are used for "application specific" information. These blocks may be used for tagging images with ICC profiles.

The Application Identifier for an embedded profile shall be the following 8 bytes: "ICCRGBG1". The Authentication Code shall be "012". The entire profile shall be embedded as application data, using the conventional technique of breaking the data into chunks of at most 255 bytes of data.

Annex C
Relationship between ICC Profiles and PostScript CSAs and CRDs

C.1 Introduction

When ICC profiles are used to generate PostScript color space arrays (CSAs) or color rendering dictionaries (CRDs) it is useful to be able to identify the profile used to define the CSA or CRD. This can be achieved by adding the following keys to the CSA or CRD. This mechanism does not rely on comments, and enables a parser to obtain the original profile from outside the PostScript file.

C.2 Profile identification keys for a PostScript CSA

The following keys are recommended by Adobe Systems for inclusion in PostScript (and EPS) color space arrays.

/CreationDate (string)

Identifies the date and time at which the color space array was created or most recently modified.

The value of this entry should be coordinated with the calibrationDateTimeTag attribute of any associated ICC profile, and its syntax should conform to the international standard ASN.1, defined in the document ISO/IEC 8824.

/RenderingIntent (name or string)

Identifies the rendering intent that this color space array is designed to achieve. Must be one of: AbsoluteColorimetric, RelativeColorimetric, Saturation or Perceptual.

/Description (string)

7-bit ASCII description string from the ICC profile 'desc' tag.

/Copyright (string)

7-bit ASCII copyright string from the ICC profile 'cprt' tag.

Note: In version 4 profiles, the copyright and description strings are multi-lingual. Only the U.S. English string from the ICC profile is present in the CSA/CRD. If the ICC profile does not contain a U.S. English string, one should be computed from the first multi-lingual string.

/ColorSpace (string)

Color model of the profile data from the ICC profile header. Must be the 4-character ASCII string representing the ColorSpace signature (see section 6.1.5).

/ProfileID (hexadecimal string)

This is the Profile ID of the ICC profile. This shall be encoded as hexadecimal data, enclosed in < and >.

For version 4 profiles, Profile ID is generally present in the profile header. For those ICC profiles not containing Profile ID, a Profile ID should be computed using the method described in section 6.1.13.

Example CSA from Photoshop:

```
[ /CIEBasedABC
  <<
    /CreationDate    (19990603000000)
    /RenderingIntent (Perceptual)
    /Description     (not Adobe RGB (1998))
    /ColorSpace      (RGB )
    /Copyright       (Copyright 1999 Adobe Systems Incorporated)
    /ProfileID       <33BC7F1C156FA0D72F8F717AE5886BD4>
    /DecodeLMN       [{2.1992 exp}bind {2.1992 exp}bind {2.1992 exp}bind]
    /MatrixLMN       [0.3805 0.7083 0.9959
                      0.1282 0.0593 0.7144
                      0.4554 0.2324 0.0145]
    /WhitePoint      [0.9642 1.0000 0.8249]
  >>
]
```

C.3 Profile identification keys for a PostScript CRD

The following keys are recommended by the PostScript Language Reference Manual for inclusion in PostScript colorrendering dictionaries:

/CreationDate (string)

Identifies the date and time at which the color rendering dictionary was created or most recently modified.

The value of this entry should be coordinated with the calibrationDateTimeTag attribute of any associated ICC profile, and its syntax should conform to the international standard ASN.1, defined in the document ISO/IEC 8824-1.

/RenderingIntent (name or string)

Identifies the rendering intent that this color rendering dictionary is designed to achieve. Must be one of: AbsoluteColorimetric, RelativeColorimetric, Saturation or Perceptual.

The use of the following additional keys is also recommended in cases where it is important to establish a clear relationship between the CRD and the ICC profile from which it was derived.

/Description (string)

7-bit ASCII description string from the ICC profile 'desc' tag.

/Copyright (string)

7-bit ASCII copyright string from the ICC profile 'cprt' tag.

Note: In ICC4 Profiles, the copyright and description strings are multi-lingual. Only the U.S. English string from the ICC Profile is present in the CSA/CRD. If the ICC Profile does not contain a U.S. English string, one should be computed from the first multi-lingual string.

/ColorSpace (string)

Color Model of the profile data from the ICC profile header. Must be the 4-character ASCII string representing the ColorSpace signature (see section 6.1.5).

/ProfileID (string)

ASCII string representation of the hex-encoded Profile ID of the ICC Profile.

For version 4 profiles, Profile ID is generally present in the profile header. For those ICC profiles not containing Profile ID, a Profile ID should be computed using the method described in section 6.1.13.

Annex D
Profile Connection Space

The information necessary to adequately define the Profile Connection Space (PCS) is contained in annex D.1. While complete, this information is difficult to interpret without additional explanation and background material. This supporting information, along with examples and suggestions, is contained in annex D.2. These two types of material have been separated to facilitate future movement of the ICC Profile Specification into a formal standard in which normative information must be separated from informative information. Annex D.1 represents the normative information necessary to define the PCS. Annex D.2 would form the informative part of an ISO or IEC standard.

D.1 Requirements

D.1.1 The PCS Definition

The Profile Connection Space is the reference color space in which colors shall be encoded in order to provide an interface for connecting source and destination transforms. The PCS values shall constitute an encoding of a CIE colorimetric specification.

In perceptual transforms the PCS values shall represent hypothetical measurements of a color reproduction on a reference medium. By extension, for the perceptual intent, the PCS represents the appearance of that reproduction as viewed in the reference viewing environment by a human observer adapted to that environment.

In transforms for the media-relative colorimetric intent the PCS values shall represent media-relative measurements of the captured original (for input profiles), or media-relative color reproductions produced by the output device (for output profiles).

In transforms for the ICC-absolute colorimetric intent the PCS values shall represent measurements of the captured original relative to a hypothetical perfectly reflecting or transmitting diffuser (for input profiles), or color reproductions produced by the output device relative to a hypothetical perfectly reflecting or transmitting diffuser (for output profiles).

In transforms for the media-relative and ICC-absolute colorimetric intents, the PCS values may represent a color rendering of the actual original captured for input profiles. Likewise for output profiles, the PCS values may be color rendered by the output device to the actual medium. However, wherever ICC profiles are used, the PCS values resulting from such transforms shall be interpreted as the colorimetry of the original and reproduction, regardless of whether such colorimetry is the actual colorimetry.

D.1.2 PCS Colorimetric Specification

The colorimetric values used by the PCS shall be as described in Annex A. The colorimetry shall be assumed not to contain any flare or other defect caused by inadequacies in the optical system of the instrument and illumination used to make the measurements, but is assumed to include the surface reflection component normally associated with the prescribed measurement geometry.

D.1.3 PCS Encoding

In transforms for the media-relative colorimetric, perceptual, and saturation rendering intents (intents other than ICC-absolute colorimetric), the white point of the actual medium, and the white point of the reference medium are represented in PCS XYZ and PCS Lab formats as follows.

Table 84 — White point encodings

Component	Value	8-bit Encoding	16-bit Encoding		Component	Value	Encoding
L*	100	255	65535		**X**	0,9642	31595
a*	0	128	32896		**Y**	1,0000	32768
b*	0	128	32896		**Z**	0,8249	27030

In transforms for the media-relative colorimetric intent the perfect absorber (a theoretical medium that reflects absolutely no light) is represented in PCS XYZ and PCS Lab formats as follows. Other reflectance values are mapped linearly to PCS XYZ.

Table 85 — Perfect absorber encoding

Component	Value	8-bit Encoding	16-bit Encoding		Component	Value	Encoding
L*	0	0	0		**X**	0,0	0
a*	0	128	32896		**Y**	0,0	0
b*	0	128	32896		**Z**	0,0	0

In transforms for the perceptual and saturation intents the black point of the reference medium is represented in PCS XYZ and PCS Lab formats as follows. This is here called the PCS perceptual black point.

Table 86 — Black point encoding of reference media

Component	Value	8-bit Encoding	16-bit Encoding		Component	Value	Encoding
L*	3,1373	8	2056		**X**	0,00336	110
a*	0	128	32896		**Y**	0,0034731	114
b*	0	128	32896		**Z**	0,00287	94

NOTE: Due to limited numerical precision, Y encoded as 114 does not exactly match L* encoded as 8.

NOTE: An adjustment must be applied to older perceptual transforms that use a PCS value of zero to represent the black point of the reference medium. The zero point of such a transform needs to be mapped to the above PCS perceptual black point. The white point remains unchanged. All other values are mapped linearly in XYZ. The following equations can be used for the adjustment of such a transform to the above PCS encoding.

$$Xp = Xt * (1 - Xb / Xi) + Xb \qquad \text{D.1.3.0.0-0}$$

$$Yp = Yt * (1 - Yb / Yi) + Yb \qquad \text{D.1.3.0.0-1}$$

$$Zp = Zt \ *(1 - Zb / Zi) + Zb \qquad \text{D.1.3.0.0-2}$$

Where:

Xt, Yt, Zt = original PCS XYZ value in the transform
Xb, Yb, Zb = CIE XYZ values for the PCS perceptual black point (X = 0,00336, Y = 0,0034731, Z = 0,00287)
Xi, Yi, Zi = CIE XYZ values of the PCS white point (X = 0,9642, Y = 1,0000, Z = 0,8249)
Xp, Yp, Zp = the adjusted PCS XYZ value

D.1.4 The Reference Viewing Environment

The reference viewing condition applies only to the perceptual transform. If the actual viewing environment differs from the reference viewing environment perceptual transforms must compensate for the differences in viewing environments.

The reference viewing environment, as specified in A.1, shall be taken as the aim viewing environment underpinning the use of the PCS. It shall be based on standard viewing condition P2, as specified for graphic arts and photography in ISO 3664. It is characterized by an "average" surround, which means that the illumination of the image shall be assumed to be similar to the illumination of the rest of the environment. The surfaces immediately surrounding the image shall be assumed to be a uniform matte gray with a reflectance of 20%. ISO 3664 specifies that the illumination in the plane of viewing shall be that of CIE standard illuminant D50 within specific tolerances. The chromaticity of this illumination (x_{10} = 0,3478, y_{10} = 0,3595 in the CIE x_{10}, y_{10} chromaticity diagram) shall define the chromatic adaptation state associated with the PCS. ISO 3664 describes the appropriate illumination level for practical appraisal of prints as 500 lux and ISO 3664 describes the appropriate illumination level for practical appraisal of prints as 500 lux, which is specified to be typical of actual home and office viewing environments. This was deemed to be most appropriate for the reference viewing environment, which shall also be assumed to have a level of stray light such that the veiling glare observed by a viewer, in addition to that measured using the prescribed measurement geometry, (the viewing flare[1)]) is a factor of 0,0075 (3/4%) times the illumination level as reflected by the reference white, with the same chromaticity as the illumination (4,25 cd/m^2).

D.1.5 The Reference Medium

The reference medium applies only to the perceptual transform.

The reference medium shall be a hypothetical print on a substrate having a neutral reflectance of 89%. The darkest printable color on this medium shall have a neutral reflectance of 0,30911%, which is 0,34731% of the substrate reflectance. These are the white point and black point of the reference medium. The reference medium therefore has a linear dynamic range of 287,9 :1 and a density range of 2,4593.

D.2 Explanation and Background Material (informative)

The following material is provided to aid in the understanding and interpretation of the PCS requirements.

[1)] E. J. Giorgianni and T. E. Madden, *Digital Color Management*, Addison-Wesley, Reading, Massachusetts, 1998, p. 561.

D.2.1 Introduction

The concept of a Profile Connection Space is a vital element in the ICC architecture. It allows the profile transforms for input, display, and output devices to be decoupled so that they can be produced independently. A well-defined PCS provides the common interface for the individual device profiles. It is the virtual destination for input transforms and the virtual source for output transforms. If the input and output transforms are based on the same PCS definition, even though they are created independently, they can be paired arbitrarily at run time by the color-management engine and will yield consistent and predictable results when applied to color values.

The key to effective use of the profile specification is an unambiguous definition of the PCS. However, there is probably no definition that will yield optimal results for all possible color-management scenarios involving all possible input media, all possible output media, and all possible market preferences. Where trade-offs are necessary, the preference has been to serve the needs of applications in graphic arts and desktop publishing. For this reason the PCS definition is biased somewhat toward scenarios that result in output to reflection-print media such as offset lithography, off-press proofing systems, computer-driven printers of various kinds, and photographic paper. Even with this bias, the PCS will provide good results in other applications such as video production, slide production, and presentation graphics.

D.2.2 Encoding of PCS Measurements

In Annex A to the specification, the PCS is defined to be based on colorimetry relative to the media white point. These two factors are accommodated by the encoding part of the PCS definition.

Part of the PCS encoding, for the perceptual intent, normalizes the reference medium's white point to the PCS white point. This procedure corresponds to using media-relative colorimetry when the reference medium's white point is the media white point (media-relative colorimetric intent).

For the perceptual intent, the normalized CIE XYZ values of the reference medium black point are used as the color rendering target black point values. This provides a specific reference in the PCS for the black point and dynamic range of the target ideal reflection print.

The choice of a reference medium with a realistic black point for the perceptual intent provides a well-defined aim when tonal remapping is required. Inputs with a dynamic range greater than a reflection print (for example, a slide film image, or the colorimetry of high-range scenes) can have their highlights and shadows smoothly compressed to the range of the print in such a way that these regions can be expanded again without undue loss of detail on output to wide-range media. Note that while this does not impose a limit on the precision of the PCS values, it does require that appropriate precision be maintained in both the image data and the calculations using that data.

In transforms for the colorimetric intents, the range of valid (but not necessarily physically realizable) PCS XYZ values is unrelated to the reference media black point. Instead they reflect instrument readings without tonal remapping. In theory, the dynamic range of the PCS for colorimetric transforms is infinite.

All transforms in an output profile should be able to process all values in the PCS, regardless of whether the values are outside of the destination device gamut.

NOTE The PCS encoding defined here is different to that in version 3 of the ICC specification which defined the PCS as being the encoded colorimetry of an ideal reflection print on a spectrally non-selective substrate with 100% reflectance. This ideal print had an infinite dynamic range, since black could have 0% reflectance.

D.2.3 Color Measurements

In order to establish the relationship between the colorimetry encoded in the PCS (and oriented toward the reference medium and reference environment) and the measured colorimetry of an actual medium, intended for an actual viewing environment, it is useful to describe the measurement conditions more precisely.

In general, the actual viewing illumination source may have a spectral power distribution different from D50. In such cases, the actual illumination source should be used in the color measurements, or, equivalently, the actual illumination spectrum should be used in calculating tristimulus values from the measured spectral reflectances or transmittances of the medium. If the chromaticity of the illumination source is different from that of D50, corrections for chromatic adaptation may be needed and must be incorporated into the colorimetric transforms (see D.2.4 and D.2.6.1) and the perceptual transform (see section D.2.4 and D.2.7.7) by the profile builder. For example, an Alexandrite stone appears to be a purple color when viewed under tungsten illumination. The same stone appears to be sea-green when viewed under daylight. If an image of such a stone is captured under tungsten illumination, its PCS colorimetry (as produced by the input profile) should correspond to a purple color.

For media intended for the graphic arts, it is best that the color measurements conform to ISO 13655, Graphic technology - Spectral measurement and colorimetric computation for graphic arts images. Here, the illumination source spectral power distribution for the calculation of colorimetry is specified to be that of D50. No corrections for chromatic adaptation are required in this case, since the chromaticity of the illumination source is that of D50. Other corrections, as discussed below, may still be applicable. Note that the fluorescent D50 simulators found in typical professional viewing booths, although their chromaticity may agree closely with standard D50, may have rather different spectral distributions (different from each other and different from the CIE definition) so that the measured, or calculated, tristimulus values can vary noticeably. Often, a better description of the observed color can be obtained by basing the colorimetry on the actual, rather than the theoretical, illumination source[1)]. The CIE color rendering and metamerism index criteria specified in ISO 3664 can be used to determine if an actual source is sufficiently close to D50 to minimize spectrally caused visual effects. In critical applications, filtered tungsten D50 simulators may be the best choice to minimize these effects.

As specified in D.1.2, the measurements are assumed not to be contaminated with flare due to the use of low quality instruments or poor measurement technique. This does not imply that it is necessary to remove any surface reflections that are a typical component of $0^o/45^o$ measurements of reflection materials. It is important to note that the difference in flare between the specifications for measurement and viewing is neither a contradiction nor does it add complexity. It is simply a statement of current practice. The measurement conditions have been chosen so as to not require any corrections to high quality measurements of the type typically collected for color management purposes. Similarly, the 3/4% flare of the reference viewing environment was chosen since this is representative of the amount of stray light contributed by high quality, but realistic environments in actual use.

Because the PCS is more a specification of how to reproduce a desired appearance than it is a specification of the appearance itself, it is not necessary (or desirable) to add the 3/4% flare to the measurements before encoding a color in the PCS. Instead, the 3/4% viewing flare is specified to allow compensation for any potential difference between the actual viewing environment and the reference environment.

---

1) D. Walker, "The Effects of Illuminant Spectra on Desktop Color Reproduction', in *Device Independent Color Imaging,* R. Motta and H. Berberian, ed., *Proc. SPIE,* 1909**,** 1993, pp. 236-246.

D.2.4 Chromatic Adaptation

When a person is looking at a real-world scene, the color stimulus presented to the retina by any visible surface in the scene depends on the spectral composition of the light with which the surface is illuminated. This stimulus is what colorimetry attempts to measure, by stipulating how a mixture of three specified stimuli would match it, for a standard observer. If the illuminant is changed the stimulus will also change. Colorimetry measures the change in stimulus and predicts a different color. However, because of adaptation the appearance of the color does not change significantly - despite the change in stimulus incident on the eye - and this seems to indicate a serious limitation in colorimetry which was not created to measure appearance - but only whether two colors match. But this is not the case - because a change also occurs in the white point stimulus it can be used in defining metrics of appearance, of varying complexity, which can predict the change in appearance. To understand this we need some understanding of the way the visual system adapts so flexibly to the color and intensity of the incident light.

The mechanism can be modeled as follows: Through some means, the system infers the color and strength of the presumed illumination source. (In a normal scene, this inference may be based on specular highlights, or the apparent colors of known objects, or some kind of scene average, etc.; for reproductions, the inference may be made from the image itself or, as when viewing reflection prints, objects in the real world surrounding the image.) The system then uses this information to adjust the "gain" applied to the "cone responses" to the color stimuli (the actual process is not well understood, and is most likely more complicated). The result of this adaptation is that the signals received by the brain are much less dependent on the brightness and chromaticity of the illumination source, so that objects can be more easily recognized, regardless of whether the light source is bright or dim, yellowish or bluish, etc. The adaptive mechanism does not compensate perfectly for the change of illuminant, however, so objects do appear somewhat different under different illumination. Note that this mechanism is operative in bright environments; adaptation to the dark is a separate phenomenon.

There are several available models which may be used to represent this process, among them XYZ scaling, the von Kries transformation, the Bradford transformation, and CMCCAT97. The choice of which model to use depends upon the device and the environment in which it is used. Often this choice depends upon the difference in chromaticity between the illumination sources. If the difference is small, a simple model may be suitable, but if the difference is large, a more complicated model may be needed. In some cases it may be necessary to consider the effects of the use of differing methods in the source and destination profiles.

This aspect of the PCS definition provides some flexibility to the color management system as a whole. For example, it is possible to transform data from a medium intended for tungsten illumination to a medium intended for cool-white-fluorescent: the input profile handles the adaptation from tungsten to D50, and the output profile handles the adaptation from D50 to cool-white.

D.2.5 Aesthetic Considerations and the Media White Point

Aside from the adaptive effects mentioned above, there is frequently a strong aesthetic preference for maintaining highlight detail in all renderings of an image. One way to guarantee this result for typical reflection media is to modify the colorimetry of the reproduction so as to factor out the colorimetry of the substrate. This approach is called "media-relative colorimetry," i.e., colorimetry relative to the substrate. (In contrast, "ICC-absolute colorimetry" is called "relative colorimetry" in the CIE terminology, since the data are normalized relative to the perfect diffuser viewed under the same illumination source as the sample. See CIE Publications 15.2-1986, Colorimetry (second edition).) According to this media-relative method the PCS color [100, 0, 0] in CIELAB is associated with the blank substrate, regardless of its actual colorimetry, and all other colors are modified accordingly.

However, there are applications in which the goal is to reproduce the actual colors of an image (within the limitations of color gamut and dynamic range), even if highlight detail must be sacrificed. For instance, the goal may be to simulate one medium on another, for proofing purposes. In these cases, the "ICC-absolute", or "CIE relative", colorimetry is required. The ICC specification provides a mechanism for converting "media-relative" into "ICC-absolute" colorimetry. The profile mediaWhitePointTag defines the colorimetry of the actual substrate, corrected for differences in the viewing conditions, in CIE 1931 XYZ coordinates. (See section D.2.6.2.) Annex A describes the mechanism for using these coordinates in the required conversion.

If the white point mapping discussed above is present in both the input and the output transforms, the white point of the input medium will be mapped, by way of the PCS illumination source, to the white point of the output medium (media-relative colorimetric intent). The ICC-absolute colorimetric rendering intent will also be accurately enabled through the use of the mediaWhitePointTags.

For the perceptual intent, just as it is necessary to correct for viewing environment differences, we need to convert the colorimetry of the actual medium to that desired for the reference medium. This can include mapping the white point of the actual medium to the white point of the reference medium. The white point of the reference medium is then mapped to the PCS white point (see section D.1.3 and step 5 of section D.2.7.7).

In other cases, the goal may be to introduce color shifts which provide a unique aesthetic effect[1)]. In these cases the white point of the actual medium may be mapped to a color other than the white point of the reference medium. This is another means by which unique value may be added to profiles while maintaining data interoperability.

D.2.6 Discussion of Relative Colorimetric Intent

D.2.6.1 Relative and Absolute Intents

For ICC-absolute colorimetric transformations in the context of ICC profiles the media XYZ tristimulus values are reproduced relative to the illumination source or perfect diffuser. The reproduction provided by the ICC-absolute colorimetric intent is said to be illuminant-relative, and L* = I00 for the perfect diffuser. The PCS tristimulus values for an ICC-absolute colorimetric transform of course are also illuminant-relative, that is, PCS L* = 100 for the perfect diffuser. The profile format does not define an explicit transform for ICC-absolute colorimetric intent. For a given profile, the PCS-side XYZ tristimulus values for the ICC-absolute colorimetric intent are obtained from the media-relative colorimetric transform, see below.

For media-relative colorimetric transforms in the context of ICC profiles the media XYZ tristimulus values are reproduced relative to the media white point. Given the media-relative reproduction provided by the media-relative colorimetric intent, L* = 100 for media white. The PCS tristimulus values for a media-relative colorimetric transform are also media-relative, that is, PCS L* = 100 for media white.

The PCS-side XYZ tristimulus values of a colorimetric transform ($XYZ_{\text{mediawhite-relative under D50}}$ for media-relative colorimetric transforms, $XYZ_{\text{under D50}}$ for ICC-absolute colorimetric transforms) are calculated from CIE XYZ tristimulus values ($XYZ_{\text{illuminant-relative}}$) of the device under the actual illumination source ($XYZ_{\text{illuminant}}$)

1) E. J. Giorgianni and T. E. Madden, op. cit., p. 425.

When the actual illumination source differs from the PCS reference illumination source, CIE D50, chromatic adaptation as described in section D.2.4 is required for transforming tristimulus values between the two illumination sources. The PCS-side tristimulus values are obtained as follows:

$$XYZ_{\text{under D50}} = \text{ChromaticAdaptationMatrix}\,(XYZ_{D50}, XYZ_{\text{illuminant}}) * XYZ_{\text{illuminant-relative}} \qquad \text{D.2.6.1.0-1}$$

$$XYZ_{\text{under D50 mediawhite}} = \text{ChromaticAdaptationMatrix}\,(XYZ_{D50}, XYZ_{\text{illuminant}}) * XYZ_{\text{illuminant-relative mediawhite}} \qquad \text{D.2.6.1.0-2}$$

$$\begin{aligned} X_{\text{mediawhite-relative}} &= X_{D50} / X_{\text{under D50 mediawhite}} * X_{\text{under D50}} \\ Y_{\text{mediawhite-relative}} &= Y_{D50} / Y_{\text{under D50 mediawhite}} * Y_{\text{under D50}} \\ Z_{\text{mediawhite-relative}} &= Z_{D50} / Z_{\text{under D50 mediawhite}} * Z_{\text{under D50}} \end{aligned} \qquad \text{D.2.6.1.0-3}$$

Where:

> ChromaticAdaptationMatrix (illuminant 2, illuminant 1) is a 3 by 3 matrix that adjusts XYZ tristimulus values from illuminant 1 to illuminant 2 using chromatic adaptation, see section D.2.4

$XYZ_{\text{under D50 mediawhite}}$ must be stored in the mediaWhitePointTag, whether the illuminant is D50 or not.

ChromaticAdaptationMatrix should be stored in the chromaticAdaptationTag, 'chad' (63686164h), when the actual illumination source is not CIE D50.

Note that the scaling between media-relative and ICC-absolute colorimetric values is done under the PCS illumination source. Also note that the observer is assumed to be adapted to the perfect diffusers not to the media white.

If the actual illumination source is CIE D50, that is, the same as the PCS illumination source, the above equations are simplified to:

$$\begin{aligned} X_{\text{under D50}} &= X_{\text{illuminant-relative}} \\ Y_{\text{under D50}} &= Y_{\text{illuminant-relative}} \\ Z_{\text{under D50}} &= Z_{\text{illuminant-relative}} \end{aligned} \qquad \text{D.2.6.1.0-4}$$

$$\begin{aligned} X_{\text{under D50 mediawhite}} &= X_{\text{illuminant-relative mediawhite}} \\ Y_{\text{under D50 mediawhite}} &= Y_{\text{illuminant-relative mediawhite}} \\ Z_{\text{under D50 mediawhite}} &= Z_{\text{illuminant-relative mediawhite}} \end{aligned} \qquad \text{D.2.6.1.0-5}$$

$$\begin{aligned} X_{\text{mediawhite-relative}} &= X_{D50} / X_{\text{under D50 mediawhite}} * X_{\text{under D50}} \\ Y_{\text{mediawhite-relative}} &= Y_{D50} / Y_{\text{under D50 mediawhite}} * Y_{\text{under D50}} \\ Z_{\text{mediawhite-relative}} &= Z_{D50} / Z_{\text{under D50 mediawhite}} * Z_{\text{under D50}} \end{aligned} \qquad \text{D.2.6.1.0-6}$$

The ICC profile format does not include an explicit transform for ICC-absolute colorimetric intent. On creating a profile only the $XYZ_{\text{mediawhite-relative under D50}}$ values are stored in a profile, not $XYZ_{\text{under D50}}$. When using a profile, after obtaining the media-relative colorimetric transform of the profile, the PCS-side XYZ tristimulus values for the ICC-absolute colorimetric intent are calculated from the media-relative colorimetric transform through a simple scaling operation:

$$\begin{aligned} X_{\text{under D50}} &= X_{\text{under D50 mediawhite}} / X_{D50} * X_{\text{mediawhite-relative}} \\ Y_{\text{under D50}} &= Y_{\text{under D50 mediawhite}} / Y_{D50} * Y_{\text{mediawhite-relative}} \\ Z_{\text{under D50}} &= Z_{\text{under D50 mediawhite}} / Z_{D50} * Z_{\text{mediawhite-relative}} \end{aligned} \qquad \text{D.2.6.1.0-7}$$

The following symbols are used above. The prefix XYZ identify tristimulus values in the form of a 3 rows by 1 column vector..

Table 87 — Relative and absolute rendering intent equation symbols

ChromaticAdaptationMatrix	a 3 by 3 matrix for chromatic adaptation between two illumination sources
X, Y, Z, XYZ_{D50}	relative CIE XYZ tristimulus values for the PCS illumination source, CIE D50. X=0,9642, Y= 1, Z = 0,8249
X, Y, Z, $XYZ_{illuminant}$	relative CIE XYZ tristimulus values for the perfect reflecting diffuser under the actual illumination source. Y = 1
X, Y, Z, $XYZ_{illuminant\text{-}relative}$	relative CIE XYZ tristimulus values for a color patch on the media under the actual illumination source, flare-free. Y = 1 for the perfect reflecting diffuser
X, Y, Z, $XYZ_{illuminant\text{-}relative\ mediawhite}$	relative CIE XYZ tristimulus values for the media white point under the media white actual illumination source, flare-free. Y = 1 for the perfect reflecting diffuser
X, Y, Z, $XYZ_{under\ D50\ mediawhite}$	relative CIE XYZ tristimulus values for the media white point under the PCS illumination source
X, Y, Z, $XYZ_{mediawhite\text{-}relative}$	PCS-side XYZ tristimulus values of a media-relative colorimetric transform
X, Y, Z, $XYZ_{under\ D50}$	PCS-side XYZ tristimulus values for ICC-absolute colorimetry

D.2.6.2 Procedural Summary

The various colorimetric adjustments discussed above can be organized into a computational procedure for calculating PCS coordinates for device-profile transforms. The procedure presented here is applicable to reflection media input and output profiles; monitor transforms are typically computed in a simplified manner, although it is certainly possible to treat monitors in the same way as other input and output devices in order to achieve more accurate image display.

The procedure is given in the device-to-PCS direction for the media-relative colorimetric rendering intent (AToB1Tag) transform.

1. Obtain CIE 1931 XYZ tristimulus values (XYZ measured) for a set of color patches on the device or media to be profiled. More information about measurement procedures is provided in section D.2.3. There should be at least one measurement of the "media white" and one measurement of the illumination source or perfect reflecting diffuser.

2. Remove flare from the measured XYZ values as needed to match the PCS measurement conditions, creating flare-free XYZ values (XYZ flare-free)

3. If necessary, scale the flare-free measurement values so they are relative to the actual illumination source by dividing all values by the measured Y value of the perfect diffuser. After scaling Y = 1 for the perfect diffuser.

X illuminant-relative = X flare-free / Y flare-free for perfect diffuser
Y illuminant-relative = Y flare-free / Y flare-free for perfect diffuser
Z illuminant-relative = Z flare-free / Y flare-free for perfect diffuser D.2.6.2.0-1

4. If the chromaticity of the illumination source is different from that of D50, convert the illuminant-relative XYZ values from the illumination source white point chromaticity to the PCS white point chromaticity using an appropriate chromatic adaptation transform and equation D.2.6.1.0-1. This may be done by applying one of the transformations mentioned in D.2.4. The transform used should be specified in the chromaticAdaptationTag.

$$XYZ_{\text{under D50}} = \text{ChromaticAdaptationMatrix} * \text{XYZ illuminant-relative} \qquad \text{D.2.6.2.0-2}$$

5. Record the converted media white point in the mediaWhitePointTag. Optionally, record the converted black point in the mediaBlackPointTag.

6. Convert colorimetry from D50 illuminant-relative to mediawhite-relative values, by scaling each value by the ratio of the PCS D50 illumination source over the converted media white point, using equation D.2.6.1.0-3. After scaling, the XYZ values for the media white point measurement will be equal to the XYZ values of the PCS D50 illumination source.

$$\begin{aligned} &X \text{ mediawhite-relative} = X_{\text{under D50}} * X \text{ D50 illuminant} / X_{\text{under D50}} \text{ mediawhite} \\ &Y \text{ mediawhite-relative} = Y_{\text{under D50}} * Y \text{ D50 illuminant} / Y_{\text{under D50}} \text{ mediawhite} \\ &Z \text{ mediawhite-relative} = Z_{\text{under D50}} * Z \text{ D50 illuminant} / Z_{\text{under D50}} \text{ mediawhite} \end{aligned} \qquad \text{D.2.6.2.0-3}$$

7. Optionally, convert the adjusted PCS XYZ coordinates to PCS L*a*b* as described in Annex A.

8. Encode the PCS XYZ coordinates or the PCS L*a*b* coordinates digitally in 8-bit or 16-bit representations, as defined in Annex A.

These values can now be used to populate the AToB1Tag.

D.2.6.3 Example

This example shows how the standard data for SWOP, as published in CGATS TR001, could be used when building a device to PCS transform for the media-relative colorimetric intent. The TR001 data can be used as the measurement data needed for step one in D.2.6.2. The actual viewing environment is a graphic arts viewing booth with an illuminance level of 2000 lux. The example shows how white and black would be converted into PCS values for a transform implementing the media-relative colorimetric rendering intent of a profile.

1. The white (no colorant, Patch 26 of IT8.7/3) and black (100% of all colorants, Patch 24 of IT8.7/3) patches have zero flare CIE XYZ values of

Table 88 — Zero flare CIE XYZ values

	white	black
X	0,7067	0,0097
Y	0,7346	0,0101
Z	0,5703	0,0080

2. These measurements do not need to be corrected for flare. The white and black values are unchanged.

3. These values are already relative to the illumination source, so they do not need to be scaled. The white and black values are unchanged.

4. This illumination source is D50, so no chromatic adaptation is needed. The white and black values are unchanged.

5. Record the white and black values in the media white and black point tags.

6. The CIE XYZ values are mapped to PCS by multiplying them by the ratio of the PCS white point to the actual media white point under D50 illumination source:

Table 89 — CIE XYZ to PCS multipliers

	ratio	white	black
X	0.9642 / 0.7067	0,9642	0,0134
Y	1 / 0.7364	1,0000	0,0138
Z	0.8249 / 0.5703	0,8249	0,0116

7. Convert PCS XYZ to PCS L*a*b*:

Table 90 — PCS XYZ to PCS L*a*b* conversion

	white	black
L*	100	11,8
a*	0	0,28
b*	0	-0,3

8. Convert PCS XYZ and PCS L*a*b* to PCS encodings:

Table 91 — PCS XYZ and PCS L*a*b* to PCS conversion

16-bit	white	black	16-bit	white	black	8-bit	white	black
X	31595	439	**L***	65535	7733	**L***	255	30
Y	32768	452	**a***	32896	32968	**a***	128	128
Z	27030	380	**b***	32896	32819	**b***	128	128

Note that the 8-bit L*a*b* encoding of black is imprecise because of the limited precision afforded by 8-bit data.

D.2.7 A Discussion of Perceptual Rendering Intent

D.2.7.1 Colorimetry and Appearance

One possible definition for the PCS is that it specifies the colorimetry of an image reproduction. Colorimetry, as established by the CIE, is a system of measurement and quantification of visual color stimuli. As such, it is independent of any particular device, medium, or process. This makes it a suitable candidate for

a common interface. With this choice, the output reproduction of an image would present the same color stimuli to an observer as the input, even if it employs a different process of color reproduction. This seems to guarantee the same colors on all media, which would make it the right definition for the PCS for the purposes of color management.

Unfortunately, this simple definition is inadequate for appearance matching as requested by the perceptual intent. The appearance of a color depends not only on the color stimulus presented to the retina, but also on the state of visual adaptation of the observer. In certain cases, different media require different visual color stimuli because they will be viewed in different environments. For example, differences in surround condition or illumination source chromaticity will cause the observer to experience different visual adaptation effects. In order to preserve the same color appearance in these environments, the colorimetry must be corrected to compensate for the adaptation of the human visual system and for physical differences in the viewing environments. By extension, these effects occur in images where the immediate surround of any color in an image consists of other colors in the image. If the relation of any of the colors in the image is changed, for example because of gamut limitations, the color stimuli required to reproduce the image may change, even though the viewing environment for the whole image does not change. **Color appearance is still an active research topic.** Color appearance models for single stimuli function well, and are applicable to images when gamut limitations do not come into play. The science in support of generalized image appearance modeling is less well developed.

There are also aesthetic reasons why it may be necessary or desirable to alter the colorimetry for specific media. For instance, hard-copy media - even those intended for the same viewing environment - differ considerably in their dynamic range and color gamut. A well-crafted rendering of an image on a specific medium will take advantage of the capabilities of that medium without creating objectionable artifacts imposed by its limitations. For instance, the tone reproduction of the image should attempt to provide sufficient contrast in the midtones without producing blocked-up shadows or washed-out highlights. The detailed shape of the tone curve will depend on the brightest and darkest tones (the maximum and minimum reflectances) attainable in the medium. Clearly, there is considerable art involved in shaping the tone-reproduction and color-reproduction characteristics of different media and much of this art is based on subjective, aesthetic judgments. As a result, the substrate and the colorants used in a medium will be exploited to impart a particular personality to the reproduction that is characteristic of the medium. In reproducing an image on various types of media, it may be desirable to adjust the colorimetry to accommodate the differing characteristics of those media. In any case, it is necessary to accommodate the gamut differences. Such considerations go beyond the simplistic matching of color stimuli or even of color appearance.

These adjustments need to be incorporated in the color transforms of the device profiles. Since the PCS is the common interface of these profiles, it has to be defined in a way that facilitates these adjustments. Thus, although the definition of the PCS may be based on the principles of colorimetry, it must also take into account various issues that lie outside the realm of colorimetry and that involve adaptive corrections, pragmatic considerations, and aesthetic judgments.

D.2.7.2 Purpose and Intent of the PCS

These considerations led to a fundamental statement that the PCS for perceptual rendering intent represents desired appearance. The term "desired" implies that the PCS is oriented towards colors to be produced on an output medium. Obviously, "desired" is open to various interpretations, but in order to enable the decoupling of input and output transforms, it must be interpreted in a way that, to the extent possible, transcends the capabilities and limitations of the specific color-reproduction processes, devices, and media for which profiles are to be provided.

For instance, an input profile for a slide scanner should attempt to yield "desired" colors, represented in the PCS, that are independent of the gamut and aesthetics of any specific output medium. This independence, which decouples the PCS colors from the device colors, allows the input profile to be used in con-

junction with any output profile. These desired colors will be based on the colors of the input slide but are not necessarily identical to those colors or limited to the gamut of the slide medium. They are the colors that would be desired on output if the characteristics of the potential output media could be transcended.

Similarly, the output profile for a color printer must reproduce the desired colors within the capabilities and limitations of the output medium and device. This reproduction may involve some adjustment of the colors, but it transcends the characteristics of any specific input medium and permits the use of the output profile in conjunction with a variety of different input profiles.

With this PCS definition, it is the responsibility of the profile transforms to handle any required corrections or modifications to the colorimetry of a reproduction. Input profiles are responsible for modifying the colorimetry of the input media to account for adaptation, flare, and gamut limitations. They also must provide the artistic intent implicit in the word "desired", which allows latitude for variation. For instance, the "desired" colors may be a close facsimile of the original, an aesthetic re-rendering of the original, or a simulation of a specific reproduction medium different from both the input and output media.

Output profiles for media that are viewed in environments different from the reference are responsible for modifying the colorimetry to account for the differences in the observer's state of adaptation as well as any substantial differences in viewing flare present in these environments. This is needed in order to preserve color appearance. Profiles must also incorporate adjustments to the dynamic range and color gamut of the image in order to accommodate the limitations of the actual medium.

D.2.7.3 Reference Medium and Reference Viewing Environment

While a profile is needed which represents desired appearance and transcends the actual device, it is difficult to know how to generate such a profile. It is helpful here to conceptualize a "reference medium" which is a hypothetical medium on which the colors are being rendered (see section D.1.5). It has a large gamut and dynamic range which approximate the limits of current reflection-print technology. It is described using "realworld" specifications so that even though the medium is not real, it can be treated as if it were real.

It is also necessary to define a "reference viewing environment" which is the environment in which the reference medium is being viewed (see section D.1.4). This environment is used to determine the observer's adaptation state and establishes the connection between color stimulus and color appearance.

The concept of a reference medium viewed in the reference viewing environment helps the profile designer to understand how to produce "desired appearance" in the PCS. At the same time, it preserves the goal of decoupling the characteristics of actual media through a virtual intermediate reproduction description. Where the real viewing environment differs from that of the reference environment chromatic adaptation may be an important component in the set of adaptation transforms that are applied to obtain conformance with the reference viewing environment.

NOTE For the perceptual intent, the colorimetry represented in the PCS is that of the image as optimally color rendered to the perceptual intent reference medium and viewing conditions. If the illumination source used to view the actual image has a chromaticity different from that of D50, this color rendering will typically include some type of chromatic adaptation. However, the color rendering used to produce the reference medium image colorimetry will also consider other factors, such as dynamic range and gamut mapping, adaptation for other differences between the reference and actual viewing conditions, and preferential color adjustments. For this reason, it may not make sense to invert the chromatic adaptation as specified in the chromaticAdaptationTag, because the result will be the reference medium colorimetry transformed to be relative to the actual illumination source, which may not produce the colorimetry of the actual image. There is no guarantee the colorimetry produced by the inverse of the chromaticAdaptationTag will be optimal for the reference medium under the actual illumination source, since the color rendering to the reference medium could include optimizations based on the D50 reference white.

D.2.7.4 Aesthetic Considerations and the Media White Point

As discussed in section D.2.5, for the perceptual intent the white point of the actual medium can be mapped to the white point of the reference medium. On the other hand, based on aesthetic considerations, the white point of the actual medium can be mapped to a color other than the white point of the reference medium.

In either case, the white point of the reference medium will correspond, after scaling, to the PCS white point (see section D.1.3 and step 5 of section D.2.7.7). This is another means by which unique value may be added to profiles while maintaining data interoperability.

D.2.7.5 Brightness Adaptation and Tone-scale Correction

One of the most fundamental corrections that must be applied to the measured colorimetry has to do with issues of tone reproduction and overall brightness level. These issues involve adaptive effects, as well as aesthetic and pragmatic considerations.

When viewing a reflection print under normal viewing conditions (i.e., where the print and the area surrounding the print are similarly illuminated), the observer becomes adapted to things perceived as white in the environment. A reflection print is perceived as an object in this environment. Now, the brightest areas in the image are those in which the paper (or other substrate) is blank (no colorant). Since the reflectance of any actual paper is limited (typically 85% to 90%), the medium viewed in this environment cannot realistically create the appearance of specular highlights or other very bright objects that may have existed in the original scene, which can be several times brighter than 100% diffuse white, let alone the paper substrate. Thus, the highlights must be considerably compressed in the reproduction.

On the other hand, slides or movies projected in a darkened room do not suffer from the same limitation. In the absence of dominant external references, the observer's state of adaptation is controlled by the bright image on the screen. Thus, these media are designed to reproduce diffuse white at a lower luminance than the maximum attainable, which leaves some headroom for the reproduction of specular highlights and other very bright tones. To the adapted observer, these tones actually have the appearance of being brighter than 100% diffuse white; they sparkle and shine with a more realistic intensity than is possible for a print viewed under normal conditions. Thus, their representation in the PCS would require an apparent luminance greater than that of the white reference ($Y > 1$, or $L^* > 100$). The same illusion is possible with back-lit transparencies and video, as long as the viewing environment is sufficiently dim that the observer is adapted primarily to the image, rather than the surround.

Of course, there are limits to the apparent brightness that can be simulated by these media, but they are far higher than those of reflection prints in a normal surround - perhaps 200%, as compared with 90%, relative to diffuse white. The practical consequence of this difference is that the tonal compression of highlights is much less severe in the case of movies, slides, and video, than in the case of typical prints on paper.

All real media have a limit at the dark end of the tone scale, so that tonal compression is required in the shadows as well. Furthermore, the level of flare in the intended viewing environment has a strong effect on the apparent tone scale, particularly in the shadows and three-quarter tones; media designed for viewing conditions with different levels of flare tend to incorporate different amounts of flare compensation in their tone reproduction.

PCS colorimetry must also be corrected to account for the change in color appearance caused by differences in the absolute luminance level. For example, the 500 lux illuminance of the reference viewing environment is specified to be typical of actual home and office viewing environments. Corrections will typically be needed to correct for the darker, less colorful appearance of reproductions when they are

viewed at lower levels of illumination, or the lighter, more colorful appearance when they are viewed at higher levels of illumination.

In photographic systems, the tone-reproduction characteristics are implemented in the construction of the sensitized layers and the chemistry of the emulsions and developers, or in the case of digital photography, in the image processing. In video, they are implemented in the electronics of the camera and receiver. Thus, a color management system usually deals with an image originating from a medium or device that has already imposed its own tone characteristic on the luminances captured from a scene, so that the highlights and shadows are already compressed. However, it is often necessary to reproduce the image on a different medium, for which the original compression may be less than ideal. In such cases, for best results, the tone scale of the image should be adjusted for the output medium.

D.2.7.6 The Reference Medium and Tonal Compression

The PCS and its reference medium provide a convenient interface for the tone-scale adjustments just discussed. Input transforms apply adjustments to map the tone scale of the original medium onto that of the reference; output transforms incorporate adjustments to map the tone scale of the reference medium onto that of the output medium.

These adjustments can take on many different forms, depending on the aesthetic effect to be achieved. In some cases, the appearance of the original may be accurately preserved; in others, it may be preferable to make deliberate alterations in the appearance, in order to optimize the rendering for the output medium or to simulate a third medium. This range of possibilities is implicit in the phrase "desired color appearance" in the PCS definition for the perceptual intent.

Output to media with a dynamic range different from that of the reference medium may be handled by tone-shaping techniques which compress or expand the tone scale to the range the device can handle. Furthermore, in output profiles, the different "rendering intents" can incorporate different adjustments. Some perceptual transforms, for example, can be designed to preserve the tone scale of the reference medium, clipping abruptly at the minimum reflectance if necessary, while other perceptual transforms may apply a more subtle reshaping of the highlight and shadow tones.

Input from media with a dynamic range different from the reference medium also may have tone-shaping techniques applied, along with luminance scaling to maintain brightness balance. These adjustments should be invertible (in the sense that they match the precision of the data and the computation) for high-quality output to the same devices. For instance, images with an extended highlight range (such as those from scanned photographic transparencies) must be remapped for the reference medium, so that the highlights will be compressed to the range of the PCS.

The details of these techniques may vary with the intended market, the specified "rendering intent", and aesthetic choices made by the profile builder. If the intent is to preserve the appearance of the original, adjustments to the tone scale can be limited to those compensating for differences between the actual viewing conditions and those of the reference environment. These include the effects of brightness adaptation, surround adaptation, and viewing flare. In other cases, there is plenty of latitude for profile vendors to differentiate their products with respect to aesthetic choices, while still basing their profile transforms on the common definition of the PCS. Thus, proprietary art can be fostered and encouraged in a context of interoperability.

D.2.7.7 Procedural Summary

The various colorimetric adjustments discussed above can be organized into a computational procedure for calculating PCS coordinates for device-profile transforms. The procedure presented here is applicable to reflection media input and output profiles; monitor transforms are typically computed in the simplified

manner described below, although it is certainly possible to treat monitors in the same way as other input and output devices in order to achieve more accurate image display.

The procedure is given in the device-to-PCS direction for the perceptual rendering intent (AToB0Tag) transform. This procedure is intended as a conceptual guide, recognizing that artistic preferences may result in significant variations on this procedure.

1. Obtain CIE 1931 XYZ tristimulus values for a set of color patches on the device or media to be profiled. More information about measurement procedures is provided in section D.2.3. There should be at least one measurement of the "media white." Additionally, it is necessary to obtain the colorimetry of the adaptive white point.[1)] Apply steps 2 and 3 from section D.2.6.2.

2. If the chromaticity of the adaptive white point is different from that of D50, convert colorimetry from the device adaptive white point chromaticity to the PCS white point chromaticity using an appropriate chromatic adaptation transform. This may be done by applying one of the transformations mentioned in D.2.4.

3. Other corrections must be applied to the data to account for any differences in viewing conditions between the actual environment and the reference environment. These include, but are not limited to, tonal adjustments for differences in viewing flare, general brightness adaptation, and surround effects.[2)]

4. Convert the corrected colorimetry to "desired" colors for the reference medium. The medium white and black points are mapped to the reference medium white and black points. In general, there is considerable freedom in this step, depending on aesthetic considerations. Optimal renderings will frequently require adjustments to the tone scale and color reproduction, especially when there is a significant difference in dynamic range or color gamut between the actual medium and the reference medium.

5. Scale the reference-medium CIE XYZ coordinates to PCS values, so that the reference medium white point maps to the PCS white point. This scaling is conceptually equivalent to transforming the "desired reference medium image" to media-relative PCS using a media-relative colorimetric intent mapping and equations D.2.6.1.0-4 through D.2.6.1.0-6 in section D.2.6.1.

6. Optionally, convert the PCS XYZ coordinates to PCS L*a*b* as described in Annex A.

7. Encode the PCS XYZ coordinates or the PCS L*a*b* coordinates digitally in 8-bit or 16-bit representations, as defined in Annex A.

D.2.8 Monitor Display

Some special considerations apply to monitor profiles. Since a CRT monitor is a self-luminous display, the interpretation of tone is somewhat ambiguous: Should full-drive monitor white be regarded as 100% diffuse white? In terms of color appearance, the answer to that question depends on the state of the observer's adaptation, which is influenced by the viewing environment. For example, in a brightly-lit office environment, the observer may adapt to the ambient illumination. In a dim environment, the observer may adapt to the monitor screen itself. In general, it is very difficult to predict the observer's actual state of adaptation.

However, for desktop applications the document editor or graphic artist typically has an expectation that monitor white will be associated with the blank paper (or other substrate) of the output medium, regardless of his or her actual state of adaptation. Thus, for practical reasons, it is important that the monitor profiles

---

1) E. J. Giorgianni and T. E. Madden, op. cit., p. 356.

2) E. J. Giorgianni and T. E. Madden, op. cit., p. 474.

be designed to display paper white at full-drive monitor white (R = G = B = 255 on a typical 24-bit display). Similarly, there is an expectation that R = G = B = 0 corresponds to "black" and will be reproduced by the minimum reflectance of the output medium. These user expectations are based on common practice and convenience and lie outside of strict colorimetry and color-appearance considerations.

Furthermore, the monitor profile transforms that are common on many systems are based on oversimplified mathematical models. Often they take the form of a linear transformation from XYZ to RGB (a 3 x 3 matrix) followed by a simple power law in each channel for gamma correction. Such transforms often fail to model the behavior of the monitor accurately in the shadows, since they ignore the biases that commonly occur in the CRT and support electronics. These biases are variable from unit to unit and are also dependent on the user-selectable settings of contrast and brightness. Fortunately, any departures from colorimetric accuracy that result from these simple models are relatively minor and are partially masked by face-plate reflections, often 3 to 5 percent, so that they are generally tolerated.

These simple monitor profiles will satisfy typical user expectations if monitor white is mapped to the XYZ values of the PCS white point and monitor black is mapped to the PCS black point. This means that monitor white maps to reference medium white and monitor black maps to reference medium black. When processed through a typical monitor profile transform, therefore, reference medium white will be displayed at monitor white and reference medium black will be displayed at monitor black. This provides a practical mapping between the monitor and the reference medium while permitting the use of simple monitor transforms to satisfy common user expectations.

D.2.9 Bibliography

1. CIE Publication 15.2-1986, Colorimetry (second edition).

3. ISO 13655:1996, Graphic technology - Spectral measurement and colorimetric computation for graphic arts images.

4. R. W. G. Hunt, The Reproduction of Colour, Fourth Edition, Fountain Press, 1987.

5. E. J. Giorgianni and T. E. Madden, Digital Color Management, Addison-Wesley, Reading, Massachusetts, 1998.

6. Mark D. Fairchild, Color Appearance Models, Addison-Wesley, Reading, Massachusetts, 1998.

Annex E
Chromatic Adaptation Tag

This section describes the derivation and use of the Chromatic Adaptation Tag in more detail. The first part recommends a chromatic adaptation transform (CAT) for general use. The second part provides a mathematical description of this recommended CAT. The last part provides basic guidelines and instructions for possible use of the Chromatic Adaptation Tag.

The Chromatic Adaptation Tag is required when the actual illumination source has a chromaticity different from that of CIE illuminant D50. Profiling applications can omit this tag or store an identity matrix in this tag when the actual illumination source has a chromaticity identical to D50.

E.1 Calculating the Chromatic Adaptation Matrix

The ICC profile format specification allows the use of different linear (matrix-based) CATs. This flexibility allows profile creators to select the most appropriate CAT for their applications. Criteria for selection include visual performance, the gamut of the image as transformed to the D50 PCS, and other considerations. However, the use of different CATs will produce different results, which may be undesirable. Therefore, it is recommended that the linear Bradford CAT (which is the same as the linearized CIECAM97s transformation) be used when there is no reason to use a different CAT. The linear Bradford CAT has been widely implemented in the digital imaging industry, with demonstrated excellent visual performance. If a profile creator decides to use a CAT other than linear Bradford, they should do so only to address specific known issues, recognizing that the resulting profile will most likely produce different results than profiles from other sources.

E.2 Linearized Bradford/CIECAM97s Transformation

When full adaptation is assumed and a negligible non-linearity in the blue channel is omitted, the Bradford transformation is identical to the CIECAM97s transformation. Under the above assumption both become the same variant of a cone-space transform. Similarly as in von Kries method, the cone response values can be found through the matrix equation:

$$\begin{bmatrix} \rho \\ \gamma \\ \beta \end{bmatrix} = \begin{bmatrix} 0,8951 & 0,2664 & -(0,1614) \\ -(0,7502) & 1,7135 & 0,0367 \\ 0,0389 & -(0,0685) & 1,0296 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M_{BFD} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad \text{E.2.0.0.0-1}$$

The calculation of corresponding (visually equivalent) CIE XYZ values between two white points is the same as in von Kries transformation and similarly, the following chromatic adaptation matrix can be derived:

$$M_{adapt} = M_{BFD}^{-1} \begin{bmatrix} \rho_{pcs}/\rho_{src} & 0 & 0 \\ 0 & \Upsilon_{pcs}/\Upsilon_{src} & 0 \\ 0 & 0 & \beta_{pcs}/\beta_{src} \end{bmatrix} M_{BFD} \qquad \text{E.2.0.0.0-2}$$

Where:

$$\begin{bmatrix} \rho_{src} \\ \gamma_{src} \\ \beta_{src} \end{bmatrix} = M_{BFD} \begin{bmatrix} X_{WPsrc} \\ Y_{WPsrc} \\ Z_{WPsrc} \end{bmatrix} \qquad \text{E.2.0.0.0-3}$$

$$\begin{bmatrix} \rho_{pcs} \\ \gamma_{pcs} \\ \beta_{pcs} \end{bmatrix} = M_{BFD} \begin{bmatrix} X_{WPpcs} \\ Y_{WPpcs} \\ Z_{WPpcs} \end{bmatrix} \qquad \text{E.2.0.0.0-4}$$

XY_{WPpcs} represents the illuminant of the reference viewing condition.

E.2.1 Applying the Chromatic Adaptation Matrix

The application of the chromaticAdaptationTag is under active study by the ICC. The chromaticAdaptationTag may not apply to the perceptual intent (see D.2.7.3). The user may look at the set of profiles to determine what adjustments can be made. There are several possibilities:

1. No profile has the chromaticAdaptationTag. No action can be taken.

2. All profiles have the chromaticAdaptationTag. If the same method is used, no action should be taken. If different methods are used, the user may choose to undo them first before using a consistent method of their choice.

3. Only one profile has chromaticAdaptationTag. Processing is implementation dependent.

Here is a step by step example of how to do the adjustments if the color transformation is created from two RGB Display profiles containing the chromaticAdaptationTag.

Step 1. Determine if the two methods are the same. If the two matrices are identical, the chromatic adaptation methods are the same. If the matrices are different, the methods could still be the same while the actual viewing illuminants are different. One easy way to test this is: if M1 and M2 represent the chromatic adaptation matrices from profile 1 and 2 respectively, it can be proven that chromatic adaptation algorithms are the same if the following matrix equation holds true: M1 * M2 == M2 * M1. We can stop here if two algorithms are the same.

Step 2. Determine the actual device viewing illuminant for profile 1. This can be achieved by applying the inverse chromatic adaptation matrix to the PCS D50 XYZ value.

Step 3. Invert the red, green, and blue values stored in the colorant tags to the actual device illuminant values. This is accomplished by applying the inverse of the chromatic adaptation matrix for each colorant.

Step 4. Calculate the new chromatic adaptation matrix. Follow the examples of E.1. Use your favorite cone response matrix and generate a new matrix.

Step 5. Generate new D50 relative colorant values for red, green, and blue by applying the matrix calculated in step 4 to colorant values in the device illuminant derived in step 3.

Step 6. Repeat steps 2 to 5 for profile 2.

For profiles with LUT tags, the adjustments can be made after the values are converted into the PCS by adding an extra processing step of undoing and redoing the chromatic adaptation.

Annex F
Summary of spec changes

This annex lists the major changes made to the specification for the past few spec revisions. Minor editorial and cosmetic changes are not listed.

These are the changes made from Revision ICC.1:2001-12

1. Addition to allow the use of multidimensional tables for N-component LUT-based display profiles (Section 6.3.2.3).

   Per: Online ballot #200201
2. Change to the copyright notice, per NPES lawyer recommendation.

   Per: Online ballot #200203
3. Change to recommended CAT (Annex E, E.1, E.2, E.2.1)

   Per: Online ballot #200204
4. Clarification of Section 6.1.11, field type definition.

   Per: Online ballot #200204
5. Rationalizing requirement statement in Section 6.5.3 with references to this section in Section 6.3.3.2, Section 6.3.4.1, and Section 6.4.14.

   Per: Editor judgement
6. Clarification of wording for clauses 5.3.10, 6.4.25, 6.5.25, and Annex A regarding Y range and viewingConditionsTag.

   Per: Online ballot #200111
7. Addition of informative note recommending content for tag originally included in ballot but not added into text of spec for clause 6.4.10:“charTargetTag”.

   Per: Online ballot #200019
8. Fixed Table 62 equations for consistency.

   Per: Editor judgement
9. Added informative note to clause 6.5.10:“lutBtoAType” clarifying use of Matrix element, consistent with online ballot 200210.

   Per: Editor judgement
10. Removal of unused tags: crdInfoTag, deviceSettingsTag, ps2CRD0Tag, ps2CRD1Tag, ps2CRD2Tag, ps2CRD3Tag, ps2CSATag, ps2RenderingIntentTag, screeningDescTag, screeningTag, ucrbgTag, and associated types as applicable. Many changes in Annex C as there are now no longer any PostScript Level 2 tags.

    Per: Online ballot #200209
11. Addition of informative text explaining the preview tags preview0Tag, preview1Tag, and preview2Tag.

    Per: Online ballot #200206
12. Changes to allow use of Matrix and M-curves of an mAB-type A2B0 tag in a DeviceLink profile.

    Per: Online ballot #200210
13. Clarification of clause 6.1.13:“Profile ID”.

    Per: Editor judgement
14. Clarification of why Y max is 1 rather than 100 per CIE in Annex A.2.

    Per: Editor judgement
15. Clarification of relationship between Chromatic Adaptation and Perceptual Rendering Intent.

    Per: Online ballot #200303

16. Addition of Annex C material clarifying relationship between ICC profiles and PostScript CSAs and CRDs.
    Per: Editor judgement
17. Updated Adobe PostScript Language Reference to latest Third Edition.
    Per: Editor judgement
18. Clarification of Profile ID section 6.1.13.
    Per: Editor judgement
19. Removal of obsolete BG, GCR, UCR references in section 5.2.
    Per: Editor judgement
20. Replacement of ProfileVersion to 4.1.0 in section 6.1.3.
    Per: Editor judgement
21. Replacement of output profile: TRC/Matrix with "colorimetric" because of grey printer profiles in Table 20.
    Per: Editor judgement
22. Removal of obsolete definition of ViewingCondDescTag in Table 33.
    Per: Editor judgement
23. Removal of obsolete text in Annex C.
    Per: Editor judgement
24. Removal of "7-bit ASCII" normative statement from copyrightTag description in Table 21 through Table 33.
    Per: Editor judgement
25. Change of Zr/Zn to Zr/Zi in Annex A.2.
    Per: Editor judgement

These are the changes made from Revision ICC.1:2001-04

1. Addition of the colorantOrderTag (Section 6.4.13) , colorantTableTag (Section 6.4.14), colorantOrderType (Section 6.5.2), and colorantTableType (Section 6.5.3).
   Per: Online ballot #200004
2. Replace Annex D, Annex D, with new annex.
   Per: Online ballot #200018
3. Add new lut types, lutAtoBType (Section 6.5.9) and lutBtoAType (Section 6.5.10).
   Per: Online ballot #200005
4. Clarification of colorant tags including changing the name from "Colorant" to "MatrixColumn". Affects 6.3.1.2, 6.3.2.2, 6.3.2.3, 6.4.4, 6.4.20, 6.4.33, and 6.3.
   Per: Online ballot #199907
5. Clarify the PCS CIELAB equations (see A.1).
   Per Online ballot #200011
6. Add new multiLocalizedUnicodeType and apply it to description tags. See 6.5.12, 6.4.15, 6.4.16, 6.4.17, 6.4.31, 6.4.43, 6.4.36, and 6.5.15.
   Per: Online ballot #200006
7. Add new Profile ID field to header.
   Per: Online ballot # 200008
8. Change the definition of the gray profile. See 6.4.19, 6.3.1.1, 6.3.2.1, and 6.3.3.1.
   Per: Online ballot #2000014

9. Addition to Annex A on how to handle PCS encoding bounds.
   Per: Online ballot #2000012
10. Add wording to allow handling of data/time values to be consistent. Clause 5.3.1.
    Per: Online ballot # 200009
11. Addition of parametricCurveType (Section 6.5.14).
    Per: Online ballot #199909
12. Addition of new parametric function (type 4) to parametricCurveType (Section 6.5.14).
    Per: Online ballot #200108
13. Make chromaticAdaptationTag a required tag for all profile types except the DeviceLink and named profile types. This tag was added but not required in the last specification update.
    Per: Online ballot # 2000010
14. Modify annex equation numbering to reflect annex number.
    Per: Editor judgement
15. Clarification of illumination level definition for PCS viewing conditions (A.1)
    Per. Online ballot #200101
16. Extends use of the charTargetTag to include ability to use ICC Characterization Data Registry info. (Section 6.4.10)
    Per. Online ballot #200102
17. Define new PCS Lab value range. (Section 6.5.7, Section 6.5.13, Annex A, and Annex D)
    Per. Online ballot # 200103
18. Include fallback strategy for the use of transform tags. (Section 0.8)
    Per. Online ballot # 200105
19. Clarification of interpolation for degenerate LUTs. (Section 6.5.4, Section 6.5.9, Section 6.5.10, Section 6.5.7, and Section 6.5.8)
    Per. Online ballot #200107
20. Removal of member list from specification. Web site can maintain better accuracy.
    Per. Spec Editing WG
21. Convert numbers to ISO-compliant formats (decimal points become commas and commas become spaces)
    Per. ISO number representation definitions
22. Add appendix-specific numbers to equations in appendices.
    Per. ISO requirements
23. Delete namedColorTag, namedColorType, and textDescriptionType from specification.
    Per: ICC vote
24. Clarification of wording for Section 5.3.10, Section 6.4.25, Section 6.5.26, and Annex A.
    Per. Online ballot #200111
25. Correct LAB conversion equations in A.2.
    Per. Spec Editing WG

These are the changes made from Revision ICC.1:1998-09

1. The descriptions of the various rendering intents have been clarified. (See 4.11 and all of A.4.)
   Per: Online ballot #200015
2. Two new attribute bits have been added to the profile header. (See Table 17.)
   Per: Online ballot #199805
3. The interpretation of multidimensional tags is now defined in more situations. (See Table 20 and all of Section 6.3)
   Per: Online ballot #200016

4. Multidimensional tags are now allowed in monochrome device profiles. (See 6.3.1.1, 6.3.2.1, and 6.3.3.1.)
   Per: Online ballot #200016
5. A new optional tag, chromaticAdaptationTag, has been added. (See 6.4.11 and Annex E.)
   Per: Online ballot #200010
6. A new optional tag, chromaticityTag, has been added. (See 6.4.12 and 6.5.1.)
   Per: Online ballot #199908
7. The description of gamutTag has been reworded to make clear that its input is PCS values. (See 6.4.18.)
   Per: Spec Editing WG
8. The descriptions of lut8Type and lut16Type have been expanded to explain how tables map to one another. (See 6.5.7, 6.5.8, and A.3.)
   Per: Online ballot #199806
9. The offset of the name prefix in namedColorType has been corrected to read "16..t".
   Per: Spec Editing WG
10. A rule was added to prevent the modification of certain text inside textDescriptionType strings.
11. The illumination level of the reference viewing condition is defined as 500 lux (previously it was undefined). (See A.1)
    Per: Online Ballot #200101
12. The term "relative colorimetry" has been changed to "media-relative colorimetry" and the term "absolute colorimetry" has been changed to "ICC-absolute colorimetry" to avoid confusion with CIE terminology. (See A.4.1.2 and A.4.1.3.)
    Per: Online ballot #200015
13. The C header file example has been removed from the specification. It can be found online at the ICC Web site.
    Per: Spec Editing WG
14. The formatting has been altered to more closely match the format of ISO and IEC standards. (NOTE This specification is not an International Standard, and it does not meet all of the ISO/IEC drafting rules.)
    Per: Spec Editing WG

These are the changes made from Version 3.4 (August 1997):

1. The Normative References have been brought up to date and expanded to include all cited International Standards. (See clause 2.)
   Per: Spec Editing WG
2. The CMM Type and Primary Platform signatures are now allowed to be set to zero. (See 6.1.2 and 6.1.7.)
   Per: Resolution voted 1998-03-15
3. The profile version number in the header has been changed to 2.2.0. (See 6.1.3.)
   Per: Resolution voted 1998-07-24
4. The terms “low *n* bits” and “first *n* bits” have been changed to “least-significant *n* bits” to avoid confusion. (See 6.1.8, 6.1.10, 6.1.11, 6.2.2, and 6.5.)
   Per: Spec Editing WG
5. A tag can now only appear once in a profile. (See 6.2.)
   Per: Resolution voted 1998-03-15
6. The table describing the interpretation of context-dependent tags has been expanded to explicitly list the contexts where the interpretation is undefined. (See Table 20.)
   Per: Resolution voted 1998-07-24

7. The rules concerning which classes of profiles can and cannot be embedded in images have been made consistent. (See 6.3.4.2, 6.3.4.3 and B.)
   Per: Resolution voted 1998-03-15
8. A new optional tag, deviceSettingsTag, has been added. (See 6.4.19 and 6.5.8)
   Per: Online ballot #199706
9. A new optional tag, outputResponseTag, has been added. (See 6.4.27 and 6.5.16.) The new basic numeric type response16Number has been added to support the responseCurveSet16Type. (See 5.3.2.)
   Per: Online ballot #199801
10. The possibility for alignment problems in crdInfoType has been pointed out. (See 6.5.4.)
    Per: Spec Editing WG
11. All curveType tags now must follow the same interpretation rules for zero-entry and one-entry tables. (See 6.5.4.)
    Per: Resolution voted 1998-03-15
12. The method for embedding profiles in GIF images has been added. (See B.5.)
    Per: Online ballot #199704
13. The C header file has been updated to reflect the addition of new tags.
    Per: Spec Editing WG

These are the changes made from Version 3.3 (November 1996):

1. The Definitions clause has been updated. (See clause 4.)
   Per: Spec Editing WG
2. A new clause has been added for symbols and abbreviations used in the Specification. Abbreviations that were previously under Definitions have been moved to the new clause. (See 5.2.)
   Per: Spec Editing WG
3. The dataType and textType descriptions now spell out how the data size is calculated. (See 6.5.5 and 6.5.19.)
   Per: Spec Editing WG
4. The method for embedding profiles in JFIF images has been added. (See B.4.)
   Per: Online ballot #199701
5. The C header file has been updated. The conditional compilation has been altered to provide a default definition of the data types in all circumstances. The typedef for `icCrdInfoType` has been altered, and comments have been added to explain its use.
   Per: Spec Editing WG

These are the changes made from Version 3.2 (November 1995):

1. The requirements for Input Profiles have changed. Instead of having categories for RGB and CMYK input devices, the requirements have been changed to cover “three-component matrix-based” profiles and “N-component LUT-based” profiles. (See 6.3.1.2 and 6.3.1.3.)
   Per: Online ballot with results reported on 1996-07-15
2. The list of color space signatures has expanded to include generic color spaces (2colorData to 15colorData). (See Table 13.)
   Per: Resolution voted 1996-07-15
3. The profile version number in the header has been changed to 2.1.0. (See 6.1.3.)
   Per: Included in "NCLR" (generic color space) resolution
4. A new optional tag, crdInfoTag, has been added. (See 6.4.16, 6.5.4, and C.1.)
   Per: Resolution voted 1996-03-28

5. The signature of namedColor2Type was incorrectly listed as ‘ncol’ in Version 3.2 of the spec. The correct signature is ‘ncl2’. (See Table 58.)
   Per: Spec Editing WG
6. The description of the PCS encodings has been rewritten to clarify some issues. The actual encodings have not changed. (See A.1.)
   Per: Resolution voted 1996-07-15
7. The possibility for alignment problems in textDescriptionType and ucrbgType has been pointed out. (See 6.5.24.)
   Per: Resolution voted 1996-07-15
8. The examples for embedding profiles in EPS files have been corrected to show the required colon (:) after `%%BeginSetColorSpace` and `%%BeginRenderingIntent`. (See B.2.)
   Per: Spec Editing WG

Media Formats

- Sound
  - WAV (uncompressed, PCM)
  - MIDI
- Image
  - BMP
  - XML dia
- Video
  - MPEG

Sources

- WAV Sound
  - Chapter 4, main text
  - Kientzle, T. (1998). A Programmer's Guide to Sound. Addison-Wesley.

WAV file format

- A file header
- Binary samples
- Variable file format
  - Compression is possible
  - Uncompressed format also possible (“PCM”)
- In the following, we will discuss one type of WAV file format
  - If you create a file in this format, it will be interpreted as a WAV file
  - There are other formats (e.g., different headers, compressed data) that are also interpreted as WAV files

WAV file header (PCM)

```
/* My assumptions about data type lengths */
#define     int8        char
#define     int16       short
#define     uint16      unsigned short
#define     int32       int

/* WAV file header;
Mostly from Kientzle (1998). */
```

```
typedef struct {
    int8 ChunkType1[4];    /* 'R', 'I', 'F', 'F' */
    int32 length;          /* int bytes, length of everything after this in file */
    int8 ContainerType[4];          /* 'W', 'A', 'V', 'E' */
    int8 ChunkType2[4];    /* 'f', 'm', 't', ' ' */
    int32 FormatChunkDataLength; /* 16 */
    FormatChunkData fcd; /* 16 bytes long */
    int8 ChunkType3[4];    /* 'd', 'a', 't', 'a' */
    int32 SoundLength;              /* in bytes */
    /* After this, SoundLength bytes of data in file */
} WavHeader;
```

```
typedef struct {
    int16 CompressionCode; /* 1 for PCM */
    int16 NumberOfChannels; /* 1 for mono */
    int32 SamplesPerSecond;
    int32 AvgNumberBytesPerSecond;
    int16 BlockAlignment;
    int16 SignificantBitsPerSample; /* 8 or 16 */
    /* 16 bytes total */
} FormatChunkData;
```

dog.wav from Chapter 3

- http://www.cprince.com/courses/cs3121fall02/lectures/media/dog.wav
- File size: 29764 bytes

FormatChunkDataLength= 16

CompressionCode= 1

NumberOfChannels= 1

SamplesPerSecond= 22050

AvgNumberBytesPerSecond= 22050

BlockAlignment= 1

SignificantBitsPerSample= 8

sizeof header= 44

sizeof fcd= 16

SoundLength= 29720

length= 29756

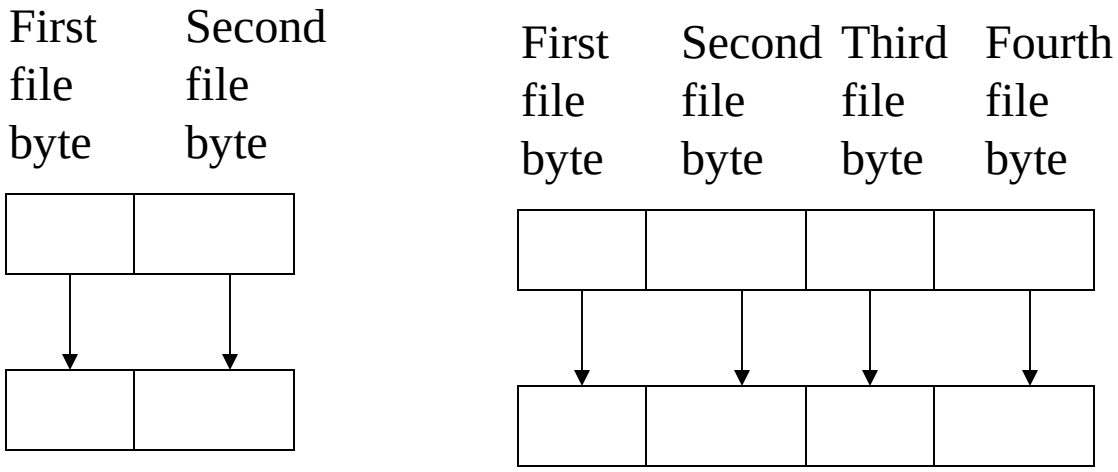
Text vs. Binary Files

- A text file is usually one that can be displayed in a word processor, or text editor
  - The file example.txt is a text file
    - (No particular reason the .txt is needed!)
  - .cpp, .c, .h program code source files are text files
- Binary files
  - Are usually not viewable, interpretable in a text editor
  - On UNIX systems, “cat –v filename” will display in a ‘printable’ format
  - These files need special interpretation
  - E.g., integers stored in binary format

Binary Integers

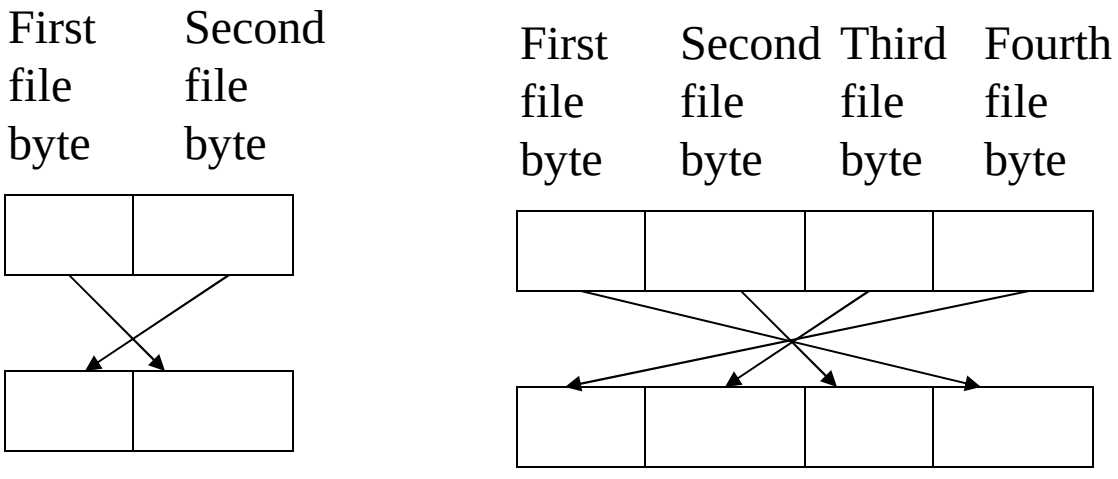
- MSB: Most significant byte first

Logical view



- LSB: Least significant byte first

Logical view



WAV files

- WAV files are ‘binary’
  - Integer values such as lengths are stored in binary integer format
- Header integer values
  - Stored in LSB format
- Data samples
  - E.g., 8 bit or 16 bit integers
  - Stored in LSB format
    - Not relevant for 8 bit samples
  - *Unsigned* integers
    - E.g., values between 0 … 255
    - What does this mean for amplitude values?

MIDI: Sources

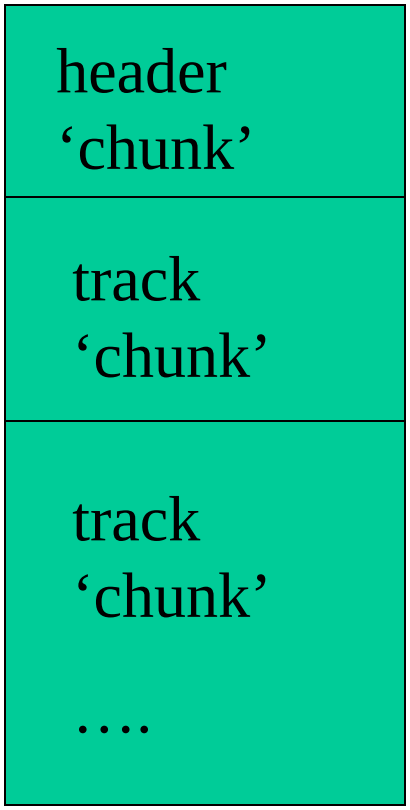
- Kientzle, T. (1998).
- http://crystal.apana.org.au/~ghansper/midi_introduction/

MIDI sound

- MIDI: Musical Instrument Digital Interface
  - Used to specify music
- Abstract representation
  - Part of the standard is sample based
  - Part based in specification of timing and instrument types
- General MIDI
  - Specifies 175 standard instrument sounds
- MIDI files are organized as a series of tracks
  - Each track might store piano, trumpet, and flute parts of a piece of music as separate tracks
  - Tracks contain events
    - Events specify a musical action; e.g., key press, or release
- Three types of files
  - 0) one track only
  - 1) multiple tracks, to be played back simultaneously
  - 2) multiple tracks, but no timing relation assumed

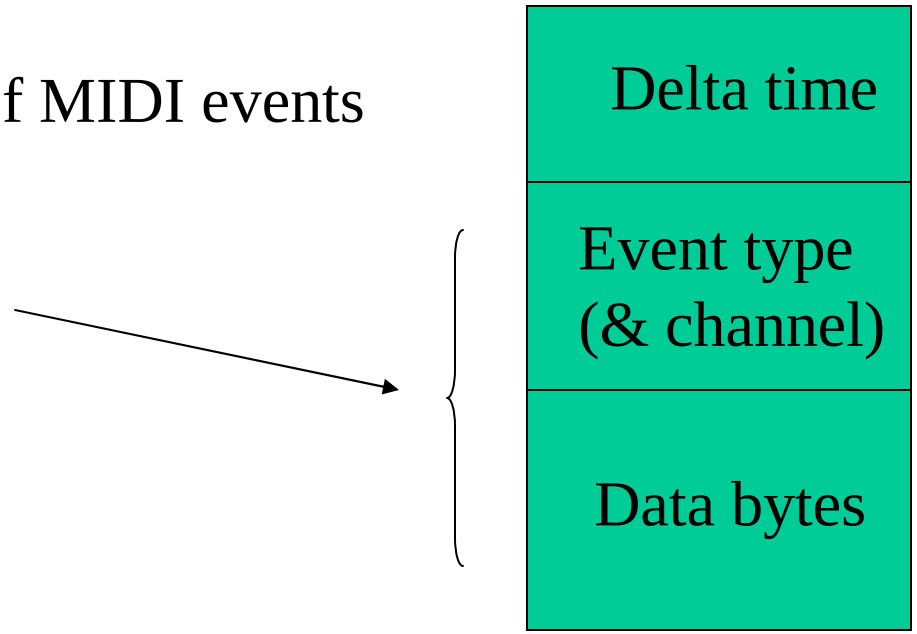
MIDI files

- Binary file



MIDI file format

- Header
  - Four byte ‘MThd’ chunk name
  - Four byte MSB integer
  - 6 bytes of format specification
    - file type (2 byte, MSB) (0, 1, 2)
    - number of tracks (2 byte, MSB)
    - time format (2 byte, MSB)
- Track
  - List of MIDI events
- Event



Variable length integer

Aka. *Status byte* (repeated status bytes can be omitted)

Delta time (unit: ticks)

- Amount of time that separates this event from the previous event
- *variable length* integer
  - Positive values only
  - 1 to 4 bytes
  - Last byte of integer stored with a 1 bit in high-order bit, others stored with a 0 in high order bit
- Interpretation of this time in terms of a duration depends on time format in header of file
  - -ve time format value: SMPTE (Society of Motion Picture and Television Engineers) standard
    - Tick values specified in hh:mm:ss;ff format
    - Time format specified as frames/second & ticks/frame
  - +ve time format value: musical tempo
    - Ticks per beat given in format value

Specification of Instrument Sounds

- General MIDI
  - Channels 1-9, 11-16
    - melody channels
    - One of 125 instruments (Table 22.5,Kientzle, 1998)
  - Channel 10
    - Rhythm channel
    - One of 47 instrument sounds (Table 22.6, Kientzle, 1998)
- DLS: Downloadable Samples Standard
  - Method for storing MIDI instrument sounds in a file
  - Format based on WAV

Image formats

- XML dia format (abstract)
  - Supports circuits, networks, flowcharts
  - Enables some new types of drawing objects to be defined
  - dia program generates gzipped xml
- BMP format (sample-based)
  - Device-independent “bitmap” files
  - Bitmap = sample-based
  - Often used in Microsoft operating systems
  - Based on RGB color model

XML dia Format

- Sources
  - http://www.gnome.org/gnome-office/dia.shtml
  - Dia home page
    - http://www.lysator.liu.se/~alla/dia/
  - Book chapter on dia
    - http://www.togaware.com/linuxbook/dia.html
- Example
  - dia-box.xml
  - http://www.cprince.com/courses/cs3121fall02/lectures/media/dia-box.xml
- Shape extension example
  - http://www.lysator.liu.se/~alla/dia/custom-shapes
  - dia-shape-ns.xml
    - http://www.cprince.com/courses/cs3121fall02/lectures/media/dia-shape-ns.xml

BMP Format

- Sources
  - Steinmetz, R. & Nahrstedt, K. (2002). Media Coding and Content Processing. Volume 1. Prentice Hall.
  - http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/BMP.txt
  - http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html

BMP

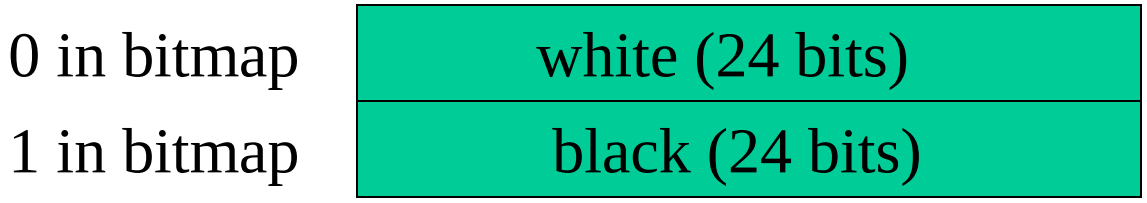
- Header followed by a data region

BMP

- Header
  - Sizes
    - Of file, in bytes
    - Width & height of image– in pixels
  - Color
  - Bits per pixel (color depth): 1, 4, 8, 24
  - Color lookup table (CLUT)
  - Compression method (omit for now)
- Data region
  - Value of pixels in a line
  - Lines of image stored in reverse order

Color Lookup Tables (CLUT) - 1

- Provides a way to specify color with a smaller number of bits
  - Each entry in the CLUT will be a full color specification for the present system hardware
  - E.g., 24 bit RGB
- 1 bit: needs 2 entries in CLUT
  - Black, white



Color Lookup Tables (CLUT) - 2

- 4 bits: Needs 16 entries in CLUT
  - Each of the possible 4 bit values indexes into the CLUT
  - Each entry in the CLUT is full RGB representation (e.g., 24 bits)
- 8 bits: Similar, but 256 entries in CLUT
- 24 bits: No clut, just RGB specification

CLUT Example

- How much information in a BMP file is needed to represent a 10x15 pixel b&w image with 1 bit pixels. Include only the pixel data and the CLUT in your calculations (assume 24 bit RGB in CLUT entries).
- Bytes for pixels of image

  10x15 image = 150 pixels

  150 pixels * 1 bits/pixel = 150 bits

  150 bits/(8 bits/byte) = 18.75 bytes (need to round up)

  $\lceil 18.75 \rceil = 19$
- Bytes for CLUT

  2 entries in CLUT (one for black, one for white) * 3 bytes for RGB per entry = 6 bytes
- Total

  19 + 6 = 25 bytes

Example without the CLUT

- How much information in a BMP file is needed to represent a 10x15 pixel b&w image, assume 24 bit RGB needed to specify colors of pixels (no CLUT)
- Bytes for pixels of image

  10x15 image = 150 pixels

  150 pixels * 3 bytes/pixel = 450 bytes
- Comparison

  26 bytes with CLUT

  450 bytes without

  450/26 = 17.3

  Savings of space: factor of 17

Media File Formats

- Image formats:
  - XML/dia, BMP; Others include: JPEG, GIF, PNG, TIFF
- Sound formats:
  - WAV, MIDI; Others include MP3, AIFF, AU
- Video formats:
  - AVI; Others include MPEG, Quicktime, Realmedia

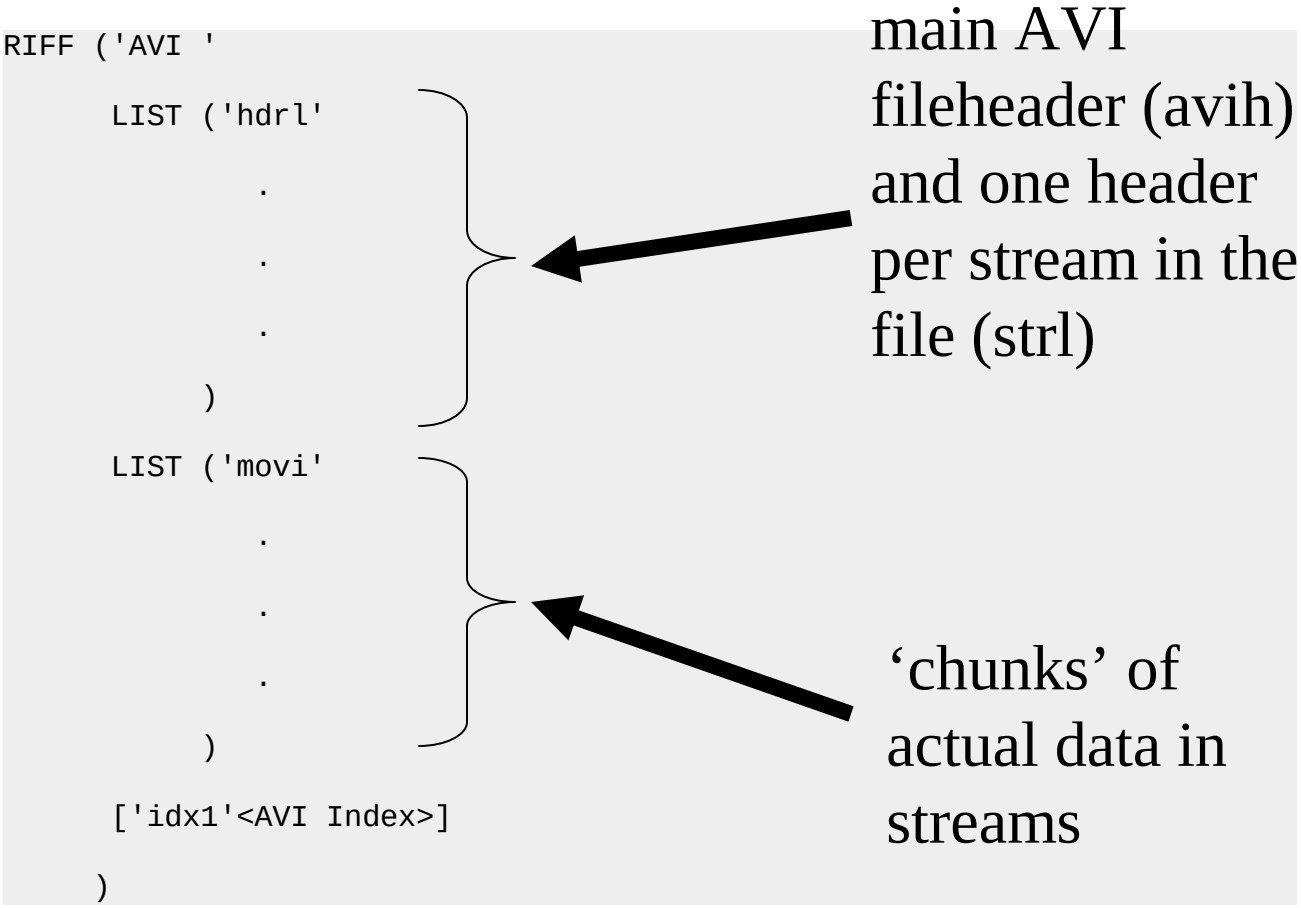
Video Formats

- Sources
  - Chapter 10 of lab text, pp. 213-223
  - Chapter 4 of main text, pp. 209-212
  - Steinmetz & Nahrstedt (2002), Chapter 7
  - AVI format reference:
  - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dx8_c/directx_cpp/htm/aviriifffilereference.asp
- Digital video formats
  - More complex than formats discussed so far
    - Typically include two data streams: video & audio
    - More data (in video); compression is more important
    - Quality of data required in applications may vary widely
      - E.g., Commercial quality vs. home quality

AVI files

- AVI: Audio-video interleave
- RIFF file format (same class as WAV)
- Used for video applications (capture, edit, playback)
- Can be compressed (“raw” video) or uncompressed
- Can contain >= 1 stream of data
  - Video, audio, text

AVI File Structure



http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dx8_c/directx_cpp/htm/aviriifffilereference.asp

Main header

```
typedef struct {

    DWORD dwMicroSecPerFrame;

    DWORD dwMaxBytesPerSec;

    DWORD dwReserved1;

    DWORD dwFlags;

    DWORD dwTotalFrames;

    DWORD dwInitialFrames;

    DWORD dwStreams;

    DWORD dwSuggestedBufferSize;

    DWORD dwWidth;

    DWORD dwHeight;

    DWORD dwReserved[4];

} MainAVIHeader;
```

- **dwMicroSecPerFrame**
  - Number of microseconds between frames.
- **dwMaxBytesPerSec**
  - Maximum data rate of the file: number of bytes per second.
- **dwFlags**
  - Contains any flags for the file; e.g., does the file contain an index chunk, or is the file *interleaved*, copyright
- **dwTotalFrames**
  - number of frames of data in the file.
- **dwInitialFrames**
  - Specifies the initial frame for interleaved files (zero for noninterleaved files).
- **dwStreams**
  - Number of streams in the file. For example, a file with audio and video has two streams.
- **dwSuggestedBufferSize**
  - Suggested buffer size for reading file. Should be large enough for largest chunk or record in file.
- **dwWidth**
  - Width of the AVI file in pixels.
- **dwHeight**
  - Height of the AVI file in pixels.

Stream header - 1

```
typedef struct {
    FOURCC fccType;
    FOURCC fccHandler;
    DWORD  dwFlags;
    DWORD  dwPriority;
    DWORD  dwInitialFrames;
    DWORD  dwScale;
    DWORD  dwRate;
    DWORD  dwStart;
    DWORD  dwLength;
    DWORD  dwSuggestedBufferSize;
    DWORD  dwQuality;
    DWORD  dwSampleSize;
    RECT   rcFrame;
} AVIStreamHeader;
```

After each stream header, is a *stream format* chunk.

- *Video*: same header as a BMP file
- *Audio*: Same format as FormatChunkData in wav.h in lab 3

Stream header - 2

- **fccType:** Type of the data in the stream: video, audio, text.
- **fccHandler:** Optionally, specifies installable compressor or decompressor.
- **dwPriority:** For example, relevant in a file with multiple audio streams.
- **dwInitialFrames**
  - How far audio data is skewed ahead of the video frames in interleaved files.
- **dwScale, dwRate**
  - Time scale for stream. **dwRate** / **dwScale** = number of samples per second. For video streams, this rate should be the frame rate. For PCM audio, the sample rate.
- **dwStart**
  - Starting time of file. For units, see **dwRate** & **dwScale**. Can specify a delay time
- **dwLength:** Length of this stream. For units, see **dwRate** & **dwScale**.
- **dwSuggestedBufferSize**
  - Size of buffer to read this stream. Typically, size of largest chunk in the stream.
- **dwQuality**
  - Measure of quality of data in stream. For compressed data, this typically represents the value of quality parameter passed to compressor.
- **dwSampleSize:** Size of a single sample of data. Set to zero if the samples can vary in size.
- **rcFrame**
  - Used in support of multiple video streams. Units for this member are pixels.
  - Destination rectangle for a text or video stream within the movie rectangle specified by the **dwWidth** and **dwHeight** members of the AVI main header structure.

Data Chunks

- Audio data
  - In AVI file as: (## is the stream identifier)
    ‘##wb’
    <frame data bytes>
- Video data
  - Either uncompressed
    ‘##db’
    <frame data bytes>
  - or compressed
    ‘##dc’
    <frame data bytes>

MPEG Standards

- MPEG-1:
  - Parts: Systems, Video, Audio
  - Applications include VCD (Video CD)
- MPEG-2
  - Parts: Systems, Video, Audio, Conformance, Reference Software Digital Storage Media Command and Control (DSM-CC), Advanced Audio Coding (AAC), Real Time Interface
  - Applications include MP3, DVD
- MPEG-3 (See MPEG-2)
- MPEG-4
  - Proposed fixed & mobile web standard
- MPEG-7: Multimedia Content Description
- MPEG-21: Multimedia Framework

MPEG: Audio

- MPEG standards have *layers:* Operating modes with increasing complexity and performance
- MPEG-1 provides
  - Mono and stereo coding at 32, 44.1, and 48 kHz sampling rate
  - Layer I, II, III: varying bit rates from 32 to 448 kbit/s
  - Optimized for 128 kbits/s stero
- MPEG-2 BC (Backwards compatible)
  - multichannel extension to MPEG-1; >= 5 channels
  - Bit rate range: up to 1 Mbit/s
  - Some lower sampling rates 16, 22.05, and 24 kHz for bitrates from 32 to 256 kbit/s (Layer I) and from 8 to 160 kbit/s (Layer II & Layer III).
- MPEG-2 Advanced Audio Coding (AAC)
  - High-quality audio coding standard for 1 to 48 channels
  - Sampling rates of 8 to 96 kHz
  - Multichannel, multilingual, and multiprogram capabilities
  - Bit rates from 8 kbit/s to more than 160 kbit/s/channel
  - Multiple encode/decode cycles
  - Three profiles (layers) of AAC provide varying levels of complexity and scalability.

Saving Palace CLUTs in *GraphicConverter*

GraphicConverter uses color table files much like Photoshop's. The procedure for creating a Palace CLUT is about the same.

> *Photoshop users take note — GraphicConverter color table files are not in the same file format as Photoshop's. If you use both programs, you will need to save a color table file for each program.*

First open most any Palace room GIF ("pgate.gif" is prefered, since it's guaranteed to have the complete map). The room pictures *should* all have the same color map, and your avs should use that same set of colors.

Once you've opened the GIF, select "Save As..." under the GraphicConverter "File" menu. A dialog will pop up, showing you the image. Change the Format from "GIF" to "Color Table" (the name will change to "pgate.PAL") and hit the "Save..." button (I saved mine as "Palace.PAL"). Once the color map is saved, you can just close the GIF — you won't be needing it any more.

> *One extra note of caution — if you've been using GraphicConverter's "split" option (part of the "Save As..." dialog box), make sure you've turned it off before hitting the "Save" button, or you'll get lots of little (and identical) color table files..... you only need one!*

Using the Color Map

For any picture you want to use with the Palace, as a room background or a prop, you'll need to make sure it uses that color map. Unless the picture started as a Palace image (like the room GIF, or something pasted *from* the Place prop editor), we'll need to convert it. There are essentially three kinds of pictures we will convert:

- RGB (24-bit or 32-bit) full-color images (such as color JPEGs)
- Grayscale (8-bit) images (such as B&W Photos, JPEG or GIF)
- "Alien" indexed-color images (such as GIFs you might find on the net)

For both grayscale and indexed-color images, you must convert to RGB color via the "Picture->Colors->Change to 16.7 Mill. Colors (32-bit)" menu *first,* before converting to indexed color using the Palace color map. GraphicConverter will always assume that grayscale GIFs should be mapped to exactly 256 grayscales; and it has no good way to directly convert from one color table to another. A short time in RGB mode is the neccesary in-between step.

Remember: *Save Your RGB Image*. You may need it later, for changes or fixes.

To force GraphicConverter to use your stored Palace color map, you select "Colors->Options..." under "Picture." A dialog will pop up. Press "Use Custom Color Table" and the "Open..." button. In the "Dither" check box, press whatever you like — "dither off" will generally give you the smallest file size, while "dither on" will give the nicest tonal gradations. Now press "OK."

Once the color table is ready, we can convert RGB images the Palace map. Select "Colors->Change to 256 Colors (8 bit)" under "Picture." Now look at your image: it's living in the Palace world.

RGB Original

No Dithering

Dithered

Not sure about that "Dither" option? Just hit "Undo" and convert it again in several different ways. Or duplicate the image and convert each copy a different way, to compare side-by-side. Above are three samples, superimposed over a bit of Harry's Bar.

The Palace Ring 

[Feedback/Suggestions/Guess Book]

Updated 4 Nov 96

EMFFT.DOC

(last update Mar 5, 1992)

A. INTRODUCTION

EMFFT is used to examine and/or manipulate FFT data. Filtered FFT data may be output to a disk file with the normal FFT format or it may be back-transformed ("B") to produce a filtered IMAGE stored on disk (normal IMAGE file format). Note that use of two or more options may produce additive or multiplicative effects on an FFT.

NOTE ON TERMINOLOGY: the term TSU (transform sample unit) denotes a unit of reciprocal space measure. Each TSU corresponds to a single step in the transform. For example, for a IDIM1 by IDIM2 transform in which IDIM1=IDIM2, each TSU will represent 1.0/(IDIM1*pixel size). Thus, if IDIM1=512 and the pixel size = 0.51nm, then each TSU is (1.0/261 nm). Because the right, left, top, or bottom limits of a 512x512 transform are at 256 TSU, the resolution limit at these extremes is 256*(1.0/261 nm) = (1.0/.981 nm), or 2.0 pixel resolution, consistent with the Whittaker-Shannon sampling restrictions.

B. PROGRAM OPTIONS

EMFFT provides several options which are selected from a TOUCH-KEY MENU as shown below.

TOUCH-KEY OPTIONS:

A	Multiply two FFTs (CCF)	M	FFT --> MAP format
B	BACK-transform the FFT	S	Get STATISTICS
C	Compute CIRCULAR average		
D	DISPLAY on graphics device	O	OPEN new FFT file
F	FILTER the FFT	R	READ and transform IMAGE data
G	Extract structure factors	W	WRITE current FFT to disk
K	*FIX values in FFT	E	EXIT program
L	LATTICE refinement	?	INFO on EMFFT program

PF1 Spawn to VMS DCL commands
PF2
PF3 Turns main MENU option list ON/OFF
PF4 Graphics routines
* Option not yet available

The first option the user must choose is either "O", to OPEN a new FFT file and read in the array of structure factor data, or "R", to READ in an IMAGE file which is then Fourier transformed. Whenever option "W" is chosen to WRITE a FFT array to a disk file, the user must again choose either "O" or "R", or else EXIT the program ("E"). You may string several operations together before saving the result in a disk file. Realize that operations are performed sequentially, thus the only way to fully recover an "untouched" FFT is to reuse option "O" or "R".

OPTION A: Multiply two FFTS

This allows one to multiply the current FFT data with another FFT. The filename for the second FFT is entered by the user. Note that it is possible, using this option, to produce either an auto-correlation or cross-correlation FFT which can be back-transformed using option "F" to produce the corresponding auto- or cross-correlation image as is accomplished using the program EMCOR.

OPTION B: BACK transform the current FFT

This option will take the current FFT, including any filter masks that have been applied, and back transforms it to produce a "filtered" IMAGE. If an FFT that has had no mask applied is back transformed, then an unfiltered IMAGE identical to an original boxed IMAGE will be the result. The only input required by this option is a filename for storing the resulting IMAGE and an optional HEADER. To do anything with the back-transformed IMAGE you must exit EMFFT ("E" or touch PF1 and run, for example, EMIMG).

OPTION C: Compute CIRCULAR average

This option will produce a circular average of the FFT amplitudes (DEFAULT) or intensities that can be displayed on the graphics screen and/or dumped to an ASCII file (FFT_1D.TMP). This option is similar to the "C" option of EMIMG.

First decide what interpolation factor to use (type a negative number to exit the routine). If you plan to use the FFT_1D.TMP data file to produce a plot on one of the personal computers with, for example, a program like LOTUS FREELANCE, etc., then just use an interpolation of 1.0. Choose whether you want to display amplitudes <CR> or intensities (1), whether or not you want to see a display on the graphics device (DEFAULT=N), and whether or not you want to create the FFT_1D.TMP file (DEFAULT=N).

If you display the circular average on the graphics screen, you will then be able to use the data tablet puck to pick points and get a readout of the radial distance (in TSU) from the origin. Hit the GREEN button to exit this routine and either start over with a new interpolation factor, or exit the "C" option by setting the interpolation value to a negative number.

OPTION D: DISPLAY the current FFT

"D" is used to display the FFT amplitudes or intensities, scaled in a variety of ways. The scaled FFT values are displayed with a black & white Color Look-Up Table, CLUT, using the lowest 6-bit planes of the graphics memory. Several display options are available as specified in the following touch-key menu.

FFT_DSP TOUCH-KEY MENU		Values
A	AMPS/INTENSITY toggle	amps/INTENSITIES
C	CLEAR display	
D	DISPLAY the FFT	
M	MAGNIFICATION (FFT size)	512 X 512
L/R	Portion of FFT displayed	LEFT/RIGHT
T/B	Portion of FFT displayed	TOP/BOTTOM
P	POSITION center of display	320 256
S	SCALING method	LINEAR/log/non-linear
%	Threshold FRACTIONS	0.000 0.005
E	EXIT	

SUB-OPTION A: AMPS/INTENSITY toggle

This toggles between a choice of whether amplitude or intensity FFT data are used.

SUB-OPTION C: CLEAR the display screen

"C" erases the entire graphics display screen.

SUB-OPTION D: DISPLAY the current FFT

"D" will display the FFT with the current settings given in the right half of the touch-key menu.

SUB-OPTIONS L/R/T/B: Select portions of the FFT for display

These keys select/deselect portions of the FFT for display. The initial DEFAULT is to have LEFT/RIGHT/TOP/BOTTOM all selected so the entire FFT is displayed. Certain options are disallowed (e.g. TOP and BOTTOM or LEFT and RIGHT cannot both be deselected simultaneously).

SUB-OPTION M: Toggle FFT display MAGNIFICATION

"M" is used to toggle through different size displays (if allowed). The initial DEFAULT is to display the FFT at maximum size allowed by the graphics device (1024x1024 for the LEXIDATA in B-141 and 640x512 for the LEXIDATA in B-403). The display can be no SMALLER than the actual dimensions of the FFT (IDIM1,IDIM2).

SUB-OPTION P: Select center POSITION for display

Choose "P" to select a new center position (2I FORMAT) for the FFT display. The initial DEFAULT (either 640,512 or 320,256) depends on which graphics display device is being used. Anytime you input a new center position, that becomes the new DEFAULT. Note that this routine permits FFTs to be displayed with a portion outside the graphics screen boundary.

WARNING: EMFFT options such as "L" assume that the FFT is Displayed EXACTLY centered in the graphics screen, otherwise they will not work properly.

SUB-OPTION S: Toggle SCALING method

"S" is a toggle switch that selects how the FFT data are scaled. Linear scaling is the initial DEFAULT mode, but the data can also be displayed on logarithmic or non-linear scales. DEFAULT thresholds (see sub-option "%" below) are automatically selected each time you switch to a different mode of scaling. These DEFAULTS can be overridden with the "%" option. Non-linear scaling produces an asymptotic CLUT which depends on two parameters, K and S, set using the "%" option.

SUB-OPTION %: Select threshold FRACTIONS

The "%" key is used to select threshold FFT values (FRACT_MIN and FRACT_MAX, given in fractional units of the maximum, scaled FFT value) so that values below FRACT_MIN*FMIN are displayed with the lowest CLUT intensity (usually 0) and values above FRACT_MAX*FMAX are displayed with the highest CLUT intensity (usually 63). DEFAULT threshold values are used whenever the linear or logarithmic scale modes are selected but a different scheme is used if non-linear scaling is selected (see below). The DEFAULTS for linear and logarithmic scaling modes are set as follows:

Scale Mode	FRACT_MIN	FRACT_MAX
Linear	FAVG/FMAX	(FAVG+FSTD)/FMAX
Log	LOG10(FMAX)*FAVG/FMAX	LOG10(FMAX)*(FAVG+FSTD)/FMAX

where FAVG, FMAX, and FSTD are the average, maximum, and standard deviations of the FFT amplitudes or intensities (depending on the current choice selected by "A").

If "%" is chosen when non-linear scaling is chosen, two parameters, K and S, are specified. K fixes the midpoint of the CLUT (range = 1.0-255.0) and S effectively sets the slope (contrast) of the CLUT. The initial DEFAULT values for K and S are set as follows:

```
K = 255.0*(FAVG+FSTD)/FMAX
S = 0.25
```

The value for S is set initially to 0.25 although this may need to to be changed. Higher values give a steeper curve (higher contrast) whereas lower values give a flatter curve (lower contrast). Beware that very small or large values of S can and probably will lead to overflow or underflow problems and will crash the program. The value of K will almost certainly have to be adjusted from the DEFAULT setting. If the screen appears totally white, K is too small (and S may be too small). If the screen is totally black, K is probably too large. Obviously, to get non-linear scaling to work properly requires much operator intervention and experience. The linear and logarithmic scaling modes should suffice for most "normal" circumstances.

Once K and S are set, the CLUT value for a given FFT intensity or amplitude is remapped as followings:

```
CLUT_VALUE = 255.0*(1.0/((10.0**(S*(K-VALUE-1.0)))+1.0))
```

where VALUE = (F*(LUTHI-LUTLO)/FMAX) + 5.5 and LUTLO and LUTHI are normally set equal to 6 and 63 for the compressed black and white FFT CLUT.

OPTION F: FILTER the FFT

"F" is used to generate filter masks to multiply the FFT data by. There are currently two separate sub-options in the following touch-key menu:

EMFFT_FILTER TOUCH-KEY OPTIONS

A Apply low and/or hi-pass filter
B Apply 2D reciprocal lattice mask
C Apply 2D layerline mask
E EXIT routine

SUB-OPTION A: Apply low and/or high-pass filter (Resolution cutoff)

Enter values for RES_MIN, RES_MAX, CUT_MIN, and CUT_MAX (4F FORMAT) to identify the inner and outer radii in the Fourier transform (in TSU) for the HI-PASS (RES_MIN) and LOW-PASS (RES_MAX) filters. CUT_MIN and CUT_MAX specify hard (0.0) or soft (>0.0) edges.

For example, for HI-PASS filtering, the FFT is zeroed out to RES_MIN-CUT_MIN, and rises exponentially up to full weight (1.0) at RES_MIN. A LO-PASS filter has full weight out to RES_MAX and drops exponentially to 1/exp at RES_MAX+CUT_MAX (if CUT_MAX=0.0, the FFT is zeroed beyond RES_MAX). To produce a simple low-pass filter (i.e. with no high-pass filtering), set RES_MIN=0.0. For simple high-pass filtering, set RES_MIN=RES_MAX>0.0. Use CUTMIN or CUT_MAX = 0.0 to apply a sharp cutoffs with no Gaussian falloff.

SUB-OPTION B: Apply 2D reciprocal lattice mask

This is used to produce a mask with circular "holes" at the positions of an ideal, 2D reciprocal lattice. The vector coordinates of the two principal lattice dimensions (a* and b* given in TSU) may be entered by hand (4F FORMAT) or read in directly from the FFTLAT.DAT file produced after refinement of the reciprocal lattice by use of EMFFT option "L".

Specify the radius for all circular holes (in TSU). You may create holes with "soft" or "hard" edges. For holes with "hard" edges, points inside the hole are given full weight and points outside are set equal to zero. In "soft" holes, points inside the hole are scaled with a Guassian weighting function with the weight determined by the relative distance of the point from the center of the hole: points at the hole edge are down weighted by 1/exp of their original value. "Soft" holes are generally made using a slightly larger radius than is appropriate for "hard" holes.

SUB-OPTION C: Apply a 2D layerline mask

This produces a mask consisting of a series of parallel slits (rectangular holes). Specify values for ANGLE,SPACE and FALLOFF (3F FORMAT). ANGLE (in degrees) specifies the orientation of the slits with respect to the horizontal (X) direction of the FFT (a positive value is measured as a counterclockwise rotation from the X-axis). SPACE is the distance between slits in TSU. FALLOFF determines the "softness" of the slit edges: sample points a distance FALLOFF from the center line of the slit are downweighted by 1/exp. There is currently no provision to construct "hard" edge holes as is allowed in sub-option "B". Also, the parameters ANGLE, SPACE, and FALLOFF must be entered by hand.

OPTION G: Extract structure factor

This allows one to extract structure factors from the FFT of the image of a 2D crystalline specimen. This is usually run after running option "L" which computes refined parameters for the 2D reciprocal lattice (AX,AY,BX,BY in TSU), for a* and b*, the two principal lattice vectors). The lattice coordinates are normally stored in the file FFTLAT.DAT, which is read back by this routine. It is thus IMPERATIVE that the correct FFTLAT.DAT file be read in. You also have the option of entering the reciprocal lattice coordinates by hand.

Alternatively, this option can be used after using EMCORAVG (correlation averaging) if one wants to feed into the Fourier-based software routines. EMCORAVG.DAT contains the required reciprocal lattice coordinates. Because EMCORAVG is often run in the mode which forces the reciprocal space lattice coordinates to occur at integer transform sample unit positions, (enabling peaks of the reflections to be exactly sampled in the FFT), this routine uses the ORIGINAL refined lattice coordinates for determining a*, b*, c*, alpha*, beta*, gamma*, d*, etc., but uses the "enforced" coordinates to extract the structure factor measurements. The ORIGINAL lattice dimensions are retained in the header records of the structure factor file in order to override the distortions imposed when forcing the reciprocal lattice to exact TSU positions.

Begin by specifying a filename and header text for the SF output file.

Structure factors are computed in one of two ways:

1) Transform intensity data, obtained within a circular or

elliptical window, which is centered about the ideal reciprocal lattice position for each reflection, are INTEGRATED with each transform sample point weighted by a Gaussian function whose magnitude is proportional to the distance from the calculated lattice position to the sample point as given below:

The intensity at each sample point is weighted by W**2, where

$$W = \exp\left(\frac{-d*d}{2.0*r*r}\right)$$

with d = distance (in TSU) of the sample point from the ideal lattice position.
r = radius of the window (in TSU).

For example, with r = 1.00

d	= 0.00	0.50	1.00	1.50	2.00
W	= 1.00	0.88	0.61	0.32	0.14
W**2	= 1.00	0.78	0.37	0.11	0.02

Choose the window shape (DEFAULT=circular) and RADIUS (DEFAULT=1.0 TSU: this must be large enough to include the entire spot) and set the cutoff variable, CUT (DEFAULT=2.0). CUT determines the actual dimensions of the window used for integration and background calculations. Intensities are integrated inside a window with a radius = RADIUS*CUT and background measurements are determined from within an annulus whose inner and outer radii are RADIUS*CUT and 2*RADIUS*CUT, respectively. If you choose not to subtract backgrounds, CUT is set automatically = 2.0. Applying the background correction to the structure factor measurements helps to reduce the noise, especially for the higher resolution data. The background value at each lattice point is estimated from the average intensity in an annular region outside the integration window. **WARNING:** Choose RADIUS and CUT with care to avoid artifacts produced when peaks of neighboring spots are erroneously included in the background area.

The structure factor amplitude is computed as the square root of the (weighted) integrated intensity. Measurements are then scaled down by an arbitrary factor of 1000.0 to help avoid overflows in the formatted output file.

The structure factor phase for each reflection is obtained by interpolation of the four sample points adjacent to the calculated

lattice position. Phase measurements cannot be computed by integration of all the sample points inside the reflection window mainly because the phase term shows large fluctuations across and just outside the the reflection peak due to the SIN(X)/(X) influence from the edges of the boxed (and floated), digitized micrograph.

2) Structure factor amplitudes may be computed by interpolation as is done to obtain phases. If this option is used, there is no need to set a window shape or size, and the option to subtract backgrounds is not available. This option is mainly useful for extracting structure factor data from models or from the Fourier transforms of image averages after running the correlation analysis procedures (where there should be little or no noise surrounding the reflections).

Specify the Miller indices of the two principal lattice vectors (DEFAULTS = 1,0,0 and 0,1,0). This option is really only important for correctly identifying the spot indices for a three-dimensional data set such as from several tilt series from orthogonal sections of a Three-dimensional crystal. The [hkl] indices for the two principal lattice vectors in the two-dimensional projection data provides a transformation matrix which is used in subsequent programs (eg. EMSF2DBT) to identify the orientation of the data when processed. For example, assume we had an [h0l] projection such that the c* direction was horizontal and a* was vertical in the FFT. Since the lattice refinement routines ("L" option) use the convention a* = near horizontal vector and b* = near vertical vector, then the two principal lattice directions would be identified as [hkl] = 0,0,1 for a* and [hkl] = 1,0,0 for b*. Thus, to transform from the 2D h'k' indices used in option "L" to 3D [hkl] indices output by this routine, the following transformation matrix algebra is performed:

$$hkl = h'k' * T \quad \text{where} \quad T = \begin{vmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \end{vmatrix}$$

$$\begin{aligned} \text{Thus,} \quad h &= h'*a11 + k'*a21 \\ k &= h'*a12 + k'*a22 \\ l &= h'*a13 + k'*a23 \end{aligned}$$

```
Example for T = |0 0 1| :     h' k'     h  k  l
                |1 0 0|     ------    ---------
                            1  0      0  0  1
                            0  1      1  0  0
                            1  1      1  0  1
                            2  3      3  0  2
                           -5  9      9  0 -5    etc.
```

The SF output file contains information on the unit cell parameters, the transformation matrix, and the structure factor data (amplitudes and phases) in the format given in [TSB.FOR]EMPROGS.DOC. The SF data can be refined using EMSFREF and back transformed using EMSF2DBT (or EMSF3DBT) to produce MAP files which can be displayed with EMMAP.

This routine also generates a temporary file (FFTSPT.TMP) which contains a listing of several of the parameters used to compute the structure factors as well as the structure factor data.

You also have the option (DEFAULT=N) to create a file (FFTBOXES.TMP) that lists the 9 by 9 arrays of amplitudes and phases for each spot, in a format similar to that used in the EMFFT option "L" (sub-option "B"). Be sure to print out the file on the line or laserprinter in LANDSCAPE (NOT PORTRAIT) mode, otherwise much of the file will not print out on the page. The data as printed out in the FFTBOXES.TMP file may differ slightly from the data output in the structure factor and FFTSPT.TMP files because some parameters (e.g. RADIUS=1.0; CUT=2.0) are fixed.

OPTION K: Fix values in FFT

Option not yet available.
Documentation needs to be written.

OPTION L: LATTICE refinement

"L" allows you to refine two-dimensional crystalline lattice parameters. Before running this option, you must be sure that you have displayed the FFT to maximum size on the graphics device. You are given one of the two following warning messages depending on which graphics device you are using:

WARNING: FFT must be displayed at 1024x1024 size and centered at 640,512
WARNING: FFT must be displayed at 512x512 size and centered at 320,256

"L" currently has provides two separate sub-options in the following touch-key menu:

```
EMFFT_LATREF TOUCH-KEY OPTIONS
------------------------------
A  Use overlay grid to refine lattice
B  Least squares fit of individual spots
E  EXIT routine
```

SUB-OPTION A: Use overlay grid to refine lattice

This allows you to select two spots of known indices with the data tablet puck and the program then draws a reciprocal lattice net over the displayed FFT. You can then reenter either or both spots and keep redrawing the lattice net until you are satisfied that the net accurately fits the lattice reflections. This method is somewhat more subject to error than sub-option "B" (below) which does a least squares fit on the basis of as many input spots as you care to identify.

First enter the densitometer pixel size in nanometers (DEFAULT=0.51nm which corresponds to 25 micron sampling of a micrograph at 49,000 magnification). This is only used for giving real-space lattice statistics while you are fitting the net. Thus, if you aren't interested in accurate real-space parameters, you can just ignore this by hitting the <CR> key. Next enter the Miller indices for the two spots, [H1,K1] and [H2,K2], you plan to select with the tablet puck (4I FORMAT). If you misindex these spots, the reciprocal lattice net and lattice parameters will be incorrect and you will be unable to build a proper filter mask with option "F", for example. Choose the R,G,B color of the overlay grid (3I FORMAT: DEFAULT=150,150,150=light grey). At this point control is transferred to the data tablet puck and the following menu appears on the terminal screen:

```
Mark two spots with tablet cursor:
YELLOW = enter new position
WHITE  = keep old position
BLUE   = turn grid on/off
GREEN  = EXIT
-------------------------------------------------------------------
ASTAR     BSTAR    GAMSTR     ADIM     BDIM      GAMMA
```

Carefully select the positions of the [H1,K1] and [H2,K2] spots (in that order) with the YELLOW puck button. After entering these two positions, the overlay reciprocal lattice net is drawn. Use the BLUE puck button to toggle the net off and on. To change the lattice, you must always enter two spots. Use WHITE to accept the

previous position and YELLOW to select a new position. Hit GREEN to exit from the selection process after which you are allowed to restart or finish the session. If you choose to restart, the routine will again prompt you for the pixel size, grid overlay color, etc. as before.

If you want to exit the routine completely, you have the option to save the reciprocal lattice parameters in the file FFTLAT.DAT (DEFAULT=N). This file stores the X and Y coordinates (in TSU) for the [1,0] and [0,1] lattice vectors. These two vectors can then be used in subsequent EMFFT routines such as "F" to build filter masks with holes positioned at the points of an ideal lattice defined by these principal vectors.

SUB-OPTION B: Least squares fit of individual spots

This option helps determine the lattice parameters of diffraction patterns. The LEXIDATA TABLET cursor is used to pick out spots for a least-squares determination of a crystalline lattice. The optional end product of the routine is a file (FFTLAT.DAT) containing the coordinates (in transform sample point units) of the [10] and [01] lattice vectors.

CRYSTAL LATTICE REFINEMENT STEPS

1. Choose a cursor type and color.

2. Mark two reference spots with the Lexidata tablet cursor and enter the H,K index for each. Be certain that the two spots do not both lie on a line which intersects the origin of the diffraction pattern. WARNING: the indexing must conform to usual crystallographic conventions in order for certain options in the refinement program "EMSFREF" to work correctly. Also, it is highly recommended that you choose two reference spots which have reasonably high indices (i.e. don't just mark the [10] and [01] spots) since this will give a better initial estimate of the reciprocal lattice parameters. If you are somewhat sloppy at this step, the indexing of high resolution spots may be wrong due to the magnification of small errors made when low resolution spots are used as reference.

3. Choose the maximum number of contour intervals (default NC = 25; range = 5-25) and the maximum contour value (default CLIM = 999.0; range = 0.0-999.0) to scale the contoured amplitude displays. If the amplitudes cover a large dynamic range, it is best to set NC = 25 and CLIM < 999., otherwise some of the weaker, high resolution spots that are important for accurate lattice refinement may not show up in the display because they won't be contoured at the lowest level.

4. Pick spots using the YELLOW puck button. 9x9 amplitude and phase arrays will be displayed at the terminal and a contour display of the amplitude array will be appear on the LEXIDATA (the diffraction pattern will disappear from the graphics screen at this point).

5. Place the tablet cursor over the spot center and hit the YELLOW button to accept the position. WARNING: Try not to be overinfluenced by sharp peaks; mark the center of gravity of the WHOLE spot. DX,DY, listed at the terminal, identifies the position you have chosen, measured in coordinates relative to the center of the 9x9 array (+X to the RIGHT and +Y UP). If you wish to bypass the spot, hit either the WHITE, BLUE or GREEN buttons.

6. After a minimum of two spots have been successfully entered, the current values for the lattice parameters can be obtained by hitting either the WHITE, BLUE or GREEN button. After the lattice parameters are listed, you have the option (Y,W or B) to delete measurements by hand, or to continue (G). To delete spots, type the H,K index (2I format) for each spot followed by a <CR>. A second <CR> will return you to the program where you may then EXIT the routine or start entering additional spots. Please note that, depending on which option you choose at this part of the routine, you may be required to enter subsequent options either from the data tablet puck or from the terminal keyboard. Thus, if you hit the GREEN button to bypass deleting spots by hand, then your next option is answered through the puck, NOT at the terminal keyboard. This helps insure that you don't have to needlessly keep moving your hand from the tablet puck to the terminal and vice versa. Likewise, if you hit Y,W or B to delete spots, then your next option will be answered from the terminal keyboard unless you resume picking out new spots for refinement.

7. When the routine is exited, you are given the option to store The x,y coordinates (in transform sample point units) of the [10] and [01] lattice vectors in the ASCII file FFTLAT.DAT. This is then used by EMFFT option "G".

ADDITIONAL NOTES: Experience with samples such as catalase crystals and gap junction membranes indicates that the lattice parameters should be considered valid if at least 10 or more well resolved spots can be refined so that none deviate more than .75-1.0 transform sample point from the ideal lattice. Treat with caution any spots that deviate more
than 1 sample point.

OPTION O: OPEN an existing FFT stored on disk

"O" allows you to read in an FFT stored on disk.

OPTION R: READ and transform IMAGE data

"R" allows you to read in an IMAGE and Fourier transform it for use by the remaining EMFFT options. The DEFAULT size of the FFT will be the smallest power of two in each dimension with a minimum size limit of 128 by 128. Thus, if your IMAGE is 63 by 513 pixels (NCOL,NROW), then the DEFAULT FFT size is 128 by 1024. You may INCREASE either FFT dimension up to a maximum of 1024 TSU.

OPTION S: Get FFT STATISTICS

"S" is used to obtain the current values of FFT_AVG, FFT_STD, and FFT_MAX.

OPTION W: WRITE the current FFT to disk or line printer

"W" allows you to save the current FFT (including any changes that have been applied) to a disk file with the standard FFT file format (DEFAULT) or a file that is ASCII formatted for dumping to the lineprinter ("1").

The lineprinter-type output file contains the FFT amplitudes and/or phases in a format suitable for dumping to a lineprinter or laserprinter. Currently, the output is restricted to the central 128 TSU, centered about the Y-axis (meridian) of the FFT. This is completely adequate for displaying 128x128 up to 256x1024 FFT files, but may leave out significant data from FFT data with IDIM1>256. With this option you can output amplitudes and/or phases. If you output phases, you may change the FFT origin (FFT_ORIGX,FFT_ORIGY) if you wish. The amplitudes are normalized to a LOG2 scale with a maximum value of 9. Phases are coded using the characters 0-9,A-Z to represent 0-360 degrees in 10 degree steps.

After using "W", you MUST select "O" or "R" to continue.

==========

The FORTRAN code for EMFFT is in
JUSTEM$DKA0:[TSB.FOR]EMFFT.FOR,.SUBS.

This documentation is in JUSTEM$DKA0:[TSB.DOC]EMFFT.DOC 5-Mar-92

==========