Building ICC Profiles - the Mechanics and Engineering

Dawn Wallner

(dawn.wallner@yahoo.com) 04/2000
Corresponds to ICC Specification ICC.1:2000

ISBN 0-000-000000-0

ABCDEFGHIJ-DO-89

CHAPTER 1

Who should read this book?

If you find yourself in the exciting and adventurous position of needing an International Color Consortium (ICC) device profile and have some knowledge of color and your device, this book will help you learn where to put the bits and bytes to build a profile. If you need to find out exactly what is in an ICC profile, this book will help you to find the location of the data and read the bits and bytes.

Introduction and Disclaimer

This book was instigated by attendees of a class given by George Pawle (Eastman Kodak Company) at the Color Imaging Conference in Scottsdale, AR. in 1997. They asked if the information George was presenting was written down anywhere. This is the result of an attempt at doing so. George and I did teach a course with somewhat similar content previously, but it addressed a specific color management solution from Kodak. This book attempts to be independent of any color management interface or solution.

Any mistakes or misconceptions in this book are entirely the responsibility of the author. This book is not sanctioned by the International Color Consortium (ICC), but I do hope the members will send comments which will aid in enhancing its accuracy and completeness. The ICC specification is somewhat dynamic, but this

document will attempt to keep current. In cases of conflicting information, the specification is always right.

What does this book cover?

This book addresses the structure of an ICC profile, methods for getting data into and out of the profile and suggestions for making the lookup tables (LUTS) more accurate or smaller. Code samples are provided in C for reading and writing the profile tags.

What does this book not cover?

This book does not address the methods of obtaining device measurements, algorithms for rendering intents, gamut matching, appearance modeling, etc. There are a plethora of methods for getting color information from/about a device, some of which may be more accurate than others. There are also many algorithms for manipulation of the data to include color appearance models, some of which may be proprietary, provide varying accuracy and speed, or provide certain desired effects. Perhaps a follow-on book will be written by a color scientist on "Building ICC Profiles - the Art and Science"

What should the user know to effectively use this book?

The reader/user should know enough about color to understand color conversions and how to use device characteristics data, combined with related color data, to effect conversions while retaining color integrity. Since the measurements, equations, and methods may be provided to the reader from their local color scientist, knowledgeable readers need not be color scientists themselves.

What tools and knowledge does the user need in order to provide measurement data and other information to create a profile?

The user would need to know what hardware and methodologies are appropriate for gathering data for characterizing the hardware. They would also need in-depth knowledge of the types of transforms which need to be applied to the data to accomplish a conversion into and out of the connection space while retaining appropriate color accuracy. Descriptions of some of the appropriate transforms may be found in color science books.

ICC Profile Format Specification locations, other references

The ICC Profile Format Specification can be down-loaded from http://www.color.org. At the time of updating this book, the version of the spec to be released on the web was to be version ICC.1:2000-01. Unfortunately, the specification never quite made it to publication. For the purpose of this book, the difference between the 1998 spec (plus addendum) and the 2000 spec is primarily a change in the section numbers I reference in this book. At some time during 2000, I expect that a revision of the specification will be published on the web.

The ICC web site also provides the current list of ICC members, with links to several of their company sites for further information on current color management offerings.

This book (in pdf format) and sample C code may be obtained from the above web site. Comments may be sent to the author at dawn_wallner@yahoo.com. This book may be offered on the web in HTML in the near future.

This may be the last update I provide of this book, since I am retiring and will not be following the updates to the ICC specification. The examples should still be valid for future revisions, if the tags are retained. If anyone would like to take over updating this book and code and continue providing it to the public, please email me at the address above.

A list of the registered signatures may be obtained from the www.color.org web site. You may also register your manufacturer and model signatures at this web site.

Chapter and appendices contents

Chapter 2 provides an overview of the ICC profile, what types of ICC profiles are defined by the specification, and where they are used. The structure of the profile is presented as well as the data types and encodings used to represent the data in the profile.

Chapter 3 provides a cross reference for tag names in the specification and in a sample icc header file, as well as sample code for reading and writing each of the tag types.

Chapter 4 delves a bit deeper into the profile, detailing two models available in the ICC profile for describing/representing the color data for the hardware (or color space) and transformations for moving into and out of the connection space

Chapters 5, 6, and 7 look at hexadecimal dumps of monitor, scanner and printer device profiles, respectively. How to find and follow the various tags and accompanying data in these profiles is described.

Chapter 8 highlights differences between the previously dissected device profiles and device link, color space, abstract, and named color profiles.

The appendices include an overview of the number systems used in this book and the ICC header file C code.

The code used in examples for this book is available at the web site or from the author (dawn_wallner@yahoo.com). The examples provided would not necessarily reflect realistic data to put in a profile nor realistic device descriptions. The code itself is intended to be as simple to understand as possible, not elegant or fast. The code was developed on a sparc Unix workstation, but has been tested on Windows NT.

Available code

- icctags.c - example code which will dump any profile or create a profile with an example of each type of tag. Use this program to dump the profiles created by the code below.
- icctags-mon.c - sample code which creates the profile in “Dissecting Display Profiles”

- icctags-in.c and icctags-in2.c - sample code which creates the profiles in “Dissecting Input Profiles”
- icctags-out.c - sample code which creates the profile in “Dissecting Output Profiles”.
- icctags-link.c, icctags-ncl2.c, icctags-abs.c - sample code which creates the device link, named color 2, and abstract profiles in “Dissecting Other Profile Types”.
- icctags-resp.c - sample code showing the creation of two new tags which will be added to the ICC Specifications in the next release of the spec.
- makefile - compiles and links icctags.c
- makefile2 - compiles and link the rest of the sample code.
- Note that the code which creates the colorspace code example is not included due to its size. I used a real colorspace conversion profile in this case.

Differences between this version and previous version

Note that this is the second version of this book and of the code. The differences include the addition of new tags, fixed typos and bugs, and updated code which really has been tested in Visual C++ thanks to Max Derhak of Onyx Graphics. Thanks also to James Cao of Xerox for finding so many bugs and telling me about them!

This April, 2000 version of the book and code fixes a couple additional bugs in the header tag handling. Both the attributes and rendering intent values were located in the wrong bytes of the header. As always, if there is a difference between this book/code and the specification, the specification should take precedent.

CHAPTER 2

ICC Profile Overview

What is an ICC Profile?

An International Color Consortium (ICC) profile is a file of data describing the color characteristics of a device, such as a scanner, monitor, or printer. The primary purpose or use for this file is tobe used by color management software to maintain color consistency in imagery viewed, displayed or printed on various devices.

The file contains text descriptions of specific devices and their settings along with numeric data describing how to transform the color values which are to be displayed or printed on the device. The numeric data includes matrices and tables that a color management module (CMM) uses to convert that device's color results to a common color space, defined by the ICC and called the profile connection space (PCS), and back to the device's color space.

The device descriptions aid the profile user in determining the precise setup of the device for which the numeric data applies. For example, a scanner may have RGB or XYZ settings describing the data output from the scanner. A printer's description would include the type of paper (media) being print on. The color intended for different media will vary and the numeric data will reflect those variances.

The numeric data describes the conversion between the device's color and a common color space (PCS) so that profiles may be easily linked together to provide

conversions from one device to another, or through another device to a third. Without the intermediate PCS, one would need a separate profile for data converted from, for instance, a scanner to each of many possible printers. Using the intermediate PCS, data may be converted from a scanner's RGB to an intermediate CIELAB space. The data can then be taken, via a printer profile, from the CIELAB space to the printer's color space - perhaps CMYK. Figure 1, "Color Management Problem Description," on page 8 depicts the various devices whose profiles could be linked together to accomplish these conversions.

The ICC has currently defined 2 color spaces as intermediate PCS's - CIELAB and CIEXYZ. It is expected and assumed that the CMM will do any necessary XYZ to LAB or LAB to XYZ conversions itself in the case of a PCS mismatch.

There are assumptions about the device's conversion to and from a PCS which must be accounted for in the numeric data in the profile. The white point of the device may need to be converted to the white point of the ICC PCS (D50), for example. This additional conversion must be folded into the numeric data within the profile. Other considerations include various viewing conditions and gamut mapping differences between the device and the PCS.

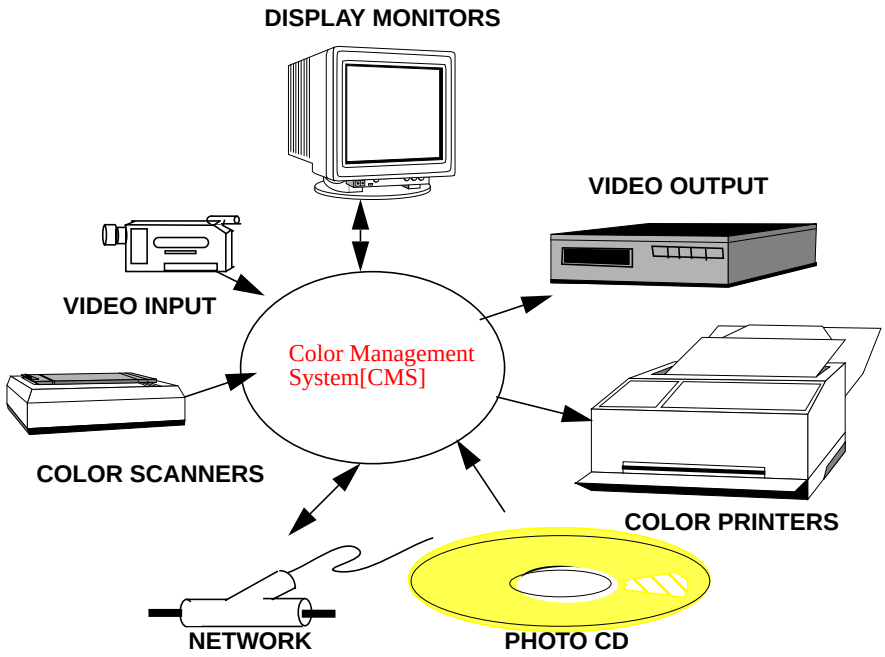


FIGURE 1. Color Management Problem Description

Types of profiles and their use

So far, only device profiles have been mentioned. There are other types of profiles which act on the image data to be processed. Colorspace profiles can convert data between one colorspace and another if the goal is to store or transport data in a certain format. Another type, the device link profile, is the combination of 2 or more profiles. This type can be produced once and used for numerous sets of data. For example, a number of images, produced on the same scanner and destined for the same printer, could share the same device link profile without the overhead of recre-

ating it for each image. The device link profile is the result of combining 2 or more profiles and cannot be further linked with additional profiles by current color management systems.

An abstract profile could be an individual or common color enhancement or affect one would want induced on one or more images. Perhaps a hazy blue tint to effect a desired artistic result. This profile can be inserted between device profiles and become one conversion in the resulting device link profile.

A named color profile allows data which is described in Pantone colors to be converted for viewing or printing, etc. Each Pantone color in the profile is accompanied by its associated numeric CIE color description.

An example of the required data to create each of these profiles is presented in this book. All of the profile types use the concept of providing conversion to and from the intermediate colorspaces (PCS) so that profiles may be linked together. The ICC specification is very clear on what tags are required for each profile type.

The ICC has tried to anticipate the types of conversions that a user will desire. A scanner to scanner conversion can be possible, but not terribly useful, for instance. Certain cases are not sufficiently covered or accommodated by the current ICC profile format. The format is expandable and proposals for expansion are discussed within the ICC. Backwards compatibility is a goal as the specification is enhanced.

Basic file structure

The profile file structure is shown in Figure 2, “Profile File Structure,” on page 10. There is a required 128 byte header followed by predefined tags (data identifiers), each with a set structure consisting of the number of bytes for the tag’s data and a pointer into the file where that data is located. The file is byte-based and big-endian (see APPENDIX A for an explanation of big-endian numbers).

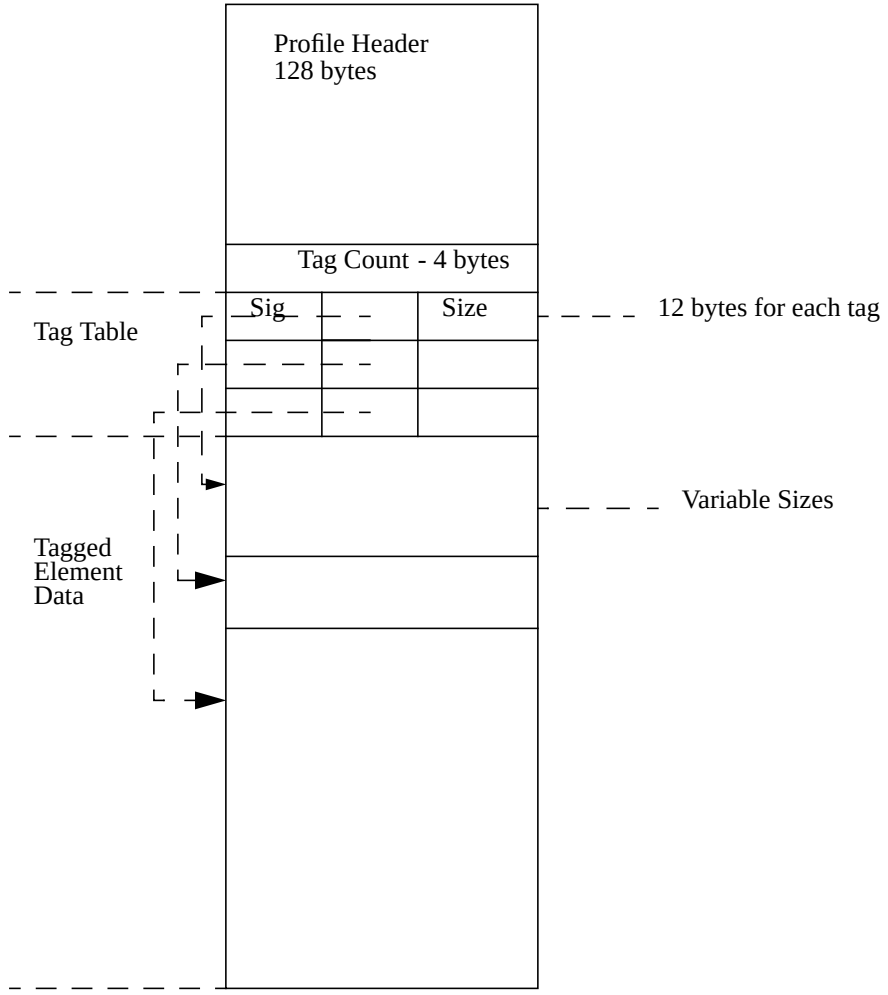


FIGURE 2. Profile File Structure

The bytes following the header and tag descriptions contain the actual profile data. As you will see, the profiles can vary in size dramatically based on the tags used and the amount of the numeric data included to maintain accuracy through the various color transformations.

The file structure is expandable, allows for tags specific to a company (called private tags), and allows properly written software to pass over tags which are not required for its purpose.

Data types and encodings

There are a number of data types used in the ICC profile specification to describe the data. The specification does a very good job of defining these types, but the next chapter will take those descriptions a step further and provide cross-reference links between tags and the data types.

CHAPTER 3

Tags, Datatypes and Encodings Cross Reference

There are a number of data types used in the ICC profile specification to describe the data. The specification does a very good job of defining these types, but this section will take those descriptions a step further and provide additional cross-references and information. It also links the tags to one implementation via a C-based ICC header file. If a discrepancy appears between these descriptions and the current version of the ICC specification, the specification takes precedence.

Sample code segments use the sample header file (icc.h), which is provided in the appendix and in an appendix of the ICC specification. The listing begins with the ICC header fields and proceeds with the tags in alphabetical order. Information is included about each tag's name and type in both the ICC specification and in the icc.h file, about what profile types require the tag, and about where to find more information about the tag in the ICC spec. Sample C code for reading and writing the tag follows each tag description. This sample code is the same code used to create most of the profiles, which are dissected in subsequent chapters. A set of declarations, used by all the sample code, is included just once, in the header description section.

The base data types used (for example, s15Fixed16Number) are thoroughly described in Appendix A, with sample code to covert into and out of these formats. The icc.h file also defines them in C language.. The sample code used in reading the tag has example text output which can be found in the chapters on dissecting profiles.

The sample code for reading tags assumes that the profile has been opened:

```
if ((fd = fopen(argv[2], "rb")) == NULL) {
   perror("Bad file name");
   exit(1);
}
```

The header is read first, followed by the tag directory. Each tag is then an entry in a “case” statement, keying off its signature. The tag directory includes the size and offset of each tag in the profile. The programs then use *lseek* to get to the tag data for reading.

The tag writing code assumes that the profile has been opened for writing:

```
if ((fd = fopen(argv[1], "wb+")) == NULL) {
  perror("Unable to create file");
  exit(1);
}
```

The size of the profile being written has already been determined and the pointers to the tag directory and tag data are set - see the original C programs.

Fixed Length Tags

Tags come in fixed or variable lengths. The fixed length tag is always a known size. The size normally ensures that the tag data ends up on a 4-byte boundary, negating the need to check whether or not padded bytes must be written to the profile prior to writing the next tag. It is, however, a good practice to always check if padding is needed in case a tag definition has fields added to it in the future.

Variable Length Tags

A variable length tag contains a variable number of bytes. The tag directory indicates how many total bytes in the tag and fields with the tag data indicate the size of variable fields within the the tag. Tags containing the “text description” type fields are always variable length. Other tag fields that tend to be variable are arrays of

numbers describing curves or lookup tables. In writing these tags, one must always check to see if padding bytes are needed to assure that any following tag starts on a 4-byte boundary.

Header Description and Cross Reference

The header is a required, 128-byte, description of the profile. It can be used to programmatically prune a list of available profiles of interest. One can easily accept or reject only certain profiles based on header signatures. For example, one may prune profiles based upon version number, device class, manufacturer, colorspace, cmm type, etc. The signatures, which are well defined and stable, are included in the icc.h file. There is a registry process, set up by the ICC, to allow companies to register unique manufacturer and model signatures. This is a rather dynamic list and may be downloaded from the icc web site. Tags introduced and used only by one company, “Private tags”, may be registered with the ICC to allow other companies’ color management systems to use them. The private tags are not part of the specification and their use is generally discouraged.

Header fields should be set to 0 if not used. It is recommended to use all the header fields to provide the most complete information about a profile.

Header:

Specification Tag name	NA (ICC spec does not specify a header tag/header signature)
Specification Tag Type	NA
ICC Header Tag Name	icSigHeaderTag
ICC Header Tag Type	icHeader
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.1
Description	ICC profile header - 128 bytes, last 44 bytes reserved

Header Contents:

Specification Field Description	Profile size
Specification Field Type	uInt32Number
ICC Header Field Name	size
ICC Header Field Type	uInt32Number
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.1.1
Description	Size of profile in bytes

Specification Field Description	CMM type
Specification Field Type	signature
ICC Header Field Name	cmmId
ICC Header Field Type	icSignature
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.1.2
Description	Signature of registered, preferred CMM

Specification Field Description	ProfileVersion Number
Specification Field Type	4 8-bit bytes, first two used
ICC Header Field Name	version
ICC Header Field Type	icUInt32Number
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	All

ICC Spec section / more info pages	ICC spec: 6.1.3
Description	Major and minor profile version number in first 2 bytes

Specification Field Description	Profile/Devise class
Specification Field Type	signature
ICC Header Field Name	deviceClass
ICC Header Field Type	icProfileClassSignature
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.1.4
Description	Input, display, output, link, colorspace, abstract or named color profile class signature

Specification Field Description	Color space of data
Specification Field Type	signature
ICC Header Field Name	colorSpace
ICC Header Field Type	icColorSpaceSignature
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.1.5
Description	registered color space signature

Specification Field Description	Profile connection space
Specification Field Type	signature
ICC Header Field Name	pcs
ICC Header Field Type	icColorSpaceSignature
Signature	NA

Fixed or Variable Length?	Fixed
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.1.6
Description	XYZ or Lab color space signature

Specification Field Description	date and time profile was first created
Specification Field Type	dateTimeNumber
ICC Header Field Name	date
ICC Header Field Type	icDateTimeNumber
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: dateTimeType, 6.5.4
Description	date and time of profile creation

Specification Field Description	‘acsp’ profile file signature
Specification Field Type	signature
ICC Header Field Name	magic
ICC Header Field Type	icSignature
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.1
Description	“magic” number identifier - literally ‘acsp’, 61637370h

Specification Field Description	Primary platform target for the profile
Specification Field Type	signature
ICC Header Field Name	platform
ICC Header Field Type	icPlatformSignature

Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.1.7
Description	Signature indicating the primary platform/operating system framework for which the profile was created.

Specification Field Description	flags to indicated various options for the CMM such as distributed process and caching option
Specification Field Type	bit fields
ICC Header Field Name	flags
ICC Header Field Type	icUInt32Number
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.1.8
Description	A bit indicating an embedded profile and a bit to indicate an embedded profile's allowed use

Specification Field Description	Device manufacturer
Specification Field Type	signature
ICC Header Field Name	manufacturer
ICC Header Field Type	icSignature
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.1.9
Description	Device manufacturer of the device for which the profile is created

Specification Field Description	Device model
Specification Field Type	signature
ICC Header Field Name	model
ICC Header Field Type	icUInt32Number
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.1.9
Description	Device model of the device for which this profile is created

Specification Field Description	Device attributes unique to a particular device setup such as media type
Specification Field Type	64 bit word with the least significant 32 bits reserved for the ICC
ICC Header Field Name	attributes
ICC Header Field Type	icUInt64Number
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.1.10
Description	A bit for reflective/transparency, glossy/matte, positive/negative, color/b&w

Specification Field Description	Rendering Intent
Specification Field Type	32 bit int but only the first 16 bits are used - least significant 16 bits are reserved for the ICC
ICC Header Field Name	renderingIntent
ICC Header Field Type	icUInt32Number
Signature	NA
Fixed or Variable Length?	Fixed

Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.1.11
Description	Perceptual, relative colorimeter, saturation, or absolute colorimetric rendering intent

Specification Field Description	XYZ values of illuminant
Specification Field Type	XYZNumber
ICC Header Field Name	illuminant
ICC Header Field Type	icXYZNumber
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.1
Description	The XYZ values of the illuminant of the profile connection space. Must correspond to D50.

Specification Field Description	Profile creator
Specification Field Type	signature
ICC Header Field Name	creator
ICC Header Field Type	icSignature
Signature	NA
Fixed or Variable Length?	Fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.1.12
Description	Signature from the manufacturer signature list identifying the creator of the profile

Header Sample Code

Each of the sample profile creation programs include creation of a header. The variable declarations common to each program are listed once, here. The additional functions declared here are also part of each program and are provided at the end of this chapter.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#ifdef _WIN32
#define _LITTLE_ENDIAN
#else
#include <sys/isa_defs.h>
#include <sys/ddi.h>
#endif
#include <time.h>
#include <math.h>
#include "icc.h"
double  icfixed2double(long val, long type);
double  icfixed2gamma(icUInt16Number);
long    double2icfixed(double val, long type);
#ifdef _WIN32
typedef unsigned short ushort;
#endif

void    print_luttag(FILE *fd,  icTag tag, icUInt32Number verbose, char buf[5]);

   icInt32Number      readfile, writefile, verbose, rc, ri, gi, bi;
   char               buf[100][5], charstring[5];
   icInt32Number      i, ii, ii2, j, jj, jj2, channelvals[10], dumpfile, size, dirptr,
                      tagdataptr, tempdataptr, perrow;
   FILE               *fd;
   float              f32;
   icUInt32Number     n32;
```

```
icUInt16Number     n16;
icUInt8Number      n8;
icUInt16Number     val;
icUInt32Number     ntags;
icTag              tag[100];
long               swap(long);
ushort             swap16(ushort);
icSignature        sig;
icUInt64Number     tmp64;
icUInt32Number     count;

icHeader           header, *hdr;
icXYZType          xyz;
icCurveType        *curvealloc;
icCurve            *curve;
icDataType         *datatalloc;
icData             *datat;
icDateTimeType     datetime;
icTextDescriptionType      *textdescalloc;
icTextDescription  *textdesc;
icTextType         *textalloc;
icText             *text;
icLut16Type        *lut16alloc;
icLut16            *lut16;
icLut8Type         *lut8alloc;
icLut8             *lut8;
icMeasurementType          measurement;
icProfileSequenceDescType  *profseqdescalloc;
icProfileSequenceDesc      *profseqdesc;
icInt8Number       *dataptr;
icSignatureType    tech;
icCrdInfoType      *crdalloc;
icCrdInfo          *crd;
icScreeningType    *screenalloc;
icScreening        *screen;
```

```
icScreeningData     *scptr;
icUcrBgType         *ucrbgalloc;
icUcrBg             *ucrbg;
icViewingConditionType          view;
icNamedColor2Type   *nc2alloc;
icNamedColor2       *nc2;
icChromaticityType  *chromaalloc;
icChromaticity      *chroma;
char                *privatealloc;
icUInt32Number      *private;
time_t              clocktime;
struct tm           *tmdatetime;
icTag               tagdir;
icUInt32Number      tblenum, lutnum;
icTagBase           tagbase;
div_t               divt;
icInt8Number        *ptr8, *ptr8save;
icDescStruct        *descstructalloc, *descstruct;
icDeviceSettingsType *settingstype;
icUInt32Number      ptr32;
icSettingsData      *settingsdata;
icResponseCurveSet16Type        *responsetype;
icResponse          *responsedata;
icResponseCurveSet16Type        *respcurvetype;
icResponse          *response;
```

Sample code to read a header:

Code to read the header includes a function called swap, which is an example method of handling the difference between data on “big-endian” and “little-endian” computers. Please see Appendix A.6 for more information on this subject, including the swap and swap16 function code. ICC profiles are stored “big-endian”.

```
if (fread(&header, sizeof(icHeader), 1, fd) != 1) {
    printf("error reading file\n");
```

```
    exit(1);
  }
 printf("Size in bytes = %d\n", swap((long)header.size));
 memcpy(charstring, &header.cmmId, 4);
 printf("CMM Id = %.4s\n", charstring);
 printf("Version number = 0x%x\n", swap((long)header.version));

 switch(swap((long)header.deviceClass)) {
 case icSigInputClass :
  printf("deviceClass = input\n");
  break;
 case icSigDisplayClass :
  printf("deviceClass = display\n");
  break;
 case icSigOutputClass :
  printf("deviceClass = output\n");
  break;
 case icSigLinkClass :
  printf("deviceClass = link\n");
  break;
 case icSigAbstractClass :
  printf("deviceClass = abstract\n");
  break;
 case icSigColorSpaceClass :
  printf("deviceClass = colorspace\n");
  break;
 default :
  printf("Unknown\n");
  break;

}
```

```
memcpy(charstring, &header.colorSpace, 4);
printf("colorspace = %.4s\n", charstring);
memcpy(charstring, &header.pcs, 4);
printf("profile connection space = %.4s\n", charstring);
printf("date = %d/%d/%d,  ",swap16((ushort)header.date.day),
  swap16((ushort)header.date.month), swap16((ushort)header.date.year));
printf("time = %d:%d:%d\n",swap16((ushort)header.date.hours),
  swap16((ushort)header.date.minutes),swap16((ushort)header.date.seconds));

memcpy(charstring, &header.magic, 4);
printf("magic number = %.4s\n", charstring);

switch(swap((long)header.platform)) {
case icSigMacintosh :
  printf("platform = Macintosh\n");
  break;
case icSigMicrosoft :
  printf("platform = Microsoft\n");
  break;
case icSigSolaris :
  printf("platform = Solaris\n");
  break;
case icSigSGI :
  printf("platform = SGI\n");
  break;
case icSigTaligent :
  printf("platform = Taligent\n");
  break;
default :
  printf("Unknown\n");
```

```
      break;
    }

    if(swap((long)header.flags) && icEmbeddedProfileTrue)
      printf("Embedded profile.\n");
    else
      printf("Non-embedded profile\n");

    if(swap((long)header.flags) && i(long)header.flags) && icEmbeddedProfi
leTrue)
      printf("Embedded profile.\n");
    else
      printf("Non-embedded profile\n");

    if(swap((long)header.flags) && icUseWithEmbeddedDataOnly) {
      printf("If this profile is embedded, ");
      printf("it is not allowed to strip it out");
      printf(" and use it independently.\n");
    } else
      printf("OK to strip embedded profile out and use independently\n");

    memcpy(charstring, &header.manufacturer, 4);
    printf("manufacturer = %.4s\n", charstring);

    memcpy(charstring, &header.model, 4);
    printf("model = %.4s\n", charstring);

    if ((swap((long)header.attributes[0]) & 0x00000001) == 1)
      printf("Attributes = transparency, ");
    else
      printf("Attributes = reflective, ");
```

```
if ((swap((long)header.attributes[0]) & 0x00000002) == 2)
  printf("matte, ");
else
  printf("glossy, ");
if ((swap((long)header.attributes[0]) & 0x00000004) == 4)
  printf("negative, ");
else
  printf("positive, ");
if ((swap((long)header.attributes[0]) & 0x00000008) == 8)
  printf("black & white\n");
else
  printf("color\n");
header.renderingIntent =
  (swap((long)header.renderingIntent) & 0x00030000);
switch (header.renderingIntent) {
case 0:
  printf("rendering intent = Perceptual\n");
  break;
case 65536:
  printf("rendering intent = Relative Colorimetric\n");
  break;
case 131072:
  printf("rendering intent = Saturation\n");
  break;
case 196608:
  printf("rendering intent = Absolute Colorimetric\n");
  break;
default :
  printf("Unknown\n");
  break;
}
```

```
printf("Illuminat X=%f Y=%f Z=%f\n",
        icfixed2double(header.illuminant.X, icSigS15Fixed16ArrayType),
        icfixed2double(header.illuminant.Y, icSigS15Fixed16ArrayType),
        icfixed2double(header.illuminant.Z, icSigS15Fixed16ArrayType));
memcpy(charstring, &header.creator, 4);
printf("creator = %.4s\n", charstring);
printf("\n\n");
```

Sample code to write a header:

```
hdr = (icHeader *)calloc(sizeof(icHeader), sizeof(icUInt8Number));
hdr->size = swap((size + 4)); /*update with final profile size*/
hdr->cmmId = swap((long)0x4b434d53L);          /*'KCMS'*/
hdr->version = swap((long)0x21000000L);
hdr->deviceClass = swap((long)0x6d6e7472L);    /*'mntr'*/
hdr->colorSpace = swap((long)0x52474220L);     /*'RGB '*/
hdr->pcs = swap((long)0x4c616220L);            /*'Lab '*/
hdr->cmmId = swap((long)0x4b434d53L);          /*'KCMS'*/
hdr->version= swap((long)0x21000000L);
hdr->deviceClass = swap((long)0x6d6e7472L);    /*'mntr'*/
hdr->colorSpace = swap((long)0x52474220L);     /*'RGB '*/
hdr->pcs = swap((long)0x4c616220L);            /*'Lab '*/
/* Get the time from the system */
clocktime = time(NULL);
tmdatetime = localtime(&clocktime);

hdr->date.seconds = (icUInt16Number)swap16((ushort)tmdatetime->tm_sec);
hdr->date.minutes = (icUInt16Number)swap16((ushort)tmdatetime->tm_min);
hdr->date.hours = (icUInt16Number)swap16((ushort)tmdatetime->tm_hour);
hdr->date.day = (icUInt16Number)swap16((ushort)tmdatetime->tm_mday);
1));hdr->date.month = (icUInt16Number)swap16((ushort)(tmdatetime->tm_mon +
```

```
    hdr->date.year = (icUInt16Number)swap16((ushort)(tmdatetime->tm_year +
 1900)
);

    hdr->magic = swap((long)0x61637370L);          /*'acsp'*/
    hdr->platform = swap((long)0x53554e57L);       /*'SUNW'*/
    hdr->flags = swap((long)0x00000000L);          /*Profile not embedded and
                                    can be used independently*
/
    /*maufacturer and model should be registered with the ICC*/
    hdr->manufacturer = swap((long)0x53554e20L);   /*'SUN '*/
    hdr->model = swap((long)0x31393938L);          /*'1998'*/
    hdr->attributes[0] = swap((long)9);    /*Transparency, glossy, positive,
B&W*/
    hdr->attributes[1] = swap((long)0);
    hdr->renderingIntent = swap((long)1 << 16);          /*relative colorimetric*/
    hdr->illuminant.X = double2icfixed((double)0.964188,
 icSigS15Fixed16ArrayTyp
e);
    hdr->illuminant.Y = double2icfixed((double)1.000,
 icSigS15Fixed16ArrayType);
    hdr->illuminant.Z = double2icfixed((double)0.824890,
 icSigS15Fixed16ArrayTyp
e);
    hdr->creator = swap((long)0x53554e57L);        /*'SUNW'*/
    if (fwrite(hdr, sizeof(icHeader), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    fseek(fd, sizeof(icHeader), SEEK_SET);
    ntags = swap((long)ntags);
```

```
if (fwrite(&ntags, sizeof(icUInt32Number), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
dirptr =  sizeof(icHeader) + 4;          /*point to start of tag directory*/
free(hdr);
```

AToB0Tag

Specification Tag Name	AToB0Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icSigAToB0Tag
ICC Header Tag Type	icLut8 or icLut16
Signature	A2B0 (41324230h)
Fixed or Variable Length?	Variable
Required by Profile Types	N-component LUT-based input profiles RGB and CYMK output profiles DeviceLink profiles Colorspace conversion profiles Abstract profiles
ICC Spec section / more info pages	ICC spec: 6.4.1
Description	Multidimensional transformation structure; Device space to PCS, perceptual intent

Sample code to read the AToB0Tag:

There are a number of tags which are identical in structure so the code to read them is also identical. This code will apply to the following additional tags: BToA0Tag, AToB1Tag, BToA1Tag, AToB2Tag, BToA2Tag, gamutTag, preview0Tag, preview1Tag, preview2Tag.

Since the tag type can be either 8 bit or 16 bit, meaning the tables consist of either 8 bit or 16 bit data, this code has the capability of recognizing and printing either.

```
void print_luttag(FILE *fd, icTag tag,
                  icUInt32Number verbose, char buf[5])
{
   icLut16Type        *lut16alloc;
   icLut16            *lut16;
   icLut8Type         *lut8alloc;
   icLut8             *lut8;
   long               swap(long);
   ushort             swap16(ushort);
   icSignature        sig;
   icInt32Number      i, ii, ii2, j, jj, jj2,  size, perrow, dirptr;
   div_t              divt;


      printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
            tag.sig, buf, tag.offset, tag.size);
      fseek(fd, (long) tag.offset, 0);
      if (fread(&sig, 4, 1, fd) != 1) {
        printf("error reading file\n");
        exit(1);
      }
      sig = swap(sig);
      fseek(fd, (long) tag.offset, 0);       /*reset */
      if (sig == 0x6d667431L) {              /* mft1 = 8 bit lut */
      printf("8 bit lut type\n");
      lut8alloc = (icLut8Type *)calloc(tag.size, sizeof(icUInt8Number));
      if (fread(lut8alloc, tag.size, 1, fd) != 1) {
        printf("error reading file\n");
        exit(1);
      }
```

```
lut8 = &lut8alloc->lut;
printf("Lut 8 type\n");
printf("%d  ", (short)lut8->inputChan);
printf("%d  ", (short)lut8->outputChan);
printf("%d\n", (short)lut8->clutPoints);
printf("%f ", icfixed2double((long)lut8->e00, icSigS15Fixed16ArrayType));
printf("%f ", icfixed2double((long)lut8->e01, icSigS15Fixed16ArrayType));
printf("%f\n", icfixed2double((long)lut8->e02, icSigS15Fixed16ArrayType));
printf("%f ", icfixed2double((long)lut8->e10, icSigS15Fixed16ArrayType));
printf("%f ", icfixed2double((long)lut8->e11, icSigS15Fixed16ArrayType));
printf("%f\n", icfixed2double((long)lut8->e12, icSigS15Fixed16ArrayType));
printf("%f ", icfixed2double((long)lut8->e20, icSigS15Fixed16ArrayType));
printf("%f ", icfixed2double((long)lut8->e21, icSigS15Fixed16ArrayType));
printf("%f\n", icfixed2double((long)lut8->e22, icSigS15Fixed16ArrayType));

if (!verbose) printf("tables will not be dumped unless you use -v option\n");

if(verbose) {
/*print the input table*/
j = lut8->inputChan * 256;
printf("input tables: #channels * 256 = %d\n", j);
perrow = 9;
dirptr = 0;
divt = div(j, perrow);
for (ii=0; ii<divt.quot; ii++) {
  for (ii2 =0; ii2<perrow; ii2++) {
     printf("%d  ",lut8->data[dirptr + (ii*perrow) + ii2]);
  }
  printf("\n");
}
for (ii2 =j - divt.rem; ii2<j; ii2++) {
```

```
  printf("%d  ",lut8->data[ii2]);
}
printf("\n");
dirptr += j;

/*print the clut*/
printf("\n");
j = (pow(lut8->clutPoints,lut8->inputChan)
        *lut8->outputChan) ;
printf("clut: #clut points**#input channels * #output channels = %d\n",
      j);
divt = div(j, perrow);
for (ii=0; ii<divt.quot; ii++) {
  for (ii2 =0; ii2<perrow; ii2++) {
    printf("%d  ",lut8->data[dirptr + (ii*perrow) + ii2]);
  }
  printf("\n");
}
for (ii2 =dirptr + j - divt.rem; ii2<dirptr + j; ii2++) {
  printf("%d  ",lut8->data[ii2]);
}
printf("\n");
dirptr += j;

/*print the output table*/
j = (lut8->outputChan * 256);
printf("output tables: #channels * 256 = %d\n", j);

divt = div(j, perrow);
for (ii=0; ii<divt.quot; ii++) {
  for (ii2 =0; ii2<perrow; ii2++) {
```

```
      printf("%d  ",lut8->data[dirptr + (ii*perrow) + ii2]);
    }
    printf("\n");
  }
  for (ii2 =dirptr +j - divt.rem; ii2<dirptr + j; ii2++) {
    printf("%d  ",lut8->data[ii2]);
  }
  printf("\n");
  dirptr += j;

  }  /*end if verbose*/

  free(lut8alloc);

  } else
  if (sig == 0x6d667432L) {              /* mft2 = 16 bit lut */
  printf("16 bit lut type\n");
  lut16alloc = (icLut16Type *)calloc(tag.size, sizeof(icUInt8Number));
  if (fread(lut16alloc, tag.size, 1, fd) != 1) {
    printf("error reading file\n");
    exit(1);
  }
  lut16 = &lut16alloc->lut;
  printf("Lut 16 type\n");
  printf("%d  ", (short)lut16->inputChan);
  printf("%d  ", (short)lut16->outputChan);
  printf("%d\n", (short)lut16->clutPoints);
  printf("%f ", icfixed2double((long)lut16->e00, icSigS15Fixed16ArrayType));
  printf("%f ", icfixed2double((long)lut16->e01, icSigS15Fixed16ArrayType));
  printf("%f\n", icfixed2double((long)lut16->e02, icSigS15Fixed16ArrayType));
  printf("%f ", icfixed2double((long)lut16->e10, icSigS15Fixed16ArrayType));
```

```
 printf("%f ", icfixed2double((long)lut16->e11, icSigS15Fixed16ArrayType));
printf("%f\n", icfixed2double((long)lut16->e12, icSigS15Fixed16ArrayType));
 printf("%f ", icfixed2double((long)lut16->e20, icSigS15Fixed16ArrayType));
 printf("%f ", icfixed2double((long)lut16->e21, icSigS15Fixed16ArrayType));
printf("%f\n", icfixed2double((long)lut16->e22, icSigS15Fixed16ArrayType));
 printf("%d   %d \n", swap16((ushort)lut16->inputEnt),
        swap16((ushort)lut16->outputEnt));

 if (!verbose) printf("tables will not be dumped unless you use -v option\n");

 if(verbose) {
 /*print the input table*/
 j = lut16->inputChan * swap16((ushort)lut16->inputEnt);
 printf("input tables: #channels * #input entries = %d\n", j);
 perrow = 9;
 dirptr = 0;
 divt = div(j, perrow);
 for (ii=0; ii<divt.quot; ii++) {
   for (ii2 =0; ii2<perrow; ii2++) {
      printf("%d  ",swap16((ushort)lut16->data[dirptr + (ii*perrow) + ii2]));
   }
   printf("\n");
 }
 for (ii2 =j - divt.rem; ii2<j; ii2++) {
   printf("%d  ",lut16->data[ii2]);
 }
 printf("\n");
 dirptr += j;

 /*print the clut*/
 printf("\n");
```

```
j = (pow(lut16->clutPoints,lut16->inputChan)
        *lut16->outputChan) ;
printf("clut: #clut points**#input channels * #output channels = %d\n",
      j);
divt = div(j, perrow);
for (ii=0; ii<divt.quot; ii++) {
  for (ii2 =0; ii2<perrow; ii2++) {
    printf("%d  ",lut16->data[dirptr + (ii*perrow) + ii2]);
  }
  printf("\n");
}
for (ii2 =dirptr + j - divt.rem; ii2<dirptr + j; ii2++) {
  printf("%d  ",lut16->data[ii2]);
}
printf("\n");
dirptr += j;

/*print the output table*/
j = (swap16((ushort)lut16->outputEnt)*lut16->outputChan);
printf("output tables: #channels * #output entries = %d\n", j);

divt = div(j, perrow);
for (ii=0; ii<divt.quot; ii++) {
  for (ii2 =0; ii2<perrow; ii2++) {
    printf("%d  ",swap16((ushort)lut16->data[dirptr + (ii*perrow) + ii2]));
  }
  printf("\n");
}
for (ii2 =dirptr +j - divt.rem; ii2<dirptr + j; ii2++) {
  printf("%d  ",swap16((ushort)lut16->data[ii2]));
}
```

```
    printf("\n");
    dirptr += j;

    }  /*end if verbose*/

    free(lut16alloc);
    }
    printf("\n\n");

}
```

Sample code to write the AToB0Tag:

Again, the code to write these identical tags is usually very similar. One difference is the differnce between writing an 8 bit and a 16 bit tag. The number of elements in the tables is completely up to the profile builder as is the algorithms/methods of obtaining and massaging the data for these tags.

This example creates a 16-bit identity AToB0Tag - essentially a null effect.

```
 tagdir.sig = swap((long)0x41324230L);     /*'A2B0'*/
tagdir.offset = swap((long)tagdataptr);
tblenum = 6;                          /*input/ouptut tables will
                                    contain 3X2 values each*/
lutnum = 1536;                        /*lut will be 8X8X8 X3chanels */
size = 52 +                          /*tag base + pre-table data*/
  (4 * tblenum) + (2*lutnum) + (4*tblenum);
printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
```

```
    exit(1);
  }
dirptr += sizeof(tagdir);                /*keep this pointed to
                                  the end of the tag directory*/
/*create a null lut 16 tag for testing purposes*/
tagbase.sig = swap((long)0x6d667432L);    /*'mft2'*/
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
tagdataptr += sizeof(tagbase);
lut16alloc = (icLut16Type *)calloc(size, sizeof(icUInt8Number));
lut16 = &lut16alloc->lut;
lut16->inputChan = (icUInt8Number)3;
lut16->outputChan = (icUInt8Number)3;
lut16->clutPoints = (icUInt8Number)8;      /*per-side count*/
lut16->pad = (icInt8Number)0;

/*create an identity matrix*/
lut16->e00 =  double2icfixed(1.000, icSigS15Fixed16ArrayType);
lut16->e01 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
lut16->e02 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
lut16->e10 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
lut16->e11 =  double2icfixed(1.000, icSigS15Fixed16ArrayType);
lut16->e12 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
lut16->e20 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
lut16->e21 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
lut16->e22 =  double2icfixed(1.000, icSigS15Fixed16ArrayType);

lut16->inputEnt = (icUInt16Number)swap16((ushort)2);/*will be 2X#chan total
```

```
entries*/
    lut16->outputEnt = (icUInt16Number)swap16((ushort)2);
    /*input tables -  values supplied in the uInt16Number range  */
    i = 0;
    lut16->data[i++] = (icUInt16Number)swap16((ushort)0);
    lut16->data[i++] = (icUInt16Number)swap16((ushort)65535);
    lut16->data[i++] = (icUInt16Number)swap16((ushort)0);
    lut16->data[i++] = (icUInt16Number)swap16((ushort)65535);
    lut16->data[i++] = (icUInt16Number)swap16((ushort)0);
    lut16->data[i++] = (icUInt16Number)swap16((ushort)65535);
    /*clut table*/
    for (ri = 0; ri< 256; ri+=32)
     {
       for (gi = 0; gi < 256; gi+=32)
         {
           for (bi = 0; bi < 256; bi+=32)
             {
               lut16->data[i++] = (icUInt16Number)swap16((ushort)(min (max (0,
(int)
                                                   (ri * 255)), 65535
)));
               lut16->data[i++] = (icUInt16Number)swap16((ushort)(min (max (0,
(int)
                                                   (gi * 255)), 65535
)));
               lut16->data[i++] = (icUInt16Number)swap16((ushort)(min (max (0,
(int)
                                                   (bi * 255)), 65535
)));
             }
         }
```

```
  }
/*output tables */
lut16->data[i++] = (icUInt16Number)swap16((ushort)0);
lut16->data[i++] = (icUInt16Number)swap16((ushort)65535);
lut16->data[i++] = (icUInt16Number)swap16((ushort)0);
lut16->data[i++] = (icUInt16Number)swap16((ushort)65535);
lut16->data[i++] = (icUInt16Number)swap16((ushort)0);
lut16->data[i++] = (icUInt16Number)swap16((ushort)65535);
if (fwrite(lut16, size - sizeof(tagbase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
tagdataptr += size - sizeof(tagbase);
/*update the final size of profile in the header*/
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
free(lut16alloc);
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

AToB1Tag

Specification Tag Name	AToB1Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icSigAToB1Tag
ICC Header Tag Type	icLut8 or icLut16
Signature	A2B1 (41324231h)
Fixed or Variable Length?	variable
Required by Profile Types	RGB and CYMK output profiles
ICC Spec section / more info pages	ICC spec: 6.4.2
Description	Multidimensional transformation structure; Device space to PCS, relative colorimetric intent

Sample code to write the AToB1Tag:

Please see AToB0Tag code sample for reading the tag.

This sample code uses the same data for writing a lut tag as was used for the 16 bit tag example, but this is for the 8 bit version of the tag. Note that two fields, the number of input and output table entries, are not part of the 8bit tag. The number of table entries is set at 256 for each.

```
 tagdir.sig = swap((long)0x41324231L);     /*'A2B1'*/
tagdir.offset = swap((long)tagdataptr);
tblenum = 3 * 256;                    /*input/ouptut tables will
                                  contain 256 values each*/
lutnum = 1536;                        /*lut will be 8X8X8 X3chanels */
size = 48 +                          /*tag base + pre-table data*/
  (tblenum) + (lutnum) + (tblenum);       /*byte data*/
printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
```

```
        printf("error writing file\n");
        exit(1);
      }
    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
the
                                    tag directory*/
    /*create a null lut 8 tag for testing purposes*/
    tagbase.sig = swap((long)0x6d667431L);    /*'mft1'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    tagdataptr += sizeof(tagbase);
    lut8alloc = (icLut8Type *)calloc(size, sizeof(icUInt8Number));
    lut8 = &lut8alloc->lut;
    lut8->inputChan = (icUInt8Number)3;
    lut8->outputChan = (icUInt8Number)3;
    lut8->clutPoints = (icUInt8Number)8;      /*per-side count*/
    lut8->pad = (icInt8Number)0;

    /*create an identity matrix*/
    lut8->e00 =  double2icfixed(1.000, icSigS15Fixed16ArrayType);
    lut8->e01 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
    lut8->e02 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
    lut8->e10 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
    lut8->e11 =  double2icfixed(1.000, icSigS15Fixed16ArrayType);
    lut8->e12 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
    lut8->e20 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
    lut8->e21 =  double2icfixed(0.000, icSigS15Fixed16ArrayType);
    lut8->e22 =  double2icfixed(1.000, icSigS15Fixed16ArrayType);
```

```
    /*input tables - each must be 256 values - supplied in the UInt8Number range
*/
    i = 0;
    for (j=0; j<256; j++) {
      lut8->data[i++] = (icUInt8Number)j;
    }
    for (j=0; j<256; j++) {
      lut8->data[i++] = (icUInt8Number)j;
    }
    for (j=0; j<256; j++) {
      lut8->data[i++] = (icUInt8Number)j;
    }
    /*clut*/
    for (ri = 0; ri< 256; ri+=32)
      {
        for (gi = 0; gi < 256; gi+=32)
          {
            for (bi = 0; bi < 256; bi+=32)
              {
                lut8->data[i++] = (icUInt8Number)(min (max (0, (int)
                                                    (ri * 255)), 65535));
                lut8->data[i++] = (icUInt8Number)(min (max (0, (int)
                                                    (gi * 255)), 65535));
                lut8->data[i++] = (icUInt8Number)(min (max (0, (int)
                                                    (bi * 255)), 65535));
              }
          }
      }
    /*output tables - each must be 256 values*/
    for (j=0; j<256; j++) {
```

```
  lut8->data[i++] = (icUInt8Number)j;
}
for (j=0; j<256; j++) {
  lut8->data[i++] = (icUInt8Number)j;
}
for (j=0; j<256; j++) {
  lut8->data[i++] = (icUInt8Number)j;
}

if (fwrite(lut8, size - sizeof(tagbase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
tagdataptr += size - sizeof(tagbase);
/*update the final size of profile in the header*/
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
free(lut8alloc);
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

AToB2Tag

Specification Tag Name	AToB2Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icSigAToB2Tag
ICC Header Tag Type	icLut8 or icLut16
Signature	ATpoB2 (41324232h)
Fixed or Variable Length?	variable
Required by Profile Types	RGB and CYMK output profiles
ICC Spec section / more info pages	ICC spec: 6.4.3
Description	Multidimensional transformation structure; Device space to PCS, saturation intent

Please see the AToB0Tag and AToB1Tag for sample code.

blueColorantTag

Specification Tag Name	blueColorantTag
Specification Tag Type	XYZType
ICC Header Tag Name	icSigBlueColorantTag
ICC Header Tag Type	icXYZType -> icXYZArray (3 icS16Fixed16Number's)
Signature	bXYZ (6258595Ah)
Fixed or Variable Length?	fixed
Required by Profile Types	3-component matrix-bases input profiles RGB display profiles
ICC Spec section / more info pages	ICC space: 6.4.4
Description	Relative XYZ values of blue phosphor or colorant

Sample code to read the blueColorantTag:

```
case 0x6258595AL:  /* 'bXYZ' */
```

```
printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
      tag[i].sig, buf[i], tag[i].offset, tag[i].size);
fseek(fd, (long) tag[i].offset, 0);
if (fread(&xyz, sizeof(icXYZType), 1, fd) != 1) {
  printf("error reading file\n");
  exit(1);
}
printf("XYZ type\n");
printf("X=%f, Y=%f, Z=%f\n",
      icfixed2double(xyz.data.data[0].X,
                  icSigS15Fixed16ArrayType),
      icfixed2double(xyz.data.data[0].Y,
                  icSigS15Fixed16ArrayType),
      icfixed2double(xyz.data.data[0].Z,
                  icSigS15Fixed16ArrayType));

printf("\n\n");
break;
```

Sample code to write the blueColorantTag:

```
tagdir.sig = swap((long)0x6258595aL);                /*'bXYZ'*/
tagdir.offset = swap((long)tagdataptr);
size = sizeof(tagbase) + sizeof(icXYZNumber);
printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
```

```
    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
the
                                    tag directory*/
    tagbase.sig = swap((long)0x58595a20L);          /*'XYZ '*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    tagdataptr +=sizeof(tagbase);
    xyz.data.data[0].X = double2icfixed (0.279984, icSigS15Fixed16ArrayType);
    xyz.data.data[0].Y = double2icfixed (0.200272, icSigS15Fixed16ArrayType);
    xyz.data.data[0].Z = double2icfixed (0.840454, icSigS15Fixed16ArrayType);
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&xyz.data, sizeof(icXYZNumber), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }

    /*update the final size of profile in the header*/
    tagdataptr += sizeof(icXYZNumber);
    fseek(fd, 0, SEEK_SET);
    tempdataptr = swap((long)tagdataptr);
    if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    /* May need add to the tagdataptr to make sure the next tag
      lands on a four byte boundary
```

```
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;

tagdir.sig = 0x6258595aL;                /*'bXYZ'*/
tagdir.offset = tagdataptr;
size = sizeof(tagbase) + sizeof(icXYZNumber);
printf("tag size = %d\n",size);
tagdir.size = size;
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }

dirptr += sizeof(tagdir);              /*keep this pointed to the end of the
              tag directory*/
tagbase.sig = 0x58595a20L;              /*'XYZ '*/
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
tagdataptr +=sizeof(tagbase);
xyz.data.data[0].X = double2icfixed (0.279984, icSigS15Fixed16ArrayType);
xyz.data.data[0].Y = double2icfixed (0.200272, icSigS15Fixed16ArrayType);
xyz.data.data[0].Z = double2icfixed (0.840454, icSigS15Fixed16ArrayType);
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&xyz.data, sizeof(icXYZNumber), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
```

```
  }

/*update the final size of profile in the header*/
tagdataptr += sizeof(icXYZNumber);
fseek(fd, 0, SEEK_SET);
if (fwrite(&tagdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

blueTRCTag

Specification Tag Name	blueTRCTag
Specification Tag Type	curveType
ICC Header Tag Name	icSigBlueTRCTag
ICC Header Tag Type	icCurveType ->icCurve (count and 16 bit values)
Signature	bTRC (62545243h)
Fixed or Variable Length?	variable
Required by Profile Types	3-component matrix-bases input profiles RGB display profiles
ICC Spec section / more info pages	ICC spec: 6.4.5
Description	Blue channel tone reproduction curve

This tag's tag type (curveType) allows data to be provided in one of several ways. When the tag's count field is 0, a linear response (slope = 1.0) is assumed. When the count is 1, then the data entry is interpreted as a simple gamma value. This example shows a truely populated curve, i.e. the count is the number of values provided for the curve. The tag reader must account for all types.

Sample code to read the blueTRCTag:

```
case 0x62545243L:  /* 'bTRC' */
 printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
       tag[i].sig, buf[i], tag[i].offset, tag[i].size);
 fseek(fd, (long) tag[i].offset, 0);
 curvealloc = (icCurveType *)calloc(tag[i].size, sizeof(icUInt8Number));
 if (fread(curvealloc,  tag[i].size, 1, fd) != 1) {
  printf("error reading file\n");
  exit(1);
 }
 curve = &curvealloc->curve;
 curve->count = swap((long)curve->count);
 printf("Curve type, curve count = %d\n",curve->count );
    /* Linear */
    if (curve->count == 0) {
       printf("Count = 0 Curve is linear\n");
     }
    /* Gamma value */
    else if (curve->count == 1) {
       printf("Count = 1 Curve is a gamma of %f\n",
            icfixed2gamma((icUInt16Number)curve->data[0]));
    }
    /* Beginning/end points of a line */
    else if (curve->count == 2) {
       printf("Count = 2 Line Start = %d End = %d\n",
```

```
            (icUInt16Number)swap16((ushort)curve->data[0]),
              (icUInt16Number)swap16((ushort)curve->data[1]));
        } else {

        /* Just a plain old curve */
        for (j=0; j<curve->count; j++) {
          printf("%d ", (icUInt16Number)swap16((ushort)curve->data[j]));
          if (!(j%16) && (j!=0))
            printf("\n");
        }
        }

      printf("\n\n");
      free(curvealloc);
        break;
```

Sample code to write the blueTRCTag (populated curve option):

There are several options for writing this tag. If there is a curve count of 1, then the value in the curve "array" is a gamma value. This example shows how to write the tag with that option. See the red and green TRC tags for other examples.

```
    /*This TRC example will use the gamma option by setting count to 1 */
      tagdir.sig = swap((long)0x62545243L);                  /*bTRC'*/
      tagdir.offset = swap((long)tagdataptr);

      size = sizeof(tagbase) + sizeof(icUInt32Number) + sizeof(icUInt16Number);
      curvealloc = (icCurveType *)calloc(size, sizeof(icUInt8Number));
      curve = &curvealloc->curve;
      printf("tag size = %d\n",size);
      tagdir.size = swap((long)size);
      fseek(fd, dirptr, SEEK_SET);
      if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
```

```
        printf("error writing file\n");
        exit(1);
      }
    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
the
                                    tag directory*/
    tagbase.sig = swap((long)0x63757276L);          /*'curv'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
      printf("error writing file\n");
      exit(1);
     }
    tagdataptr +=sizeof(icTagBase);
    curve->count = swap((long)1);
    /*Only one value, gamma, if curve count is 1 - actually a U8Fixed8Number*/
    curve->data[0] = (icUInt16Number)swap16((ushort)(2.22 * 256.0 + 0.5));
    if (fwrite(curve, size - sizeof(icTagBase), 1, fd) != 1) {
      printf("error writing file\n");
      exit(1);
     }

    /*update the final size of profile in the header*/
    tagdataptr += size - sizeof(tagbase);
    fseek(fd, 0, SEEK_SET);
    tempdataptr = swap((long)tagdataptr);
    if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
      printf("error writing file\n");
      exit(1);
     }
    free(curvealloc);
    /* May need add to the tagdataptr to make sure the next tag
```

```
  lands on a four byte boundary
  */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

BToA0Tag

Specification Tag Name	BToA0Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icBToA0Tag
ICC Header Tag Type	icLut8 or icLut16
Signature	BToA0 (42324130h)
Fixed or Variable Length?	variable
Required by Profile Types	RGB and CYMK output profiles Colorspace conversion profiles
ICC Spec section / more info pages	ICC spec: 6.4.6
Description	Multidimensional transformation structure; PCS to Device space, perceptual intent

See the AToB0Tag and AToB1Tag for sample code.

BToA1Tag

Specification Tag Name	BToA1Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icBToA1Tag

ICC Header Tag Type	icLut8 or icLut16
Signature	BToA1 (42324131h)
Fixed or Variable Length?	variable
Required by Profile Types	RGB and CYMK output profiles
ICC Spec section / more info pages	ICC spec: 6.4.7
Description	Multidimensional transformation structure; PCS to Device space, relative colorimetric intent

See the AToB0Tag and AToB1Tag for sample code.

BToA2Tag

Specification Tag Name	BToA2Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icBToA2Tag
ICC Header Tag Type	icLut8 or icLut16
Signature	BToA2 (42324132h)
Fixed or Variable Length?	variable
Required by Profile Types	RGB and CYMK output profiles
ICC Spec section / more info pages	ICC spec: 6.4.8
Description	Multidimensional transformation structure; PCS to Device space, saturation intent

See the AToB0Tag and AToB1Tag for sample code.

calibrationDateTimeTag

Specification Tag Name	calibrationDateTimeTag
Specification Tag Type	dateTimeType
ICC Header Tag Name	icSigCalibrationDateTimeTag
ICC Header Tag Type	icSigDateTimeType -> icSigDateTimeNumber (6 icUInt16Number's)
Signature	calt (63616C74h)
Fixed or Variable Length?	fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.9
Description	Profile calibration date and time

This tag indicates when the device was last calibrated and the profile updated with the calibration corrections. Using a monitor example, these corrections can be captured in a simple lookup table to be applied to the generic monitor characterization data already in the tag. Devices like scanners and monitors should be calibrated regularly, but don't necessarily need to be recharacterized, as many printers do.

Sample code to read the calibrationDateTimeTag

```
case 0x63616c74L:  /* 'calt' */
  printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  if (fread(&datetime, sizeof(icDateTimeType), 1, fd) != 1) {
   printf("error reading file\n");
   exit(1);
  }
  printf("Date type\n");
  printf("Date = %d/%d/%d Time = %d:%d:%d\n",
       swap16((ushort)datetime.date.month),
       swap16((ushort)datetime.date.day),
       swap16((ushort)datetime.date.year),
```

```
        swap16((ushort)datetime.date.hours),
        swap16((ushort)datetime.date.minutes),
        swap16((ushort)datetime.date.seconds));

    printf("\n\n");
    break;
```

Sample code to write the calibrationDateTimeTag

Note that the code uses the computer system clock to extract the date and time.

```
tagdir.sig = swap((long)0x63616c74L);                    /*'calt'*/
tagdir.offset = swap((long)tagdataptr);
size = sizeof(tagbase) + sizeof(icDateTimeNumber);
printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }

dirptr += sizeof(tagdir);            /*keep this pointed to the end of
                                     the tag directory*/
tagbase.sig = swap((long)0x6474696dL);          /*'dtim'*/
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
tagdataptr +=sizeof(icTagBase);
/* Get the time from the system */
```

```
    clocktime = time(NULL);
    tmdatetime = localtime(&clocktime);

    datetime.date.seconds = (icUInt16Number)
swap16((ushort)tmdatetime->tm_sec);
    datetime.date.minutes = (icUInt16Number)
swap16((ushort)tmdatetime->tm_min);
    datetime.date.hours = (icUInt16Number)
swap16((ushort)tmdatetime->tm_hour);
    datetime.date.day = (icUInt16Number)s
wap16((ushort)tmdatetime->tm_mday);
    datetime.date.month = (icUInt16Number)s
wap16((ushort)(tmdatetime->tm_mon+1));
    datetime.date.year = (icUInt16Number)
swap16((ushort)(tmdatetime->tm_year + 1900));

    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&datetime.date, sizeof(icDateTimeNumber), 1, fd) != 1) {
      printf("error writing file\n");
      exit(1);
     }

    /*update the final size of profile in the header*/
    tagdataptr += sizeof(icDateTimeType) - sizeof(icTagBase);
    fseek(fd, 0, SEEK_SET);
    tempdataptr = swap((long)tagdataptr);
    if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
      printf("error writing file\n");
      exit(1);
     }
    /* May need add to the tagdataptr to make sure the next tag
```

```
    lands on a four byte boundary
    */
  divt = div(tagdataptr, 4);
  tagdataptr += divt.rem;
```

charTargetTag

Specification Tag Name	charTargetTag
Specification Tag Type	textType
ICC Header Tag Name	icSigCharTargetTag
ICC Header Tag Type	icTextType -> icText (array of characters)
Signature	targ (74617267h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.10
Description	Characterization target such as IT8/7.2

This tag contains the measurement data for a characterization target such as IT8.7/2. The sample code only identifies the target and does not include measurement data. The next version of this book may include real data, if such is donated by an interested party.

Sample code to read the charTargetTag

```
  case 0x74617267L:  /* 'targ' */
    printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
          tag[i].sig, buf[i], tag[i].offset, tag[i].size);
    fseek(fd, (long) tag[i].offset, 0);
    textalloc = (icTextType *)calloc(tag[i].size, sizeof(icUInt8Number));
    if (fread(textalloc, tag[i].size, 1, fd) != 1) {
```

```
        printf("error reading file\n");
        exit(1);
      }
      text = &textalloc->data;
      printf("Text type\n");
      printf("%s", text->data);
      printf("\n\n");
      free(textalloc);
      break;
```

Sample code to write the charTargetTag

```
    tagdir.sig = swap((long)0x74617267L);                    /*targ'*/
    tagdir.offset =  swap((long)tagdataptr);
    /*The string to be entered is 18 characters, including ending space, plus ba
se */
    size = sizeof(tagbase) + 18;
    textalloc = (icTextType *)calloc(size, sizeof(icUInt8Number));
    text = &textalloc->data;
    printf("tag size = %d\n",size);
    tagdir.size =  swap((long)size);
    fseek(fd, dirptr, SEEK_SET);
    if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
the
                                    tag directory*/
    tagbase.sig =  swap((long)0x74657874L);          /*'text'*/
    fseek(fd, tagdataptr, SEEK_SET);
```

```
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
tagdataptr +=sizeof(icTagBase);
strncpy(text->data,"ANSI IT8.7/1-1993",18);

if (fwrite(text->data, 18, 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
free(textalloc);
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

chromaticityTag

Specification Tag Name	chromaticityTag
Specification Tag Type	chromaticitType
ICC Header Tag Name	icSigChromaticityTag
ICC Header Tag Type	icChromaticityType
Signature	chrm(6368726dh)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec 6.4.11
Description	The data and type of phosphor/colorant chromaticity set.

Sample code to read the chromaticityTag

```
case 0x6368726dL:  /* 'chrm ' */
  printf("signature 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  chromaalloc = (icChromaticityType *)calloc(tag[i].size,
                        sizeof(icUInt8Number));
  if (fread(chromaalloc, tag[i].size, 1, fd) != 1) {
   printf("error reading file\n");
   exit(1);
  }
  chroma = &chromaalloc->chromaticity;

  printf("Chromaticity description\n");
  chroma->channels = swap16((ushort)chroma->channels);
  printf("Number of channels %d\n", chroma->channels);
  printf("Phopshor or Colorant type = ");
```

```
switch( (int)swap16((ushort)chroma->type) ) {
  case icITURBT709:
    printf("ITU-R BT.709 \n");
    break;
  case icSMPTERP1451994:
    printf("SMPTE RP145-1994\n");
    break;
  case icEBUTech3213E:
    printf("EBU Tech.3213-E\n");
    break;
  case icP22:
    printf("P22\n");
    break;
  default:
    printf("Unknown\n");
    break;
  }
/* Do everything by bytes */
ii =0;
for (j=0; j<chroma->channels; j++) {
  /*      memcpy(n32, chroma->data[j], 2);*/

  printf(" %f  ",icfixed2double(chroma->data[ii++],
                icSigU16Fixed16ArrayType));
  /*      dataptr += 2;
  memcpy(&val, chroma->data[j+1], 2);*/
  printf(" %f  ",icfixed2double(chroma->data[ii++],
                icSigU16Fixed16ArrayType));
  /*      dataptr += 2;*/
}
```

```
  free(chromaalloc);

  printf("\n\n");
  break;
```

Sample code to write the chromaticityTag

```
tagdir.sig = swap((long)0x6368726dL);                /*'chrm'*/
tagdir.offset = swap((long)tagdataptr);

size = sizeof(tagbase) +
  2 * sizeof(icUInt16Number) +    /*# channels and type */
  6 * sizeof(icU16Fixed16Number);  /* 3 channels of xy coordinates */

chromaalloc = (icChromaticityType *)calloc(size, sizeof(icUInt8Number));
memset(chromaalloc,  0, size);
chroma = &chromaalloc->chromaticity;
ptr8save = (char *)chroma;
printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
dirptr += sizeof(tagdir);            /*keep this pointed to the end of
the
                                     tag directory*/
tagbase.sig = swap((long)0x6368726dL);          /*'chrm'*/
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
```

```
        printf("error writing file\n");
        exit(1);
       }
    tagdataptr +=sizeof(icTagBase);
    chroma->channels = (icUInt16Number)swap16((ushort)3);     /* number of
channels */
    chroma->type = (icUInt16Number)swap16((ushort)2);         /*SMPTE RP145-
1994*/
     /* the xy coordinates are defined int he specification*/
    chroma->data[0] = (icU16Fixed16Number)double2icfixed(0.64,
                icSigU16Fixed16ArrayType);
    chroma->data[1] = (icU16Fixed16Number)double2icfixed(0.33,
                icSigU16Fixed16ArrayType);
    chroma->data[2] = (icU16Fixed16Number)double2icfixed(0.29,
                icSigU16Fixed16ArrayType);
    chroma->data[3] = (icU16Fixed16Number)double2icfixed(0.60,
                icSigU16Fixed16ArrayType);
    chroma->data[4] = (icU16Fixed16Number)double2icfixed(0.15,
                icSigU16Fixed16ArrayType);
    chroma->data[5] = (icU16Fixed16Number)double2icfixed(0.06,
                icSigU16Fixed16ArrayType);
    if (fwrite(ptr8save, size - sizeof(tagbase), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
      }

    /*update the final size of profile in the header*/
    tagdataptr += size - sizeof(tagbase);
    fseek(fd, 0, SEEK_SET);
    tempdataptr = swap((long)tagdataptr);
    if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
```

```
    printf("error writing file\n");
    exit(1);
  }
memset(chromaalloc, 0, size);
free(chromaalloc);
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

copyrightTag

Specification Tag Name	copyrightTag
Specification Tag Type	textType
ICC Header Tag Name	icSigCopyrightTag
ICC Header Tag Type	icTextType -> icText (array of characters
Signature	cprt (63707274h)
Fixed or Variable Length?	variable
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.4.12
Description	7 bit ASCII profile copyright information

Sample code to read the copyrightTag

```
case 0x63707274L:  /* 'cprt' */
 printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
 fseek(fd, (long) tag[i].offset, 0);
 textalloc = (icTextType *)calloc(tag[i].size, sizeof(icUInt8Number));
 if (fread(textalloc, tag[i].size, 1, fd) != 1) {
```

```
        printf("error reading file\n");
        exit(1);
      }
      text = &textalloc->data;
      printf("Text type\n");
      printf("%s", text->data);
      printf("\n\n");
      free(textalloc);
      break;
```

Sample code to write the copyrightTag

```
    tagdir.sig =  swap((long)0x63707274L);                /*cprt'*/
    tagdir.offset =  swap((long)tagdataptr);
    /*The string to be entered is 34 characters, including ending space, plus ba
se */
    size = sizeof(tagbase) + 34;
    textalloc = (icTextType *)calloc(size, sizeof(icUInt8Number));
    text = &textalloc->data;
    printf("tag size = %d\n",size);
    tagdir.size =  swap((long)size);
    fseek(fd, dirptr, SEEK_SET);
    if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    dirptr += sizeof(tagdir);             /*keep this pointed to the end of
the
                                    tag directory*/
    tagbase.sig =  swap((long)0x74657874L);       /*'text'*/
    fseek(fd, tagdataptr, SEEK_SET);
```

```
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
tagdataptr +=sizeof(icTagBase);
strncpy(text->data,"Copyright:  Sun Microsystems 1998",34);

if (fwrite(text, 34, 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
free(textalloc);
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

crdInfoTag

Specification Tag Name	crdInfoTag
Specification Tag Type	crdInfoType
ICC Header Tag Name	icSigCrdInfoTag
ICC Header Tag Type	icCrdInfoType -> icCrdInfo (array of characters)
Signature	crdi (63726469h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.13
Description	Names of companion CRDs to the profile (5 sets of counts and strings)

From the ICC spec..."This tag contains the PostScript product name to which this profile corresponds and the names of the companion CRDs. Recall that a single profile can generate multiple CRDs." Being unable to get real-world data yet, the sample code uses completely unrealistic input. This will be true of many of the tag examples.

Sample code to read the crdInfoTag

```
case 0x63726469L:  /* 'crdi' */

  printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  crdalloc = (icCrdInfoType *)calloc(tag[i].size, sizeof(icUInt8Number));
  if (fread(crdalloc, tag[i].size, 1, fd) != 1) {
   printf("error reading file\n");
   exit(1);
  }
  crd = &crdalloc->info;
  dataptr = crd->data;
  printf("CRD Info type\n");
```

```
        crd->count = swap((long)crd->count);
data    printf("PostScript Product name count and string =  %d,  %s\n", crd->count,
ptr);
        dataptr += crd->count;
        for (ii=0; ii<4; ii++) {
          printf("\n");
          memcpy(&count, dataptr, sizeof(icUInt32Number));
          count = swap((long)count);
          dataptr+= sizeof(icUInt32Number);
datapt  printf("Rendering Intent %d CRD count and name =  %d, %s\n", ii, count,
r);
          dataptr += count;
                    }
        printf("\n\n");
        free(crdalloc);
        break;
```

Sample code to write the crdInfoTag

```
    tagdir.sig =  swap((long)0x63726469L);                  /*crdi'*/
    tagdir.offset = swap((long)tagdataptr);

    size = sizeof(tagbase) +
      sizeof(icUInt32Number) +         /* count*/
      5 * sizeof(icUInt32Number) +     /*5 sets of character counts*/
      136;                             /*5 strings of characters totaled */

    crdalloc = (icCrdInfoType *)calloc(size, sizeof(icUInt8Number));
    crd = &crdalloc->info;
    dataptr = (char *)crd;
```

```
    printf("tag size = %d\n",size);
    tagdir.size = swap((long)size);
    fseek(fd, dirptr, SEEK_SET);
    if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
the
                                    tag directory*/
    tagbase.sig = swap((long)0x63726469L);          /*'crdi'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    tagdataptr +=sizeof(icTagBase);
    n32 = swap((long)24);
    memcpy(dataptr, &n32, sizeof(icUInt32Number));
    dataptr += sizeof(icUInt32Number);
    strncpy(dataptr, "PostScript product name", 24);
    dataptr += 24;
    n32 = swap((long)28);
    memcpy(dataptr, &n32, sizeof(icUInt32Number));
    dataptr += sizeof(icUInt32Number);
    strncpy(dataptr, "Rendering intent 0 CRD name", 28);
    dataptr += 28;
    n32 = swap((long)28);
    memcpy(dataptr, &n32, sizeof(icUInt32Number));
    dataptr += sizeof(icUInt32Number);
    strncpy(dataptr, "Rendering intent 1 CRD name", 28);
```

```
dataptr += 28;
n32 = swap((long)28);
memcpy(dataptr, &n32, sizeof(icUInt32Number));
dataptr += sizeof(icUInt32Number);
strncpy(dataptr, "Rendering intent 2 CRD name", 28);
dataptr += 28;
n32 = swap((long)28);
memcpy(dataptr, &n32, sizeof(icUInt32Number));
dataptr += sizeof(icUInt32Number);
strncpy(dataptr, "Rendering intent 3 CRD name", 28);
dataptr += 28;


if (fwrite(crd, size - sizeof(tagbase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
free(crdalloc);
/* May need add to the tagdataptr to make sure the next tag
  lands on a four byte boundary
  */
divt = div(tagdataptr, 4);
```

```
tagdataptr += divt.rem;
```

deviceMfgDescTag

Specification Tag Name	deviceMfgDescTag
Specification Tag Type	textDescriptionType
ICC Header Tag Name	icSigDeviceMfgDescTag
ICC Header Tag Type	icTextDescriptionType -> icTextDescription
Signature	dmnd (646D6E64h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.14
Description	Displayable description of device manufacturer (Uni and Script code text descriptions)

This tag type is one of the more complicated types because it includes fields for internationalization of the ASCII description. The Unicode field can be utilized for translation of the description into a 2-byte character alphabet, which many non-Roman alphabets require. The ScriptCode field is for the localizable Macintosh description. Since I do not have examples of translations, the sample code will show how to handle these fields when they remain blank. The important point to remember is that the ScriptCode field must always be 67 bytes long, even when no data is provided.

Sample code to read the deviceMfgDescTag

This sample currently only dumps the ASCII text field - it doesn't look for the Unicode or ScriptCode fields.

```
case 0x646D6E64L:  /* 'dmnd' */
  printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  textdescalloc = (icTextDescriptionType *)calloc(tag[i].size,
```

```
sizeof(icUInt8Number));
      if (fread(textdescalloc, tag[i].size, 1, fd) != 1) {
        printf("error reading file\n");
        exit(1);
      }
      textdesc = &textdescalloc->desc;
      printf("Text description \n");
      printf("%s\n", textdesc->data);

      printf("\n\n");
      free(textdescalloc);
      break;
```

Sample code to write the deviceMfgDescTag

```
     tagdir.sig =  swap((long)0x646d6e64L);                /*dmnd'*/
    tagdir.offset =  swap((long)tagdataptr);
    /*tagbase + ascii count + data, UniCode code + count scriptCode code + count
.
     *The Unicode desc to be entered is 48 characters */
    /*see below for detail on the size calculations*/
    size = sizeof(tagbase) +
            sizeof(icUInt32Number) + 48 +
            2*sizeof(icUInt32Number) +
            sizeof(icUInt16Number) + sizeof(icUInt8Number) + 67;

    ptr8 = (char *)calloc(130, sizeof(icUInt8Number));
    ptr8save = ptr8;
    printf("tag size = %d\n",size);
    tagdir.size =  swap((long)size);
    fseek(fd, dirptr, SEEK_SET);
```

```
    if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
      }
    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
the
                                    tag directory*/
    tagbase.sig =  swap((long)0x64657363L);        /*'desc'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
      }
    tagdataptr +=sizeof(icTagBase);

    /* Write the ASCII data  (48 + 4 bytes)*/
    n32 = swap(48);
    memcpy(ptr8, &n32, sizeof(icUInt32Number));
    ptr8 += sizeof(icUInt32Number);
    strncpy(ptr8,"This is the device manufacturer description tag",48);
    ptr8 += 48;

    /* Write in the UniCode data (8 bytes minimum) */
    n32 = 0;
    memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode language code */
    ptr8 += sizeof(icUInt32Number);
    memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode description count*/
    ptr8 += sizeof(icUInt32Number);

    /* Write in the ScriptCode data (70 bytes minimum)*/
    n16 = swap16(0);
```

```
n8=0;
memcpy(ptr8, &n16, sizeof(icUInt16Number));  /*script code language code */
ptr8 += sizeof(icUInt16Number);
memcpy(ptr8, &n8, sizeof(icUInt8Number));  /*scriptcode count */
ptr8 += sizeof(icUInt8Number);
memcpy(ptr8, &n8, 67);              /*required 67 bytes of 0 */
ptr8 += 67;
n32 = 130;
if (fwrite(ptr8save, n32, 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
free(ptr8save);
/* May need add to the tagdataptr to make sure the next tag
  lands on a four byte boundary
  */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

deviceModelDescTag

Specification Tag Name	deviceModelDescTag
Specification Tag Type	textDescriptionType
ICC Header Tag Name	icSigDeviceModelDescTag
ICC Header Tag Type	icTextDescriptionType -> icTextDescription
Signature	dmdd (646D6464h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.15
Description	Displayable description of device model (Uni and Script code text descriptions)

This tag is the same structure as the deviceMfgDescTag.

deviceSettingsTag

Specification Tag Name	device SettingsTag
Specification Tag Type	deviceSettingsType
ICC Header Tag Name	icSigDeviceSettingsTag
ICC Header Tag Type	icDeviceSettingsType -> icSettingsData
Signature	devs(64657673h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.16
Description	Platform specific device settings for which this profile is valid.

Sample code to read the deviceSettingsTag

```
case 0x64657673L:  /* 'devs' */

printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
```

```
            tag[i].sig, buf[i], tag[i].offset, tag[i].size);
        fseek(fd, (long) tag[i].offset, 0);
        settingstype = (icDeviceSettingsType *)calloc(tag[i].size,
 sizeof(icUInt8Number)
);
        settingsdata = &settingstype->data;

        if (fread(settingstype, tag[i].size, 1, fd) != 1) {
          printf("error reading file\n");
          exit(1);
        }
        ptr32 = 0;
        printf("Device Settings type\n");
        settingsdata->numPlatforms = swap((long)settingsdata->numPlatforms);
        printf("Number of Platforms %d \n", settingsdata->numPlatforms);
        for (ii=0; ii< settingsdata->numPlatforms; ii++) {
          switch( swap((long)settingsdata->data[ptr32++]) ) {
          case icSigMacintosh:
            printf("platform = Macintosh\n");
            break;
          case icSigMicrosoft:
            printf("platform = Microsoft\n");
            break;
          case icSigSolaris:
            printf("platform = Solaris\n");
            break;
          case icSigSGI:
            printf("platform = SGI\n");
            break;
          case icSigTaligent:
            printf("platform = Taligent\n");
```

```
          break;
        default:
          printf("Unknown\n");
          break;
        }
        printf("Size of this platform structure %d \n", swap((long)settingsdata->data[
ptr32++]));
        ii2 = swap((long)settingsdata->data[ptr32++]);
        printf("Number of combinations %d \n", ii2);
        for (j=0; j<ii2; j++) {
          printf("Size of this combination structure %d \n",  swap((long)settingsdata-
>data[ptr32++]));
          jj2 = swap((long)settingsdata->data[ptr32++]);
          printf("Number of structures %d \n", jj2);
          for (jj=0; jj<jj2; jj++) {
            printf("Settings signature =");
            switch (swap((long)settingsdata->data[ptr32++])) {
            case icSigResolution:
              printf(" Resolution\n");
              printf("Size of of setting value = %d\n",swap((long)settingsdata->data[p
tr32++]));
              printf("Number of resolution settings = %d\n", swap((long)settingsdata->
data[ptr32++]));
>da          printf("DPI, Y and X resolution = %d %d \n", swap((long)settingsdata-
ta[ptr32++]),
                                                swap((long)settingsdata->d
ata[ptr32++]));
              break;
            case icSigMedia:
              printf(" Media\n");
              printf("Size of of setting value = %d\n",swap((long)settingsdata->data[p
```

```
tr32++]));
            printf("Number of media settings = %d\n", swap((long)settingsdata->data[
ptr32++]));
             printf("Device Media = ");
             switch(swap((long)settingsdata->data[ptr32++])) {
             case icStandard:
               printf("Standard\n");
               break;
             case icTrans:
               printf("Transparency\n");
               break;
             case icGloss:
               printf("Glossy\n");
               break;
             case icUser1:
               printf("User defined\n");
               break;
             default:
               printf(" Unknown\n");
               break;
             }
             break;
           case icSigHalftone:
             printf(" Halftone\n");
             printf("Size of of setting value = %d\n",swap((long)settingsdata->data[p
tr32++]));
            printf("Number of halftone settings = %d\n", swap((long)settingsdata->da
ta[ptr32++]));
             switch(swap((long)settingsdata->data[ptr32++])) {
             case icNone:
               printf("None\n");
```

```
  break;
case icCoarse:
  printf("Coarse\n");
  break;
case icFine:
  printf("Fine\n");
  break;
case icLineArt:
  printf("Line art\n");
  break;
case icErrorDiffusion:
  printf("Error diffusion\n");
  break;
case icReserved6:
  printf("Reserved \n");
  break;
case icReserved7:
  printf("Reserved \n");
  break;
case icReserved8:
  printf("Reserved \n");
  break;
case icReserved9:
  printf("Reserved \n");
  break;
case icGrayScale:
  printf("Gray scale \n");
  break;
case icUser2:
  printf("User defined \n");
  break;
```

```
        default:
          printf(" Unknown\n");
          break;
        }
        break;
      default:
        printf(" Unknown\n");
        break;
      }
    }
  }
}
printf("\n\n");
free(settingstype);
break;
```

Sample code to write the deviceSettingsTag

/* this sample code will include one platform with each of the 3 types of de vices settings */

```
tagdir.sig = swap((long)0x64657673L);              /*devs'*/
tagdir.offset = swap((long)tagdataptr);

size = sizeof(tagbase) +
  1 * sizeof(icUInt32Number) +    /*number of platforms */
  3 * sizeof(icUInt32Number) +    /*platform,  size, #combinations*/
  2 * sizeof(icUInt32Number) +    /*structure size and # structures*/
  4 * sizeof(icUInt32Number) +    /*setting sample 1*/
  4 * sizeof(icUInt32Number) +    /*setting sample 2*/
  5 * sizeof(icUInt32Number);     /*setting sample 3*/
```

```
settingstype = (icDeviceSettingsType *)calloc(size, sizeof(icUInt8Number));
settingsdata = &settingstype->data;

printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
 }
dirptr += sizeof(tagdir);              /*keep this pointed to the end of
                                    the tag directory*/
tagbase.sig = swap((long)0x64657673L);          /*'devs'*/
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
 }

tagdataptr +=sizeof(icTagBase);
ii = 0;
settingsdata->numPlatforms = swap(1);

settingsdata->data[ii++] = swap((long)icSigSolaris);        /*platform*/
settingsdata->data[ii++] = swap((long)18 * sizeof(icUInt32Number));
                                                        /* size*/
settingsdata->data[ii++] = swap((long)1);                /*combCount*/
settingsdata->data[ii++] = swap((long)15 * sizeof(icUInt32Number));
                                                      /*structSize*/
settingsdata->data[ii++] = swap((long)3);                /*numStructs*/
```

```
/* setting sample 1 - media*/
settingsdata->data[ii++] = swap((long)icSigMedia);          /*settingSig*/
settingsdata->data[ii++] = swap((long)sizeof(icUInt32Number));  /*size*/
settingsdata->data[ii++] = swap((long)1);                   /*numSettings*/
settingsdata->data[ii++] = swap((long)icTrans);                 /*tranparency*/
/* setting sample 2 - halftone*/
settingsdata->data[ii++] = swap((long)icSigHalftone);           /*settingSig*/
settingsdata->data[ii++] = swap((long)sizeof(icUInt32Number));      /*size*/
settingsdata->data[ii++] = swap((long)1);                       /*numSettings*/
settingsdata->data[ii++] = swap((long)icErrorDiffusion);            /*dithering*/

/* setting sample 3 - resolution*/
settingsdata->data[ii++] = swap((long)icSigResolution);   /*settingSig*/
settingsdata->data[ii++] = swap((long)sizeof(icUInt64Number));      /*size*/
settingsdata->data[ii++] = swap((long)1);               /*numSettings*/
settingsdata->data[ii++] = swap((long)600);      /*dpi,y resolution*/
settingsdata->data[ii++] = swap((long)400);       /*dpi, x resolution*/


if (fwrite(settingsdata, size- sizeof(tagbase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
   printf("error writing file\n");
```

```
    exit(1);
  }
free(settingstype);

/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

gamutTag

Specification Tag Name	gamutTag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icSigGamutTag
ICC Header Tag Type	icLut8 or icLut16
Signature	gamut (67616D74h)
Fixed or Variable Length?	variable
Required by Profile Types	RGB and CYMK output profiles
ICC Spec section / more info pages	ICC spec: 6.4.17
Description	Out of gamut: 8 or 16 bit data

This tag is the same type as the AToB0Tag and AtoB1Tag. The data and its use, however, are quite different. The tag describes the gamut of the device, aiding the CMM in dealing with colors processed through this profile which are outside the gamut of this device. Please refer to the section on LUT tags for more information. Note: An example of data for the gamut tag may be provided later.

grayTRCTag

Specification Tag Name	grayTRCTag
Specification Tag Type	curveType
ICC Header Tag Name	icSigGrayTag
ICC Header Tag Type	icCurveType ->icCurve (count and 16 bit values)
Signature	kTRC (6B545243h)
Fixed or Variable Length?	variable
Required by Profile Types	Monochrome input profiles Monochrome display profiles Monochrome output profiles
ICC Spec section / more info pages	ICC spec: 6.4.18
Description	Gray tone reproduction curve

This tag is very similar to the blueTRCTag, but contains data for only one channel. Sample code for reading the tag is identical, so included here is sample code for writing the tag only.

Sample code to write the grayTRCTag

```
/*This gray TRC example will use the curve option, supplying 16 values,
  between which the CMM will interpolate missing values.*/
tagdir.sig = swap((long)0x6b545243L);              /*kTRC'*/
tagdir.offset = swap((long)tagdataptr);

size = sizeof(tagbase) + sizeof(icUInt32Number) +
  (16 * sizeof(icUInt16Number));
curvealloc = (icCurveType *)calloc(size, sizeof(icUInt8Number));
curve = &curvealloc->curve;
printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
```

```
        printf("error writing file\n");
        exit(1);
      }
    dirptr += sizeof(tagdir);              /*keep this pointed to the end  of
the       tag directory*/
    tagbase.sig = swap((long)0x63757276L);              /*'curv'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
        printf("error writing file\n");
        exit(1);
      }
    tagdataptr +=sizeof(icTagBase);
    jj = 16;
    curve->count = swap((long)jj);

    /*Supply a curve of values - create the curve using the a gamma of 1.8*/
    for (ii=0; ii<jj; ii++) {
      f32 = (float)((16.0*ii)/255.0);
      curve->data[ii] = (icUInt16Number)swap16((ushort)(65535 * pow(f32, 1.8)));
    }
    curve->data[ii] =  (icUInt16Number)swap16((ushort)(65535 * pow(1.0, 1.8)));
    if (fwrite(curve, size - sizeof(icTagBase), 1, fd) != 1) {
        printf("error writing file\n");
        exit(1);
      }

    /*update the final size of profile in the header*/
    tagdataptr += size - sizeof(tagbase);
    fseek(fd, 0, SEEK_SET);
    tempdataptr = swap((long)tagdataptr);
    if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
```

```
    printf("error writing file\n");
    exit(1);
}
free(curvealloc);

 /* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

greenColorantTag

Specification Tag Name	greenColorantTag
Specification Tag Type	XYZType
ICC Header Tag Name	icSigGreenColorantTag
ICC Header Tag Type	icXYZType -> icXYZArray (3 icS16Fixed16Number's)
Signature	gXYZ (6758595Ah)
Fixed or Variable Length?	fixed
Required by Profile Types	3-component matrix-bases input profiles RGB display profiles
ICC Spec section / more info pages	ICC spec: 6.4.19
Description	Relative XYZ values of green phosphor or colorant

This tag is the same type as the blueColorantTag.

greenTRCTag

Specification Tag Name	greenTRCTag
Specification Tag Type	curveType
ICC Header Tag Name	icSigGreenTRCTag
ICC Header Tag Type	icCurveType ->icCurve (count and 16 bit values)
Signature	gTRC (67545243h)
Fixed or Variable Length?	variable
Required by Profile Types	3-component matrix-bases input profiles RGB display profiles
ICC Spec section / more info pages	ICC spec: 6.4.20
Description	Green channel tone reproduction curve

This tag is the same type as the blueTRCTag and sample code for reading it would be identical. Included here is sample code for writing it using another of its 3 options. This sample indicates only 2 values in the “curve”, which is then assumed to be the minimum and maximum range. All other values are interpolated evenly between that range by the CMM.

Sample code to write the greenTRCTag (min/max range option)

```
/*This TRC example will use the min/max option by setting count to 2 */
 tagdir.sig = swap((long)0x67545243L);              /*gTRC'*/
 tagdir.offset = swap((long)tagdataptr);

 size = sizeof(tagbase) + sizeof(icUInt32Number) + (2
*sizeof(icUInt16Number)
);
 curvealloc = (icCurveType *)calloc(size, sizeof(icUInt8Number));
 curve = &curvealloc->curve;
 printf("tag size = %d\n",size);
 tagdir.size = swap((long)size);
 fseek(fd, dirptr, SEEK_SET);
 if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
```

```
        printf("error writing file\n");
        exit(1);
      }
    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
the
                                    tag directory*/
    tagbase.sig = swap((long)0x63757276L);          /*'curv'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
        printf("error writing file\n");
        exit(1);
      }
    tagdataptr +=sizeof(icTagBase);
    curve->count = swap((long)2);
    /*2 values, provide min and max and cmm will interpolate */
    curve->data[0] = (icUInt16Number)swap16((ushort)0);
    curve->data[1] = (icUInt16Number)swap16((ushort)65535);
    if (fwrite(curve, size - sizeof(icTagBase), 1, fd) != 1) {
        printf("error writing file\n");
        exit(1);
      }

    /*update the final size of profile in the header*/
    tagdataptr += size - sizeof(tagbase);
    fseek(fd, 0, SEEK_SET);
    tempdataptr = swap((long)tagdataptr);
    if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
        printf("error writing file\n");
        exit(1);
      }
    free(curvealloc);
```

```
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

lluminanceTag

Specification Tag Name	luminanceTag
Specification Tag Type	XYZType
ICC Header Tag Name	icSigLuminanceTag
ICC Header Tag Type	icXYZType -> icXYZArray (3 icS16Fixed16Number's)
Signature	lumi (6C756D69h)
Fixed or Variable Length?	fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.21
Description	Absolute luminance for emissive device

The ICC specification describes this as "Absolute luminance of devices in candelas per square meter as described by the Y channel. The X and Z channels are ignored in all cases." The structure of the tag is identical to the blueColorantTag.

measurementTag

Specification Tag Name	measurmentTag
Specification Tag Type	measurementType
ICC Header Tag Name	icSigMeasurementTag
ICC Header Tag Type	icMeasurementType -> icMeasurement
Signature	meas (6D656173h)
Fixed or Variable Length?	fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.22
Description	Alternative measurement specification information (includes standard observer, backing, geometry, flare, and illuminant)

Sample code to read the measurementTag

```
case 0x6D656173L:  /* 'meas' */
  printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  if (fread(&measurement, sizeof(icMeasurementType), 1, fd) != 1) {
    printf("error reading file\n");
    exit(1);
  }
  printf("Measurement Type\n");

  switch(swap((long)measurement.measurement.stdObserver)) {
  case icStdObsUnknown :
    printf("Standard Observer = Unknown\n");
    break;
  case icStdObs1931TwoDegrees :
    printf("Standard Observer = 1931 two degrees\n");
    break;
```

```
case icStdObs1964TenDegrees :
  printf("Standard Observer = 1964 ten degrees\n");
  break;
default :
  printf("Unknown\n");
  break;
}
printf("XYZ backing:  ");
printf("X=%f, Y=%f, Z=%f\n",
      icfixed2double(measurement.measurement.backing.X,
                  icSigS15Fixed16ArrayType),
      icfixed2double(measurement.measurement.backing.Y,
                  icSigS15Fixed16ArrayType),
      icfixed2double(measurement.measurement.backing.Z,
                  icSigS15Fixed16ArrayType));
printf("Geometry Type = ");
switch(swap((long)measurement.measurement.geometry)) {
case icGeometryUnknown :
  printf("Unknown\n");
  break;
case icGeometry045or450 :
  printf("0/45 or 45/0\n");
  break;
case icGeometry0dord0 :
  printf("0/d or d/0\n");
  break;
default :
  printf("Unknown\n");
  break;
}
 printf("Flare percent %f  ",
```

```
        icfixed2double(measurement.measurement.flare,
        icSigU16Fixed16ArrayType));


printf("Illuminant =");
switch(swap((long)measurement.measurement.illuminant)) {
case icIlluminantUnknown  :
  printf("Unknown\n");
  break;
case icIlluminantD50  :
  printf("D50\n");
  break;
case icIlluminantD65  :
  printf("D65\n");
  break;
case icIlluminantD93  :
  printf("D93\n");
case icIlluminantF2  :
  printf("F2\n");
  break;
case icIlluminantD55  :
  printf("D55\n");
  break;
case icIlluminantA  :
  printf("A\n");
  break;
case icIlluminantEquiPowerE  :
  printf("EquiPower E\n");
  break;
case icIlluminantF8  :
  printf("F8\n");
```

```
      break;
    default :
      printf("Unknown\n");
      break;
    }
    printf("\n\n");
    break;
```

Sample code to write the measurementTag

```
  tagdir.sig =  swap((long)0x6d656173L);                /*'meas'*/
 tagdir.offset =  swap((long)tagdataptr);
 size = sizeof(tagbase) + sizeof(icMeasurement);
 printf("tag size = %d\n",size);
 tagdir.size =  swap((long)size);
 fseek(fd, dirptr, SEEK_SET);
 if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
   }


 dirptr += sizeof(tagdir);            /*keep this pointed to the end of
the
                                  tag directory*/
 tagbase.sig =  swap((long)0x6d656173L);       /*'meas'*/
 fseek(fd, tagdataptr, SEEK_SET);
 if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
   }
```

```
    tagdataptr +=sizeof(tagbase);
    measurement.measurement.stdObserver =  swap((long)0x00000001L);
                /*1931 2 degrees*/
    measurement.measurement.backing.X =
      double2icfixed (0.72435, icSigS15Fixed16ArrayType);
    measurement.measurement.backing.Y =
      double2icfixed (0.89567, icSigS15Fixed16ArrayType);
    measurement.measurement.backing.Z =
      double2icfixed (0.95563, icSigS15Fixed16ArrayType);
    measurement.measurement.geometry =  swap((long)0x00000001L);            /*
1/45 or 45/0*/
    measurement.measurement.flare =
      double2icfixed (0.33, icSigU16Fixed16ArrayType);     /*30% flare*/

    measurement.measurement.illuminant =  swap((long)0x00000002L);          /*
D65*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&measurement.measurement, sizeof(measurement.measurement),
            1, fd) != 1) {
      printf("error writing file\n");
      exit(1);
      }

    /*update the final size of profile in the header*/
    tagdataptr += sizeof(measurement.measurement);
    fseek(fd, 0, SEEK_SET);
    tempdataptr = swap((long)tagdataptr);
    if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
      printf("error writing file\n");
      exit(1);
      }
```

```
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

mediaBlackPointTag

Specification Tag Name	mediaBlackPointTag
Specification Tag Type	XYZType
ICC Header Tag Name	icSigMediaBlackPointTag
ICC Header Tag Type	icXYZType -> icXYZArray (3 icS16Fixed16Number's)
Signature	bkpt (626B7074h)
Fixed or Variable Length?	fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.23
Description	Media XYZ black point

From the ICC spec..."This tag specifies the media black point and is used for generating absolute colorimetry. It is referenced to the profile connection space so that the media black point as represented in the PCS is equivalent to this tag value. If the tag is not present, it is assumed to be (0,0,0)." Reading and writing it is depicted by the sample code for the blueColorantTag.

mediaWhitePointTag

Specification Tag Name	mediaWhitePointTag
Specification Tag Type	XYZType
ICC Header Tag Name	icSigMediaWhitePointTag
ICC Header Tag Type	icXYZType -> icXYZArray (3 icS16Fixed16Number's)
Signature	wtpt (77747074h)
Fixed or Variable Length?	fixed
Required by Profile Types	All but Devicelink
ICC Spec section / more info pages	ICC spec: 6.4.24
Description	Media XYZ white point

From the ICC spec..."This tag specifies the media white point and is used for generating absolute colorimetry. It is referenced to the profile connection space so that the media white point as represented in the PCS is equivalent to this tag value. " Reading and writing it is depicted by the sample code for the blueColorantTag

namedColor2Tag

Specification Tag Name	namedColor2Tag
Specification Tag Type	namedColor2Type
ICC Header Tag Name	icSigNamedColor2Tag
ICC Header Tag Type	icSigNamedColor2Type -> icNamedColor2
Signature	ncl2 (6E636C32h)
Fixed or Variable Length?	variable
Required by Profile Types	Named color profiles
ICC Spec section / more info pages	ICC spec: 6.4.26
Description	Named color information for a list of named colors (includes color names and their PCS and device coordinates)

This tag contains "named color information providing a PCS and optional device representation for a list of named colors." This tag replaces the original namedColorTag, which is obsoleted. The ICC does not obsolete tags which are in use due to its committment to backwards compatibility. It was determined that the orginal tag was not in use.

Sample code to read the NamedColor2Tag

```
case 0x6E636C32L:  /* 'ncl2' */
  printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  nc2alloc = (icNamedColor2Type *)calloc(tag[i].size, sizeof(icUInt8Number));
  if (fread(nc2alloc, tag[i].size, 1, fd) != 1) {
    printf("error reading file\n");
    exit(1);
  }
  nc2 = &nc2alloc->ncolor;
  dataptr = (char *) (icNamedColor2 *)nc2->data;
  printf("Named color type\n");
  printf("Vendor = %d\n",swap((long)nc2->vendorFlag));
  printf("Count = %d\n", swap((long)nc2->count));
  printf("Number device coordinates = %d\n",
                swap((long)nc2->nDeviceCoords));
  printf("Color prefix = %s\n", nc2->prefix);
  printf("Color suffix = %s\n",nc2->suffix);
  jj = 0;

  for (ii=0; ii<swap((long)nc2->count); ii++) {
     printf("\nColor  %d  Root name =  ", ii+1);
     printf("%s",dataptr);
     dataptr +=32;
     jj +=32;
     printf("\nPCS Coordinates=  ");

     for (j=0; j< 3; j++) {
        memcpy(&val, dataptr, sizeof(icUInt16Number));
```

```
            printf("%d ", swap16((ushort)val));
            dataptr += sizeof(icUInt16Number);
            jj += sizeof(icUInt16Number);
            if (!(j%16) && (j!=0)) printf("\n");
         }
         printf("\nDevice Coordinates=  ");

         for (j=0; j<swap((long)nc2->nDeviceCoords); j++) {
            memcpy(&val, dataptr, sizeof(icUInt16Number));
            printf("%d ",  swap16((ushort)val));
            dataptr += sizeof(icUInt16Number);
            jj += sizeof(icUInt16Number);
            if (!(j%16) && (j!=0)) printf("\n");
         }
         printf("\n");
      }
      free(nc2alloc);
      printf("\n\n");
      break;
```

Sample code to write the NamedColor2Tag

This sample defines 2 named colors.

```
      tagdir.sig =  swap((long)0x6e636c32L);                /*ncl2'*/
    tagdir.offset =  swap((long)tagdataptr);

    size = sizeof(tagbase) +
      3 * sizeof(icUInt32Number) +    /*vendor, count, # device coords*/
      (64 *sizeof(icUInt8Number)) +   /*2*32 - prefix, suffix arrrays */
      (64 * sizeof(icUInt8Number))+   /*root name array*/
      (12 * sizeof(icUInt16Number));  /*2 colors * 3 coords * 2 sets */
```

```
nc2alloc = (icNamedColor2Type *)calloc(size, sizeof(icUInt8Number));
memset(nc2alloc, 0, size);
nc2 = &nc2alloc->ncolor;
dataptr = (char *)nc2->data;

printf("tag size = %d\n",size);
tagdir.size =  swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
 }
dirptr += sizeof(tagdir);              /*keep this pointed to the end of
                                       the tag directory*/
tagbase.sig =  swap((long)0x64617461L);        /*'data'*/
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
 }
tagdataptr +=sizeof(icTagBase);
nc2->vendorFlag =  swap((long)0x00040000);
                                    /*vendor specific flag-upper 16bits*/
nc2->count =  swap((long)2);       /* number of colors described*/
nc2->nDeviceCoords =  swap((long)3);      /* # coords for each color*/
strncpy(nc2->prefix, "light", 32);
strncpy(nc2->suffix, "ish", 32);
/* color 1 */
strncpy(dataptr, "red", 32);            /* root color name */
dataptr += 32;
```

```
n16 = swap16((ushort)255);
memcpy(dataptr, &n16, sizeof(icUInt16Number));    /*the 3 PCS coords */
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)0);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)0);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
dataptr += sizeof(icUInt16Number);

n16 =  swap16((ushort)128);
memcpy(dataptr, &n16, sizeof(icUInt16Number));    /*the 3 device coords */
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)3);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)4);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
dataptr += sizeof(icUInt16Number);

/* color 2 */
strncpy(dataptr, "blue", 32);     /* root color name */
dataptr += 32;
n16 =  swap16((ushort)0);
memcpy(dataptr, &n16, sizeof(icUInt16Number));    /*the 3 PCS coords */
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)0);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)255);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
```

```
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)80);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
dataptr += sizeof(icUInt16Number);

n16 =  swap16((ushort)10);
memcpy(dataptr, &n16, sizeof(icUInt16Number));    /*the 3 device coords */
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)9);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
dataptr += sizeof(icUInt16Number);
n16 =  swap16((ushort)200);
memcpy(dataptr, &n16, sizeof(icUInt16Number));
dataptr += sizeof(icUInt16Number);

if (fwrite(nc2, size - sizeof(tagbase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
memset(nc2alloc, 0, size);
free(nc2alloc);
/* May need add to the tagdataptr to make sure the next tag
```

```
  lands on a four byte boundary
  */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

outputResponseTag

Specification Tag Name	outputResponseTag
Specification Tag Type	responseCurveSet16Type
ICC Header Tag Name	icSigOutputResponseTag
ICC Header Tag Type	icoutputResponseType -> icResponse
Signature	resp(72657370h)
Fixed or Variable Length?	variaale
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.27
Description	Description of device response for which this profile is intended.

Sample code to read the outputResponseTag

```
case 0x72657370L:  /* 'resp' */

  printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  responsetype = (icResponseCurveSet16Type *)calloc(tag[i].size,
sizeof(icUInt8Number));
  responsedata = &responsetype->data;

  if (fread(responsetype, tag[i].size, 1, fd) != 1) {
    printf("error reading file\n");
    exit(1);
```

```
        }
        ptr32 = 0;
        printf("Response Curve type\n");
        responsedata->channels = swap16((ushort)responsedata->channels);
        responsedata->numTypes = swap16((ushort)responsedata->numTypes);
        printf("Number of Channels %d \n", responsedata->channels);
        printf("Number of response types %d \n",  responsedata->numTypes);
        for (ii=0; ii< responsedata->numTypes; ii++) {
          printf("Byte offset to type %d = %d \n", ii, swap((long)responsedata-
>data[ptr32++]) );
        }
        for (ii=0; ii< responsedata->numTypes; ii++) {
          printf("Type %d = ", ii+1);
          switch( swap((long)responsedata->data[ptr32++]) ) {
          case icStaA:
            printf("StaA \n");
            break;
          case icStaE:
            printf("StaE \n");
            break;
          case icStaI:
            printf(" StaI\n");
            break;
          case icStaT:
            printf("StaT \n");
            break;
          case icStaM:
            printf("StaM \n");
            break;
          case icDN:
            printf("DN \n");
```

```
        break;
      case icDNP:
        printf("DNP \n");
        break;
      case icDNN:
        printf("DNN \n");
        break;
      case icDNNP:
        printf("DNNP \n");
        break;
      default:
        printf("Unknown\n");
        break;
      }
      ii2 = 0;
      for (jj=0; jj<responsedata->channels; jj++) {
        channelvals[ii2] = swap((long)responsedata->data[ptr32++]);
        printf("# measurements for channel %d = %d\n",jj+1,channelvals[ii2]);
        ii2++;
      }
      for (jj=0; jj<responsedata->channels; jj++) {
        printf("xyz measurements for channel %d\n", jj+1);
        for (jj2=0; jj2<channelvals[jj]; jj2++) {
          printf(" %f  ",icfixed2double(responsedata->data[ptr32++],
icSigS15Fixed16ArrayType));
          printf(" %f  ",icfixed2double(responsedata->data[ptr32++],
icSigS15Fixed16ArrayType));
          printf(" %f  \n",icfixed2double(responsedata->data[ptr32++],
icSigS15Fixed16ArrayType));
        }
      }
```

```
        for (jj=0; jj<responsedata->channels; jj++) {
          printf("response data for channel %d\n", jj+1);
          for (jj2=0; jj2<channelvals[jj]; jj2++) {
            printf("Interval = %d ", (swap((long)responsedata->data[ptr32++])>>16));
            printf("value %f  \n",icfixed2double(responsedata->data[ptr32++],
                                        icSigS15Fixed16ArrayType));
          }
          printf("\n");
        }
        printf("\n");
      }

      printf("\n\n");
      free(responsetype);
      break;

    default:
      printf("private tag:  signature = 0x%x signatureId = %s,
offset = %d size = %d\n
",
            tag[i].sig, buf[i], tag[i].offset, tag[i].size);

      printf("\n\n");
      break;
```

Sample code to write the outputResponseTag

```
  /* this sample code will include 2 types of measurement set for 3 channels */

    tagdir.sig = swap((long)0x72657370L);              /*'resp'*/
    tagdir.offset = swap((long)tagdataptr);
```

```
size = sizeof(tagbase) +
  2 * sizeof(icUInt16Number) +    /*# channels, #measurement types*/
  2 * sizeof(icUInt32Number) +    /*offset to each type's data */
  4 * sizeof(icUInt32Number) +
                        /*signature, # measurements for each channel*/
  3 * sizeof(icXYZNumber) +       /*type 1: 3 measurements for type 1*/
  3 * sizeof(icResponse16Number) +
  4 * sizeof(icResponse16Number) +
  2 * sizeof(icResponse16Number) +
  4 * sizeof(icUInt32Number) +
                        /*signature, # measurements for each channel*/

  3 * sizeof(icXYZNumber) +       /*type 2: 3 measurements for type 2*/
  2 * sizeof(icResponse16Number) +
  2 * sizeof(icResponse16Number) +
  2 * sizeof(icResponse16Number);

respcurvetype = (icResponseCurveSet16Type *)calloc(size,
                          sizeof(icUInt8Number));
response = (icResponse *)calloc (size - (4* sizeof(icUInt32Number)),
                       sizeof(icUInt8Number));
response = &respcurvetype->data;

printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
```

```
    dirptr += sizeof(tagdir);                /*keep this pointed to the end of th
e tag directory*/
    tagbase.sig = swap((long)0x72637332L);              /*'rcs2'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }

    tagdataptr +=sizeof(icTagBase);

   response->channels = swap16((ushort)3);
    response->numTypes = swap16((ushort)2);
    i = 0;
    response->data[i++] = swap((long)20); /*byte offset to first measurement type
data*/
    jj = i++;               /*save this to enter offset for second type*/
    /*measurement type 1*/
    response->data[i++] = swap((long)icDN);         /*  measurement type */
    response->data[i++] = swap((long)3);   /*# measurements for channel 1 */
    response->data[i++] = swap((long)4);    /*# measurements for channel 2 */
    response->data[i++] = swap((long)2);     /*# measurements for channel 3 */

    response->data[i++] = double2icfixed (0.53678, icSigS15Fixed16ArrayType);
/* X*/
    response->data[i++] = double2icfixed (0.84736, icSigS15Fixed16ArrayType);
/* Y*/
    response->data[i++] = double2icfixed (0.48475, icSigS15Fixed16ArrayType);
/* Z */
    response->data[i++] = double2icfixed (0.98676, icSigS15Fixed16ArrayType);
/* X */
```

```
    response->data[i++] = double2icfixed (0.77658, icSigS15Fixed16ArrayType);
/* Y */
    response->data[i++] = double2icfixed (0.99867, icSigS15Fixed16ArrayType);
/* Z/

    response->data[i++] = double2icfixed (0.09374, icSigS15Fixed16ArrayType);
/* X*/
    response->data[i++] = double2icfixed (0.04746, icSigS15Fixed16ArrayType);
/* Y */
    response->data[i++] = double2icfixed (0.04958, icSigS15Fixed16ArrayType);
/* Z */
    response->data[i++] = swap(0x00010000);                 /*interval*/
    response->data[i++] = double2icfixed (0.12, icSigS15Fixed16ArrayType);
/*measurement*/
    response->data[i++] = swap(0x00020000);             /*interval*/
    response->data[i++] = double2icfixed (0.34, icSigS15Fixed16ArrayType);
/*measurement*/
    response->data[i++] = swap(0x00030000);              /*interval*/
    response->data[i++] = double2icfixed (0.56, icSigS15Fixed16ArrayType);
/*measurement*/
    response->data[i++] = swap(0x00040000);             /*interval*/
    response->data[i++] = double2icfixed (0.78, icSigS15Fixed16ArrayType);
/*measurement*/
    response->data[i++] = swap(0x00050000);             /*interval*/
    response->data[i++] = double2icfixed (0.91, icSigS15Fixed16ArrayType);
/*measurement*/
    response->data[i++] = swap(0x00060000);             /*interval*/
    response->data[i++] = double2icfixed (0.23, icSigS15Fixed16ArrayType);
/*measurement*/
    response->data[i++] = swap(0x00070000);             /*interval*/
    response->data[i++] = double2icfixed (0.45, icSigS15Fixed16ArrayType);
```

```
/*measurement*/
    response->data[i++] = swap(0x00080000);                 /*interval*/
    response->data[i++] = double2icfixed (0.67, icSigS15Fixed16ArrayType);
 /*measurement*/
    response->data[i++] = swap(0x00090000);                 /*interval*/
    response->data[i++] = double2icfixed (0.89, icSigS15Fixed16ArrayType);
 /*measurement*/


    response->data[jj] = swap((long)(4 *i + 12));      /*byte offset to second mea
surement type data*/
    /*measurement type 2*/
    response->data[i++] = swap((long)icStaT);        /*  measurement type */
    response->data[i++] = swap((long)2);    /*# measurements for channel 1 */
    response->data[i++] = swap((long)2);   /*# measurements for channel 2 */
    response->data[i++] = swap((long)2);    /*# measurements for channel 3 */

    response->data[i++] = double2icfixed (0.0, icSigS15Fixed16ArrayType);
 /* X*/
    response->data[i++] = double2icfixed (0.0, icSigS15Fixed16ArrayType);
 /* Y*/
    response->data[i++] = double2icfixed (0.0, icSigS15Fixed16ArrayType);
 /* Z*/
    response->data[i++] = double2icfixed (0.76678, icSigS15Fixed16ArrayType);
/* X*/
    response->data[i++] = double2icfixed (0.84736, icSigS15Fixed16ArrayType);
 /* Y*/
    response->data[i++] = double2icfixed (0.98475, icSigS15Fixed16ArrayType);
 /* Z */

    response->data[i++] = double2icfixed (0.23676, icSigS15Fixed16ArrayType)
```

```
 /* X */
     response->data[i++] = double2icfixed (0.34658, icSigS15Fixed16ArrayType);
 /* Y */
     response->data[i++] = double2icfixed (0.32867, icSigS15Fixed16ArrayType);
/* Z*/

     response->data[i++] = swap(0x000a0000);                    /*interval*/
     response->data[i++] = double2icfixed (0.98, icSigS15Fixed16ArrayType);
/*measurement*/
     response->data[i++] = swap(0x000b0000);                    /*interval*/
     response->data[i++] = double2icfixed (0.76, icSigS15Fixed16ArrayType);
 /*measurement*/
     response->data[i++] = swap(0x000c0000);                     /*interval*/
     response->data[i++] = double2icfixed (0.54, icSigS15Fixed16ArrayType);
 /*measurement*/
     response->data[i++] = swap(0x000d0000);                    /*interval*/
     response->data[i++] = double2icfixed (0.32, icSigS15Fixed16ArrayType);
/*measurement*/
     response->data[i++] = swap(0x000e0000);                    /*interval*/
     response->data[i++] = double2icfixed (0.19, icSigS15Fixed16ArrayType);
 /*measurement*/
     response->data[i++] = swap(0x000f0000);                   /*interval*/
     response->data[i++] = double2icfixed (0.87, icSigS15Fixed16ArrayType);
/*measurement*/

     if (fwrite(response, size- sizeof(tagbase), 1, fd) != 1) {
        printf("error writing file\n");
        exit(1);
       }

     /*update the final size of profile in the header*/
```

```
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }

/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

preview0Tag

Specification Tag Name	preview0Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icSigPreview0Tag
ICC Header Tag Type	icLut8 or icLut16
Signature	pre0 (70726530h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.28
Description	Preview transformation: 8 or 16 bit data; PCS to Device space and back to PCS, perceptual instent

The structure of this tag is the same as AToB0Tag and AToB1Tag, so the sample code would apply. This is used for previewing an image, normally on a monitor, which is destined for a device, usually a printer. Color verification can then be

visually performed to spot problems, like gamut mismatches. The “preview” tags would be created by the CMM upon linking two or more profiles together, either automatically or at the request of the user, to allow this capability.

preview1Tag

Specification Tag Name	preview1Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icSigPreview1Tag
ICC Header Tag Type	icLut8 or icLut16
Signature	pre01(70726531h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.29
Description	Preview transformation: 8 or 16 bit data; PCS to Device space and back to PCS, relative colorimetric intent

This is the same as preview0Tag, except that it is for the relative colorimetric rendering intent.

preview2Tag

Specification Tag Name	preview2Tag
Specification Tag Type	lut8Type or lut16Type
ICC Header Tag Name	icSigPreview2Tag
ICC Header Tag Type	icLut8 or icLut16
Signature	pre2 (70726532h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.30
Description	Preview transformation: 8 or 16 bit data; PCS to Device space and back to PCS, saturation intent

This is the same as preview0Tag, except that it is for the saturation rendering intent.

profileDescriptionTag

Specification Tag Name	profileDescriptionTag
Specification Tag Type	textDescriptionType
ICC Header Tag Name	icSigProfileDescriptionTag
ICC Header Tag Type	icTextDescriptionType -> icTextDescription
Signature	desc (64657363h)
Fixed or Variable Length?	variable
Required by Profile Types	All
ICC Spec section / more info pages	ICC spec: 6.4.31
Description	Displayable profile description (Uni and Script code text descriptions)

This tag type and code sample is the same as the deviceMfgDescriptionTag.

profileSequenceTag

Specification Tag Name	profileSequenceTag
Specification Tag Type	profileSeqenceDescType
ICC Header Tag Name	icSigProfileSequenceDescTag
ICC Header Tag Type	icProfileSequenceDescType -> icProfileSequenceDesc
Signature	pseq (70736571h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4 32
Description	Profile sequence description from source to destination (includes device description signatures and character text for each profile in sequence)

This tag is created by the color management software as a result of linking 2 or more profiles together. The tag appears in a device link profile, an example of which is provided in Chapter 8.

Sample code to read the profileSequenceTag

```
case 0x70736571L:  /* 'pseq' */
  printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  profseqdescalloc = (icProfileSequenceDescType *)calloc(tag[i].size,
            sizeof(icUInt8Number));
  if (fread(profseqdescalloc,  tag[i].size, 1, fd) != 1) {
   printf("error reading file\n");
   exit(1);
  }
  profseqdesc = &profseqdescalloc->desc;
  dataptr =  (char *)(icProfileSequenceDesc *)profseqdesc;
  profseqdesc->count = swap((long)profseqdesc->count);
  printf("Number of profile descriptions in tag = %d\n",profseqdesc->count );
```

```
dataptr += sizeof(icUInt32Number);
for (jj= 0;jj<profseqdesc->count; jj++) {
  memcpy(&charstring, dataptr, sizeof(icSignature));
  printf("device manufacturer  = %.4s\n", charstring);
  dataptr +=  sizeof(icSignature);
  memcpy(&charstring, dataptr, sizeof(icSignature));
  printf("device model  = %.4s\n", charstring);
  dataptr += sizeof(icSignature);
  memcpy(&tmp64, dataptr,  sizeof(icUInt64Number));
  printf("device attributes=  %d,
       %d\n",swap((long)tmp64[0]),swap((long)tmp64[1]));
  dataptr += sizeof(icUInt64Number);
  memcpy(&charstring, dataptr, sizeof(icSignature) );
  printf("technology  = %.4s\n", charstring);
  dataptr += sizeof(icSignature);
  /* Device Manufacturer tag */
  dataptr += 8;       /* Skip the tag base and save the count */
  memcpy(&count, dataptr, sizeof(count));
  count = (icUInt32Number)swap(count);
  dataptr += sizeof(count);
  printf("Device manufacturer string length and contents = %d %s\n",count,
      dataptr);
/*
 * Skip over the string, UniCode count and type and
 * ScriptCode stuff.
 */
  dataptr += count;   /* Ascii string */
/* UniCode */
  memcpy(&count, (dataptr + 4), sizeof(icUInt32Number));
  /*count = swap((long)count);*/
  dataptr += (2*sizeof(icUInt32Number) + count*2);
```

```
/* ScriptCode */
  dataptr += (sizeof(icUInt16Number) + sizeof(icUInt8Number)
              + 67);
/* Device Model Tag */
  dataptr += 8;       /* Skip the tag base and count */
  memcpy(&count, dataptr, sizeof(icUInt32Number));
  count = swap((long)count);
  dataptr += sizeof(count);
  printf("Device model string length and contents = %d  %s\n",count, dataptr);
/*
 * Skip over the string, UniCode count and type and
 * ScriptCode stuff.
 */
  dataptr += count;   /* Ascii string */
/* UniCode */
  memcpy(&count, (dataptr + 4), sizeof(icUInt32Number));
  count = swap((long)count);
  dataptr += (2*sizeof(icUInt32Number) + count*2);
/* ScriptCode */
  dataptr += (sizeof(icUInt16Number) + sizeof(icUInt8Number)
              + 67);
  printf("\n\n");
 }
 free(profseqdescalloc);
 break;
```

Sample code to write the profileSequenceTag

```
/*This profile sequence tag example will assume 2 profiles are linked */
tagdir.sig =  swap((long)0x70736571L);                  /*pseq'*/
tagdir.offset =  swap((long)tagdataptr);
/*The profile sequence tag would normally be built by a CMM upon linking two
```

```
 * or more profiles together.  This is just an example of what would be
 * written into the tag.  The size would be derived from the fields of the
 * profiles combined.  Here it is hardcoded for 2 fictitious profiles.*/
size = sizeof(tagbase) + sizeof(icUInt32Number) + /*# profile descriptions*/
  2 * sizeof(icSignature) +                  /*deviceMfg*/
  2 * sizeof(icSignature) +                  /*deviceModel*/
  2 * sizeof(icUInt64Number) +                /*attributes*/
  2 * sizeof(icSignature) +                  /*technology*/
  138 + 132 +                 /*device & model desc from profile 1*/
  138 + 132;                  /*device & model desc from profile 2*/
descstructalloc = (icDescStruct *)calloc(size, sizeof(icUInt8Number));
descstruct = descstructalloc;
printf("tag size = %d\n",size);
tagdir.size =  swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
dirptr += sizeof(tagdir);      /*keep this pointed to the end of the tag directory*/
tagbase.sig =  swap((long)0x70736571L);          /*'pseq'*/
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
tagdataptr +=sizeof(icTagBase);
count =  swap((long)2);
if (fwrite(&count, sizeof(icUInt32Number), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
```

```
  }
tagdataptr += sizeof(icUInt32Number);

/*profile 1 info*/
descstruct->deviceMfg =  swap((long)0x53554e57L);   /*SUNW  from header*/
descstruct->deviceModel =  swap((long)0x31393938L);    /*1998 from header*/
descstruct->attributes[1] =  swap((long)15);  /*Transparency,matte,
                                                negative, B&W*/
descstruct->technology =  swap((long)0x43525420L);            /*CRT */

if (fwrite(descstruct, 20, 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
tagdataptr +=20;

/* text description tag type for profile 1 device */
ptr8 = (char *)calloc(138, sizeof(icUInt8Number));
memset(ptr8, 0, 138);
ptr8save = ptr8;

n32 =0x64657363L;                       /*desc*/
memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
n32 = 0;
memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
/* Write the ASCII data  (48 bytes)*/
n32 = (icUInt32Number)swap(48);

memcpy(ptr8, &n32, sizeof(icUInt32Number));
```

```
ptr8 += sizeof(icUInt32Number);
strncpy (ptr8,"This is the device manufacturer description tag", 48);
ptr8 += 48;

/* Write in the UniCode data (8 bytes minimum) */
n32 = swap((long)0);
memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode language code */
ptr8 += sizeof(icUInt32Number);
memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode description count*/
ptr8 += sizeof(icUInt32Number);

/* Write in the ScriptCode data (70 bytes minimum)*/
n16 = swap16((ushort)0);
n8=0;
memcpy(ptr8, &n16, sizeof(icUInt16Number));  /*script code language code */
ptr8 += sizeof(icUInt16Number);
memcpy(ptr8, &n8, sizeof(icUInt8Number));  /*scriptcode count */
ptr8 += sizeof(icUInt8Number);
memcpy(ptr8, &n8, 67);              /*required 67 bytes of 0 */
ptr8 += 67;
n32 = 138;

if (fwrite(ptr8save, n32, 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
tagdataptr += 138;
memset(ptr8save, 0, 138);

/* text description tag type for profile 1 model */
```

```
ptr8 = ptr8save;
n32 = 0x64657363L;                    /*desc*/
memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
n32 = 0;
memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
/* Write the ASCII data  (42 + 4 bytes)*/
n32 = swap((long)42);

memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
strncpy (ptr8,"This is the device model description tag ", 42);
ptr8 += 42;

/* Write in the UniCode data (8 bytes minimum) */
n32 = swap((long)0);
memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode language code */
ptr8 += sizeof(icUInt32Number);
memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode description count*/
ptr8 += sizeof(icUInt32Number);

/* Write in the ScriptCode data (70 bytes minimum)*/
n16 = swap16((ushort)0);
n8=0;
memcpy(ptr8, &n16, sizeof(icUInt16Number));  /*script code language code */
ptr8 += sizeof(icUInt16Number);
memcpy(ptr8, &n8, sizeof(icUInt8Number));  /*scriptcode count */
ptr8 += sizeof(icUInt8Number);
memcpy(ptr8, &n8, 67);              /*required 67 bytes of 0 */
ptr8 += 67;
```

```
n32 = (124 + 8);

if (fwrite(ptr8save, n32, 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
tagdataptr += 132;

memset(ptr8, 0, 138);


/*profile 2 info*/
descstruct->deviceMfg =  swap((long)0x53554e57L); /*SUNW  fromheader*/
descstruct->deviceModel =  swap((long)0x31393937L);   /*1997 from header*/
descstruct->attributes[1] =  swap((long)2);    /*Reflective, matte ,
                                                positive, color*/
descstruct->technology =  swap((long)0x696a6574L);              /*ijet */

if (fwrite(descstruct, 20, 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
tagdataptr +=20;

/* text description tag type for profile 2 device */

ptr8 = ptr8save;
n32 = 0x64657363L;                        /*desc*/
memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
n32 = 0;
```

```
memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
/* Write the ASCII data  (48 + 4 bytes)*/
n32 = swap((long)48);

memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
strncpy (ptr8,"This is the device manufacturer description tag", 48);
ptr8 += 48;

/* Write in the UniCode data (8 bytes minimum) */
n32 = swap((long)0);
memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode language code */
ptr8 += sizeof(icUInt32Number);
memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode description count*/
ptr8 += sizeof(icUInt32Number);

/* Write in the ScriptCode data (70 bytes minimum)*/
n16 = swap16((ushort)0);
n8=0;
memcpy(ptr8, &n16, sizeof(icUInt16Number));  /*script code language code */
ptr8 += sizeof(icUInt16Number);
memcpy(ptr8, &n8, sizeof(icUInt8Number));  /*scriptcode count */
ptr8 += sizeof(icUInt8Number);
memcpy(ptr8, &n8, 67);              /*required 67 bytes of 0 */
ptr8 += 67;
n32 = 138;

if (fwrite(ptr8save, n32, 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
```

```
  }
tagdataptr += 138;
memset(ptr8save, 0, 138);


/* text description tag type for profile 2 model */
ptr8 = ptr8save;
n32 = 0x64657363L;                      /*desc*/
memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
n32 = 0;
memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
/* Write the ASCII data  (42 + 4 bytes)*/
n32 = swap((long)42);

memcpy(ptr8, &n32, sizeof(icUInt32Number));
ptr8 += sizeof(icUInt32Number);
strncpy (ptr8,"This is the device model description tag ", 42);
ptr8 += 42;
/* Write in the UniCode data (8 bytes minimum) */
n32 = 0;
memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode language code */
ptr8 += sizeof(icUInt32Number);
memcpy(ptr8, &n32, sizeof(icUInt32Number));  /*Unicode description count*/
ptr8 += sizeof(icUInt32Number);

/* Write in the ScriptCode data (70 bytes minimum)*/
n16 = swap16((ushort)0);
n8=0;
memcpy(ptr8, &n16, sizeof(icUInt16Number));  /*script code language code */
```

```
ptr8 += sizeof(icUInt16Number);
memcpy(ptr8, &n8, sizeof(icUInt8Number));  /*scriptcode count */
ptr8 += sizeof(icUInt8Number);
memcpy(ptr8, &n8, 67);              /*required 67 bytes of 0 */
ptr8 += 67;
n32 = (124 + 8);
if (fwrite(ptr8save, n32, 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
tagdataptr += 132;

free(ptr8save);

/*update the final size of profile in the header*/
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
free(descstructalloc);
/* May need add to the tagdataptr to make sure the next tag
  lands on a four byte boundary
  */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

ps2CRD0Tag

Specification Tag Name	ps2CRD0Tag
Specification Tag Type	dataType
ICC Header Tag Name	icSigPs2CRD0Tag
ICC Header Tag Type	icDataType -> icData (array of characters)
Signature	psd0 (70736430h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.33
Description	PostScript Level 2 color rendering dictionary: perceptual

Sample code to read the ps2CRD0Tag

```
case 0x70736430L:  /* 'psd0' */

 printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
       tag[i].sig, buf[i], tag[i].offset, tag[i].size);
 fseek(fd, (long) tag[i].offset, 0);

 datatalloc = (icDataType *)calloc(tag[i].size, sizeof(icUInt8Number));
 if (fread(datatalloc,tag[i].size, 1, fd) != 1) {
  printf("error reading file\n");
  exit(1);
 }
 datat = &datatalloc->data;
 printf("Data type\n");
 if (swap((long)datat->dataFlag) == 0) {
    printf("Ascii data \n");
    printf("%s", datat->data);
 } else {
    printf("Binary data \n");
```

```
        printf("Size = %d bytes\n",(tag[j].size - 8 -
                                sizeof(datat->dataFlag) ));
    }

    printf("\n\n");
    free(datatalloc);
    break;
```

Sample code to write the ps2CRD0Tag

```
    tagdir.sig =  swap((long)0x70736430L);                /*psd0'*/
    tagdir.offset =  swap((long)tagdataptr);
    /*The string to be entered is 34 characters, including ending space, plus ba
se */
    size = sizeof(tagbase) + 18;
    datatalloc = (icDataType *)calloc(size, sizeof(icUInt8Number));
    memset(datatalloc, 0, size);
    datat = &datatalloc->data;
    printf("tag size = %d\n",size);
    tagdir.size =  swap((long)size);
    fseek(fd, dirptr, SEEK_SET);
    if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    dirptr += sizeof(tagdir);            /*keep this pointed to the end of
                                        the tag directory*/
    tagbase.sig =  swap((long)0x64617461L);        /*'data'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
       printf("error writing file\n");
```

```
    exit(1);
  }
tagdataptr +=sizeof(icTagBase);
datat->dataFlag = 0;                    /*indicates ascii data to follow*/
strncpy(datat->data,"PS2 CRD 0 tag",14);

if (fwrite(datat, 18, 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
memset(datatalloc, 0, size);
free(datatalloc);
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

ps2CRD1Tag

Specification Tag Name	ps2CRD1Tag
Specification Tag Type	dataType
ICC Header Tag Name	icSigPs2CRD1Tag
ICC Header Tag Type	icDataType -> icData (array of characters)
Signature	psd1 (70736431h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.34
Description	PostScript Level 2 color rendering dictionary: colorimetric

See ps2CRD0Tag.

ps2CRD2Tag

Specification Tag Name	ps2CRD2Tag
Specification Tag Type	dataType
ICC Header Tag Name	icSigPs2CRD2Tag
ICC Header Tag Type	icDataType -> icData (array of characters)
Signature	psd2 (70736432h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.35
Description	PostScript Level 2 color rendering dictionary: saturation

See ps2CRD0Tag.

ps2CRD3Tag

Specification Tag Name	ps2CRD3Tag
Specification Tag Type	dataType
ICC Header Tag Name	icSigPs2CRD3Tag
ICC Header Tag Type	icDataType -> icData (array of characters)
Signature	psd3 ((70736433h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.36
Description	PostScript Level 2 color rendering dictionary: absolute

See ps2CRD0Tag.

ps2CSATag

Specification Tag Name	ps2CSATag
Specification Tag Type	dataType
ICC Header Tag Name	icSigPs2CSATag
ICC Header Tag Type	icDataType -> icData (array of characters)
Signature	ps2s(70733273h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.37
Description	PostScript Level 2 color space array

See ps2CRD0Tag.

ps2RenderingIntentTag

Specification Tag Name	ps2RenderingIntentTag
Specification Tag Type	dataType
ICC Header Tag Name	icSigPs2RenderingIntentTag
ICC Header Tag Type	icDataType -> icData (array of characters)
Signature	ps2i(70733269h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec; 6.5.38
Description	PostScript Level 2 rendering intent

See ps2CRD0Tag.

redColorantTag

Specification Tag Name	redColorantTag
Specification Tag Type	XYZType
ICC Header Tag Name	icSigRedColorantTag
ICC Header Tag Type	icXYZType -> icXYZArray (3 icS16Fixed16Number's)
Signature	rXYZ (7258595Ah)
Fixed or Variable Length?	fixed
Required by Profile Types	3-component matrix-bases input profiles RGB display profiles
ICC Spec section / more info pages	ICC spec: 6.4.39
Description	Relative XYZ values of red phosphor or colorant

See the blueColorantTag.

redTRCTag

Specification Tag Name	redTRCTag
Specification Tag Type	curvetype
ICC Header Tag Name	icSigRedTRCTag
ICC Header Tag Type	icCurveType ->icCurve (count and 16 bit values)
Signature	rTRC (72545243h)
Fixed or Variable Length?	variable
Required by Profile Types	3-component matrix-bases input profiles RGB display profiles
ICC Spec section / more info pages	ICC spec: 6.4.40
Description	Red channel tone reproduction curve

See the BlueTRCTag.

screeningDescriptionTag

Specification Tag Name	screeningDescTag
Specification Tag Type	textDescriptionType
ICC Header Tag Name	icSigScreeningDescTag
ICC Header Tag Type	icTextDescriptionType -> icTextDescription
Signature	scrd (73637264h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.41
Description	Screening attributes description (Uni and Script code text descriptions)

See the deviceMfgDescTag.

screeningTag

Specification Tag Name	screeningTag
Specification Tag Type	screeningType
ICC Header Tag Name	icSigScreeningTag
ICC Header Tag Type	icScreeningType -> icScreening -> icScreeningData
Signature	scrn (7363726Eh)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.42
Description	Screening attributes such as frequency, angle and spot for each channel

Sample code to read the screeningTag

```
case 0x7363726EL:  /* 'scrn' */

 printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
 fseek(fd, (long) tag[i].offset, 0);
 screenalloc = (icScreeningType *)calloc(tag[i].size, sizeof(icUInt8Number));
 if (fread(screenalloc, tag[i].size, 1, fd) != 1) {
   printf("error reading file\n");
   exit(1);
 }
 screen = &screenalloc->screen;
 printf("Screening data\n");
 printf("Number of channels = %d\n", swap((long)screen->channels));
 printf("Screening flag = %d\n", swap((long)screen->screeningFlag));
 scptr = screen->data;
 for (j=0; j<swap((long)screen->channels); j++) {
    printf("Channel %d\n", j);
```

```
    printf("Frequency = %f\n",
        icfixed2double(scptr->frequency, icSigS15Fixed16ArrayType));
    printf("Angle = %f\n",
        icfixed2double(scptr->angle, icSigS15Fixed16ArrayType));
    printf("SpotShape = %d\n", swap((long)scptr->spotShape));
    scptr++;
  }

  printf("\n\n");
  free(screenalloc);
  break;
```

Sample code to write the screeningTag

```
 tagdir.sig =  swap((long)0x7363726eL);                /*scrn'*/
tagdir.offset =  swap((long)tagdataptr);

size = sizeof(tagbase) +
  2 * sizeof(icUInt32Number) +    /*# channels and screening flag*/
  6 * sizeof(icS15Fixed16Number) +  /*3 * frequency and angle */
  3 * sizeof(icUInt32Number);     /*3 * spot shape */

screenalloc = (icScreeningType *)calloc(size, sizeof(icUInt8Number));
memset(screenalloc, 0, size);
screen = &screenalloc->screen;
printf("tag size = %d\n",size);
tagdir.size =  swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
```

```
  }
 dirptr += sizeof(tagdir);              /*keep this pointed to the end of
                                        the tag directory*/
 tagbase.sig =  swap((long)0x64617461L);        /*'data'*/
 fseek(fd, tagdataptr, SEEK_SET);
 if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
 tagdataptr +=sizeof(icTagBase);
 jj = 3;
 screen->channels =  swap((long)jj);
 screen->screeningFlag =  swap((long)0x00000003L); /*use printer default,
                                                    lines per inch*/
 for (ii = 0; ii < jj; ii++) {
  screen->data[ii].frequency =  double2icfixed(30.56+ii, icSigS15Fixed16Arra
yType);
   screen->data[ii].angle =  double2icfixed(5.489+ii, icSigS15Fixed16ArrayTyp
e);
   screen->data[ii].spotShape =  swap((long)2);              /*round*/
 }
 if (fwrite(screen, size - sizeof(tagbase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }

 /*update the final size of profile in the header*/
 tagdataptr += size - sizeof(tagbase);
 fseek(fd, 0, SEEK_SET);
 tempdataptr = swap((long)tagdataptr);
 if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
```

```
    printf("error writing file\n");
    exit(1);
  }
memset(screenalloc, 0, size);
free(screenalloc);
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

technologyTag

Specification Tag Name	technologyTag
Specification Tag Type	signatureType
ICC Header Tag Name	icSigTechnologyTag
ICC Header Tag Type	icSignature
Signature	tech (74656368h)
Fixed or Variable Length?	fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.43
Description	Device technology information such as LCS, CRT, dye sublimation, etc.

Sample code to read the technologyTag

```
case 0x74656368L:  /* 'tech' */
printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
      tag[i].sig, buf[i], tag[i].offset, tag[i].size);
fseek(fd, (long) tag[i].offset, 0);
if (fread(&tech, sizeof(icSignatureType), 1, fd) != 1) {
```

```
  printf("error reading file\n");
  exit(1);
}
switch(swap((long)tech.signature)) {

case 0x6463616DL:  /* 'dcam' */
  printf("technology type is DigitalCamera \n");
  break;
case 0x6673636EL:  /* 'fscn' */
  printf("technology type is FilmScanner \n");
  break;
case 0x7273636EL:  /* 'rscn' */
  printf("technology type is ReflectiveScanner  \n");
  break;
case 0x696A6574L:  /* 'ijet' */
  printf("technology type is InkJetPrinter \n");
  break;
case 0x74776178L:  /* 'twax' */
  printf("technology type is ThermalWaxPrinter \n");
  break;
case 0x6570686FL:  /* 'epho' */
  printf("technology type is ElectrophotographicPrinter \n");
  break;
case 0x65737461L:  /* 'esta' */
  printf("technology type is ElectrostaticPrinter \n");
  break;
case 0x64737562L:  /* 'dsub' */
  printf("technology type is DyeSublimationPrinter \n");
  break;
case 0x7270686FL:  /* 'rpho' */
  printf("technology type is PhotographicPaperPrinter \n");
```

```
  break;
case 0x6670726EL:  /* 'fprn' */
  printf("technology type is FilmWriter \n");
  break;
case 0x7669646DL:  /* 'vidm' */
  printf("technology type is VideoMonitor \n");
  break;
case 0x76696463L:  /* 'vidc' */
  printf("technology type is VideoCamera \n");
  break;
case 0x706A7476L:  /* 'pjtv' */
  printf("technology type is ProjectionTelevision \n");
  break;
case 0x43525420L:  /* 'CRT ' */
  printf("technology type is CRTDisplay \n");
  break;
case 0x504D4420L:  /* 'PMD ' */
  printf("technology type is PMDisplay \n");
  break;
case 0x414D4420L:  /* 'AMD ' */
  printf("technology type is AMDisplay \n");
  break;
case 0x4B504344L:  /* 'KPCD' */
  printf("technology type is PhotoCD \n");
  break;
case 0x696D6773L:  /* 'imgs' */
  printf("technology type is PhotoImageSetter \n");
  break;
case 0x67726176L:  /* 'grav' */
  printf("technology type is Gravure \n");
  break;
```

```
  case 0x6F666673L:  /* 'offs' */
    printf("technology type is OffsetLithography \n");
    break;
  case 0x73696C6BL:  /* 'silk' */
    printf("technology type is Silkscreen \n");
    break;
  case 0x666C6578L:  /* 'flex' */
    printf("technology type is Flexography \n");
    break;       }
  printf("\n\n");
  break;
```

Sample code to write the technologyTag

```
tagdir.sig = swap((long)0x74656368L);                    /*'tech'*/
tagdir.offset = swap((long)tagdataptr);
size = sizeof(tagbase) + sizeof(icSignature);
printf("tag size = %d\n",size);
tagdir.size = swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }

dirptr += sizeof(tagdir);              /*keep this pointed to the
                                      end of the tag directory*/
tagbase.sig = swap((long)0x73696720L);          /*'sig '*/
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
```

```
  }
tagdataptr +=sizeof(tagbase);
sig = swap((long)0x43525420);     /*’CRT ’*/
if (fwrite(&sig, sizeof(icSignature), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += sizeof(icSignature);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }

/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

ucrbgTag

Specification Tag Name	ucrbgTag
Specification Tag Type	ucrbgType
ICC Header Tag Name	icSigUcrBgTag
ICC Header Tag Type	icUcrBgType -> icUcrBg -> icUrgBgCurve
Signature	bfd (62666442h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.44
Description	Under color removal and black generation description (includes ucr and bg curves and descriptions)

Sample code to read the ucrbgTag

```
case 0x62666420L:  /* 'bfd ' */
  printf("signature 0x%x signatureId = %s, offset = %d size = %d\n",
        tag[i].sig, buf[i], tag[i].offset, tag[i].size);
  fseek(fd, (long) tag[i].offset, 0);
  ucrbgalloc = (icUcrBgType *)calloc(tag[i].size, sizeof(icUInt8Number));
  if (fread(ucrbgalloc, tag[i].size, 1, fd) != 1) {
   printf("error reading file\n");
   exit(1);
  }
  ucrbg = &ucrbgalloc->data;
  dataptr = (char *)ucrbg->data;

  printf("UCR BG description\n");

  /* Do everything by bytes */

  /* UCR Curve */
```

```
memcpy(&count, dataptr, 4);
count = swap((long)count);
printf("UCR curve length = %d\n", count);
dataptr += 4;
if (count == 1) {
   /* Its a percentage */
   memcpy(&val, dataptr, 2);
   printf("UCR = %d percent\n", swap16((ushort)val));
   dataptr += 2;
} else {
   for (j=0; j<count; j++) {
      memcpy(&val, dataptr, 2);
      printf("%d\n",swap16((ushort)val));
      dataptr += 2;
   }
}

/* BG Curve */
memcpy(&count, dataptr, 4);
count = swap((long)count);
printf("BG curve length = %d\n", count);
dataptr += 4;
if (count == 1) {
   /* Its a percentage */
   memcpy(&val, dataptr, 2);
   printf("BG = %d percent\n", swap16((ushort)val));
   dataptr += 2;
} else {
   for (j=0; j<count; j++) {
      memcpy(&val, dataptr, 2);
      printf("%d\n", swap16((ushort)val));
```

```
        dataptr += 2;
    }
}

printf("UcrBg description = %s\n", dataptr);
free(ucrbgalloc);

printf("\n\n");
break;
```

Sample code to write the ucrbgTag

```
tagdir.sig =  swap((long)0x62666420L);                /*bfd '*/
tagdir.offset =  swap((long)tagdataptr);

size = sizeof(tagbase) +
  sizeof(icUInt32Number) +        /*# UCR curve points*/
  (6 *sizeof(icUInt16Number)) +   /*UCR curve points (6 of them) */
  sizeof(icUInt32Number) +        /*# BG points ( 1= percentage) */
  sizeof(icUInt16Number) +        /*percentage BG */
  28;                             /*description length*/

ucrbgalloc = (icUcrBgType *)calloc(size, sizeof(icUInt8Number));
memset(ucrbgalloc, 0, size);
ucrbg = &ucrbgalloc->data;
dataptr = (char *)ucrbg->data;

printf("tag size = %d\n",size);
tagdir.size =  swap((long)size);
fseek(fd, dirptr, SEEK_SET);
if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
```

```
        printf("error writing file\n");
        exit(1);
      }
    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
                                           the  tag directory*/
    tagbase.sig =  swap((long)0x64617461L);          /*'data'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
        printf("error writing file\n");
        exit(1);
      }
    tagdataptr +=sizeof(icTagBase);
    n32 = swap((long)6);
    memcpy(dataptr, &n32, sizeof(icUInt32Number));   /*number of UCR curve
                                                     points*/
    dataptr += sizeof(icUInt32Number);
    for (ii=0; ii< 6; ii++) {
      n16 = swap16((ushort)(ii *(256/6)));
      memcpy(dataptr, &n16, sizeof(icUInt16Number));
      dataptr+= sizeof(icUInt16Number);
    }
    n32 =  swap((long)1);
    memcpy(dataptr, &n32, sizeof(icUInt32Number));   /*number of BG curve
 points
*/
    dataptr +=  sizeof(icUInt32Number);       /* 1 means it is a percentage*/
    n16 =swap16((ushort) 94);                        /* 94%*/
    memcpy(dataptr, &n16, sizeof(icUInt16Number));
    dataptr+= sizeof(icUInt16Number);
    strncpy(dataptr, "This is a UcrBg description",28);
    dataptr += 28;
```

```
if (fwrite(ucrbg->data, size - sizeof(tagbase), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
    printf("error writing file\n");
    exit(1);
  }
memset(ucrbgalloc, 0, size);
free(ucrbgalloc);
/* May need add to the tagdataptr to make sure the next tag
   lands on a four byte boundary
   */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

viewingCondDescTag

Specification Tag Name	viewingCondDescTag
Specification Tag Type	textDescriptionType
ICC Header Tag Name	icSigViewingCondDescTag
ICC Header Tag Type	icTextDescriptionType -> icTextDescription
Signature	vued (76756564h)
Fixed or Variable Length?	variable
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.45
Description	Specifies viewing condition description (Uni and Script code text descriptions)

See the deviceMfgDescTag for samples.

viewingConditionsTag

Specification Tag Name	viewingConditionsTag
Specification Tag Type	viewingConditionsType
ICC Header Tag Name	icSigViewingConditionsTag
ICC Header Tag Type	icViewingConditionType -> icViewingCondition
Signature	view (76696577h)
Fixed or Variable Length?	fixed
Required by Profile Types	
ICC Spec section / more info pages	ICC spec: 6.4.46
Description	Specifies viewing condition parameters (includes illuminant, surround, and standard illuminant as XYZ numbers)

Sample code to read the viewingConditionsTag

```
case 0x76696577L:  /* 'view' */
 printf("signature = 0x%x signatureId = %s, offset = %d size = %d\n",
       tag[i].sig, buf[i], tag[i].offset, tag[i].size);
 fseek(fd, (long) tag[i].offset, 0);
```

```
if (fread(&view, tag[i].size, 1, fd) != 1) {
  printf("error reading file\n");
  exit(1);
}
printf("Viewing conditions type\n");
    printf("Illuminant X=%f, Y=%f, Z=%f\n",
        icfixed2double(view.view.illuminant.X,
                icSigS15Fixed16ArrayType),
        icfixed2double(view.view.illuminant.Y,
                icSigS15Fixed16ArrayType),
        icfixed2double(view.view.illuminant.Z,
                icSigS15Fixed16ArrayType));
    printf("Surround X=%f, Y=%f, Z=%f\n",
        icfixed2double(view.view.surround.X,
                icSigS15Fixed16ArrayType),
        icfixed2double(view.view.surround.Y,
                icSigS15Fixed16ArrayType),
        icfixed2double(view.view.surround.Z,
                icSigS15Fixed16ArrayType));
    printf("StdIlluminant %d\n",
        swap((long)view.view.stdIluminant));

printf("\n\n");
break;
```

Sample code to write the viewingConditionsTag

```
tagdir.sig =  swap((long)0x76696577L);                /*'view'*/
tagdir.offset =  swap((long)tagdataptr);
size = sizeof(tagbase) + sizeof(icViewingCondition);
printf("tag size = %d\n",size);
```

```
    tagdir.size =  swap((long)size);
    fseek(fd, dirptr, SEEK_SET);
    if (fwrite(&tagdir, sizeof(icTag), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }

    dirptr += sizeof(tagdir);              /*keep this pointed to the end of
                                           the tag directory*/
    tagbase.sig =  swap((long)0x76696577L);        /*'view'*/
    fseek(fd, tagdataptr, SEEK_SET);
    if (fwrite(&tagbase, sizeof(icTagBase), 1, fd) != 1) {
       printf("error writing file\n");
       exit(1);
     }
    tagdataptr +=sizeof(tagbase);
    view.view.illuminant.X = double2icfixed (0.279984,
 icSigS15Fixed16ArrayType)
;
    view.view.illuminant.Y = double2icfixed (0.200272,
 icSigS15Fixed16ArrayType)
;
    view.view.illuminant.Z = double2icfixed (0.840454,
 icSigS15Fixed16ArrayType)
;

    view.view.surround.X = double2icfixed (0.9856, icSigS15Fixed16ArrayType);
    view.view.surround.Y = double2icfixed (0.9777, icSigS15Fixed16ArrayType);
    view.view.surround.Z = double2icfixed (0.9712, icSigS15Fixed16ArrayType);

    view.view.stdIlluminant =  swap((long)2);    /*icIlluminant65*/
```

```
fseek(fd, tagdataptr, SEEK_SET);
if (fwrite(&view.view, size - sizeof(tagbase), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }

/*update the final size of profile in the header*/
tagdataptr += size - sizeof(tagbase);
fseek(fd, 0, SEEK_SET);
tempdataptr = swap((long)tagdataptr);
if (fwrite(&tempdataptr, sizeof(icUInt32Number), 1, fd) != 1) {
   printf("error writing file\n");
   exit(1);
  }
/* May need add to the tagdataptr to make sure the next tag
  lands on a four byte boundary
  */
divt = div(tagdataptr, 4);
tagdataptr += divt.rem;
```

CHAPTER 4

ICC Profile Processing Models

Some devices may need only a matrix and simple lookup tables to accurately describe their conversion to and from the ICC profile connection space (PCS). The profiles may use the model that I will call the “Shaper/Matrix Model” (a term adopted from George Pawle). This model is accomplished using the 3 1-dimensional tone reproduction curve (TRC) tables and the 3X3 matrix of colorants described by the ICC profile format. The three components of the input data are processed through their respective TRC table, then through the 3X3 matrix for the resulting common colorspace (XYZ or LAB). See Figure 3, “Shaper/Matrix Processing Model,” on page 151.

For devices whose color conversions cannot be accomplished accurately with the shaper/matrix model, the ICC profile format provides tags to allow a more complex conversion we call the “Matrix/Tabulated Function Processing Model “(another term adopted from George Pawle). See Figure 8, “Matrix/Tabulated Function Processing Model,” on page 158.

The profile builder must determine which model will be sufficient for the device they are modeling. In general, the shaper/matrix model is simple, creates a smaller profile, provides fast processing, and has no ambiguities (a problem to be presented later in this chapter). However, there is little flexibility. This model may be appropriate for monitors and linear devices.

The matrix/tabulated function model is complicated and can result in very large profiles. It can be more ambiguous in the interpolation of intermediate values and it can be slower to process data unless it is accelerated by special hardware. However, it is very flexible and can model any device to a certain amount of high accuracy, depending upon the algorithms used.

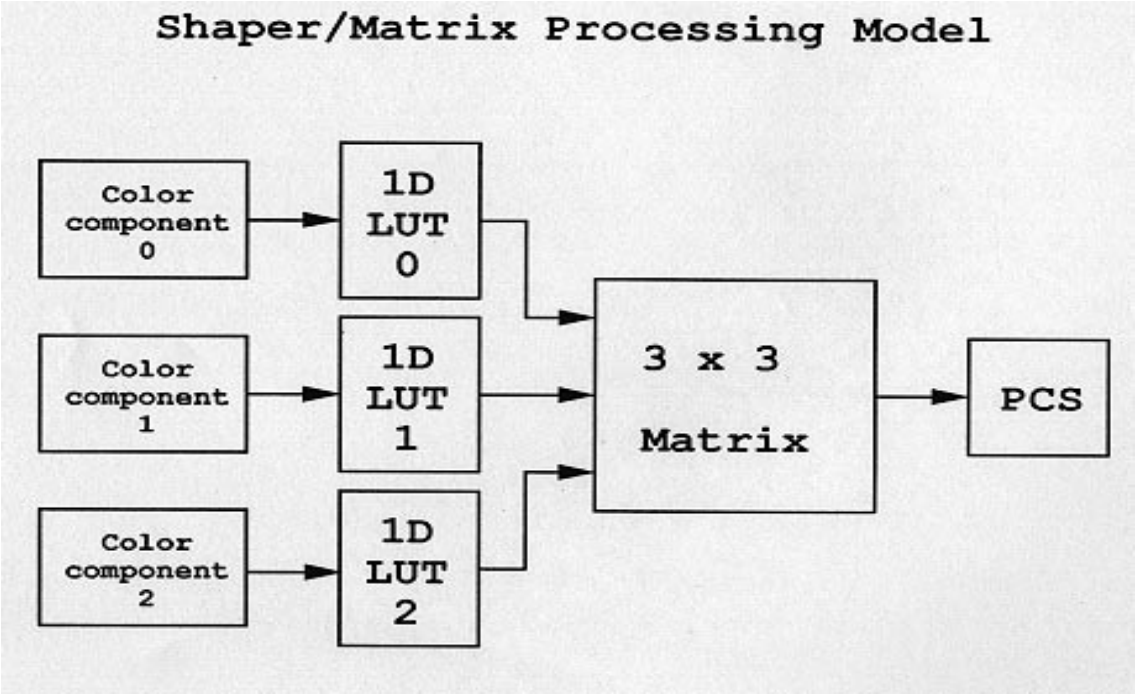


FIGURE 3. Shaper/Matrix Processing Model

Shaper/matrix model

Figure 3, “Shaper/Matrix Processing Model,” on page 151 depicts the data passing through the shaper/matrix processing model. The 3 color components are processed through the device’s tonal response curves, then through the 3x3 color conversion matrix into the PCS. The curves or the matrix may also account for a white point conversion.

The 1D lookup tables are “curv” data types whose input is applied to the 3X3 matrix. One option for the curve type contents is to supply just one value - a gamma value - which is encoded as a U8Fixed8Number. For example, a gamma of 2.0 = 2.0*256 = 512 + 0.5 = 512 integer 200 hex. See Figure 4, “Gamma Curve,” on page 153.

These tables more typically contain a set of points representing a curve. For example, to encode 5 points to represent the gamma of 2.0 the equation is y = x**2.0, where 0 <= x <= 1.0.

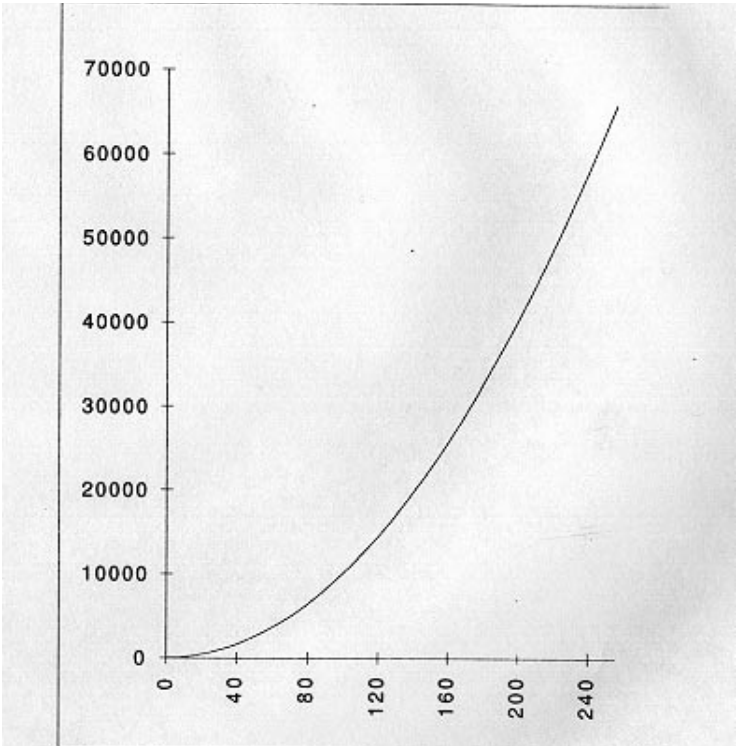


FIGURE 4. Gamma Curve

The 3rd of the 5 points would be encoded as: 2/4**2.0= 0.25 * 65535 = 16383.75 + 0.5 = 16384 integer = 4000 hex. This example may not provide enough data points to avoid interpolation error. See Figure 5, “Gamma Curve represented with 5 points,” on page 154

'curv' table	
Position	Value
0	0
1	4096
2	16384
3	36863
4	65535

8-bit LUT	
Position	Value
0	0
64	4144
128	16223
191	36751
255	65535

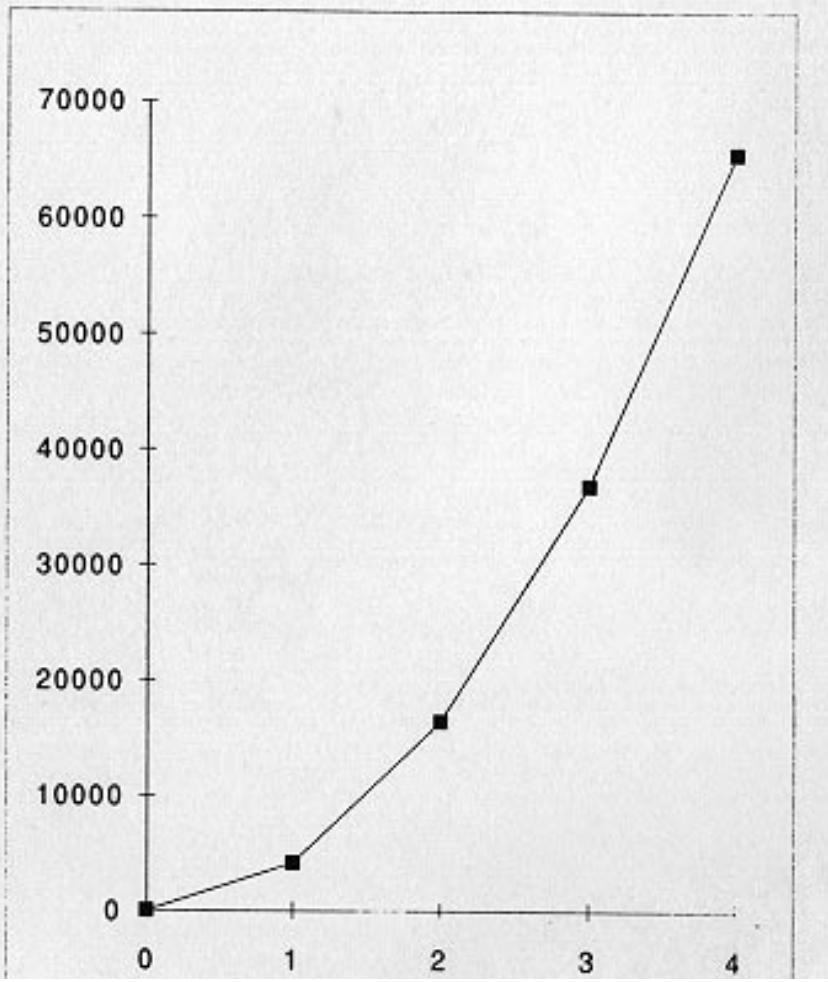


FIGURE 5. Gamma Curve represented with 5 points

In using the shaper/matrix model, one needs to take care to supply enough points in the TRC (1 dimensional lookup table) such that interpolation doesn't introduce noticable errors.

An example of the errors introduced by not using enough points to describe the curve is shown in Figures 4 and 5. The smooth curve is the ideal, the Figure 5 curve is the result of limiting the number of points provided and the resulting interpolated results are shown in the 8-bit LUT table below the intended "curv" table values. Though the interpolation is well behaved, errors are introduced due to the small number of data points and the 8-bit accuracy of the table. High order interpolations may produce better results but may not behave as well.

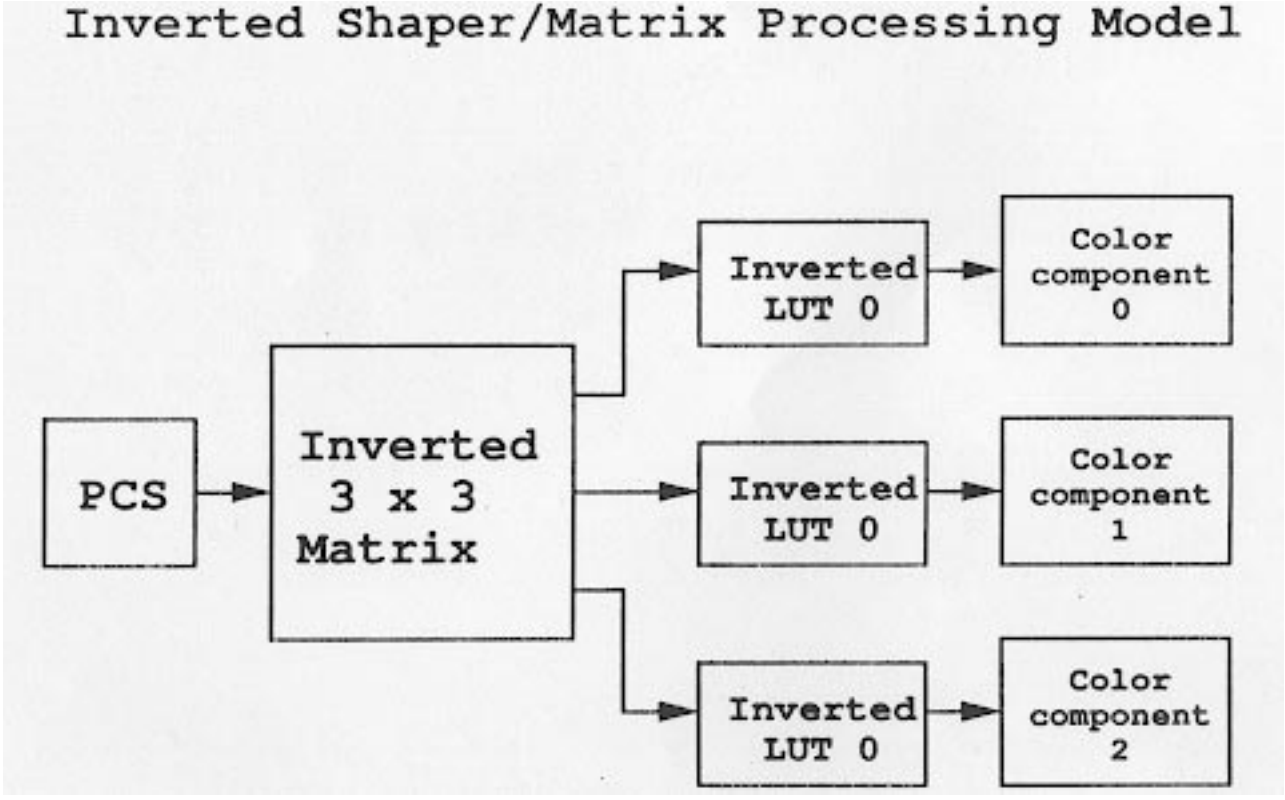


FIGURE 6. Inverted Shaper/Matrix Processing Model

Figure 6, "Inverted Shaper/Matrix Processing Model," on page 155 shows the inverse of the Shaper/Matrix Processing Model. Although the inverse TRC's and matrix are not provided in the profile, the CMM is expected to be able to calculate them. The set of points respresenting the TRC mapping curve must be monotoni-

cally increasing, otherwise an inverse is not predictable. See Figure 7, “Misbehaving curve and its inverse,” on page 156. The first curve may be the one provided in the TRC table, while the second one represents an inverse. If one were to interpolate for the 19000 value in the inverted curve, for example, would one interpolate between the 3rd and 4rd values or the 4th and 5th values? Which is the correct answer?

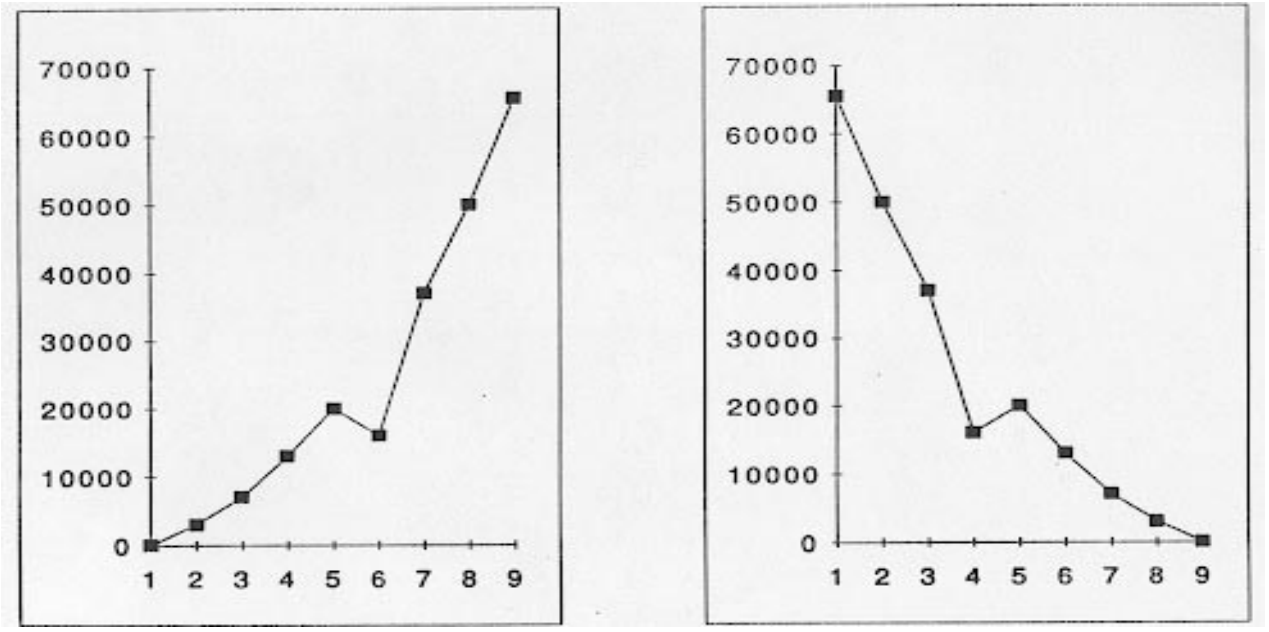


FIGURE 7. Misbehaving curve and its inverse

CLUT model

The tabulated function (CLUT model) in Figure 8, “Matrix/Tabulated Function Processing Model,” on page 158 is a representation of an analytical function in the form of evenly distributed samples of that function. The position in the table corresponds to an input value, but to convert that input value to a table position, one must know the input values corresponding to the beginning and end of the table. Interpo-

lation is usually necessary to determine the function value at arbitrary positions in the table. The simplest is linear interpolation, which is nicely bounded. The tables are used to represent a variety of situations so some scaling assumptions may be needed, however, one must assume that the input values are unsigned and that the input value of 0 maps to the first table position.

The matrix is encoded the same as in the shaper/matrix model. The input tables are sequentially ordered as red, green, and blue. These are dimension independent so that the same table may be used with any size lattice (CLUT). The number of entries in the tables are fixed at 256 for 8-bit CLUTs and allow a range 2-4096 for 16-bit CLUTs.

To convert a lattice (CLUT) entry into an index and interpolant, one must linearly interpolate the entry into the lattice points. (X/entry) = (maximum index)/(maximum entry). The integer part is the index of the base lattice point and the fractional part is the interpolant to the next lattice point. For example: assume one has 3 lattice points, an 8-lut CLUT, and an input of 200. X/200 = 2/255. X = 1.5686.

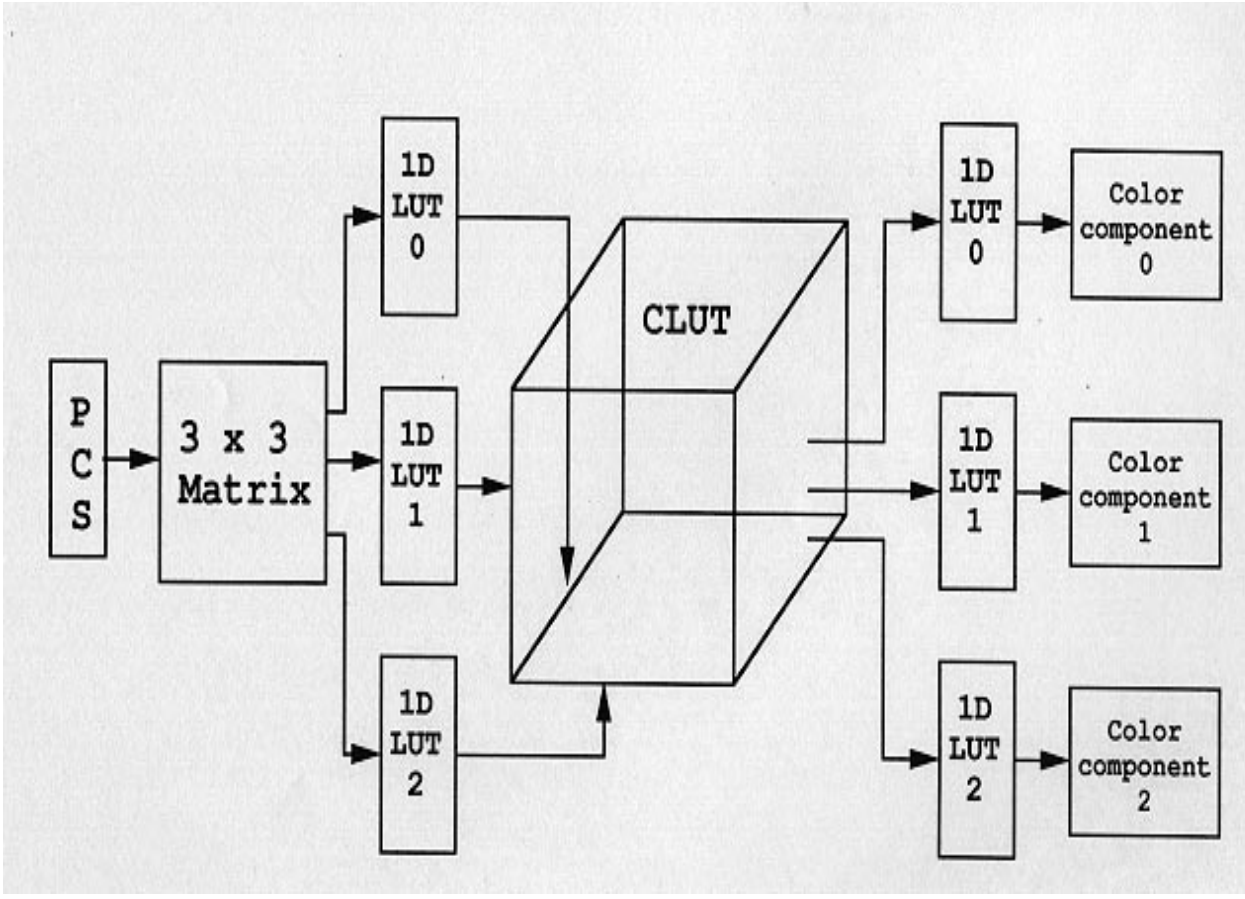


FIGURE 8. Matrix/Tabulated Function Processing Model

The lattice is interleaved storage. The first entry is the output of the first function, the second is the output of the second function, etc. The points are ordered as a multi-dimensional array where the lattice dimension corresponding to the first input varies least rapidly and the dimension corresponding to the last input varies most rapidly.

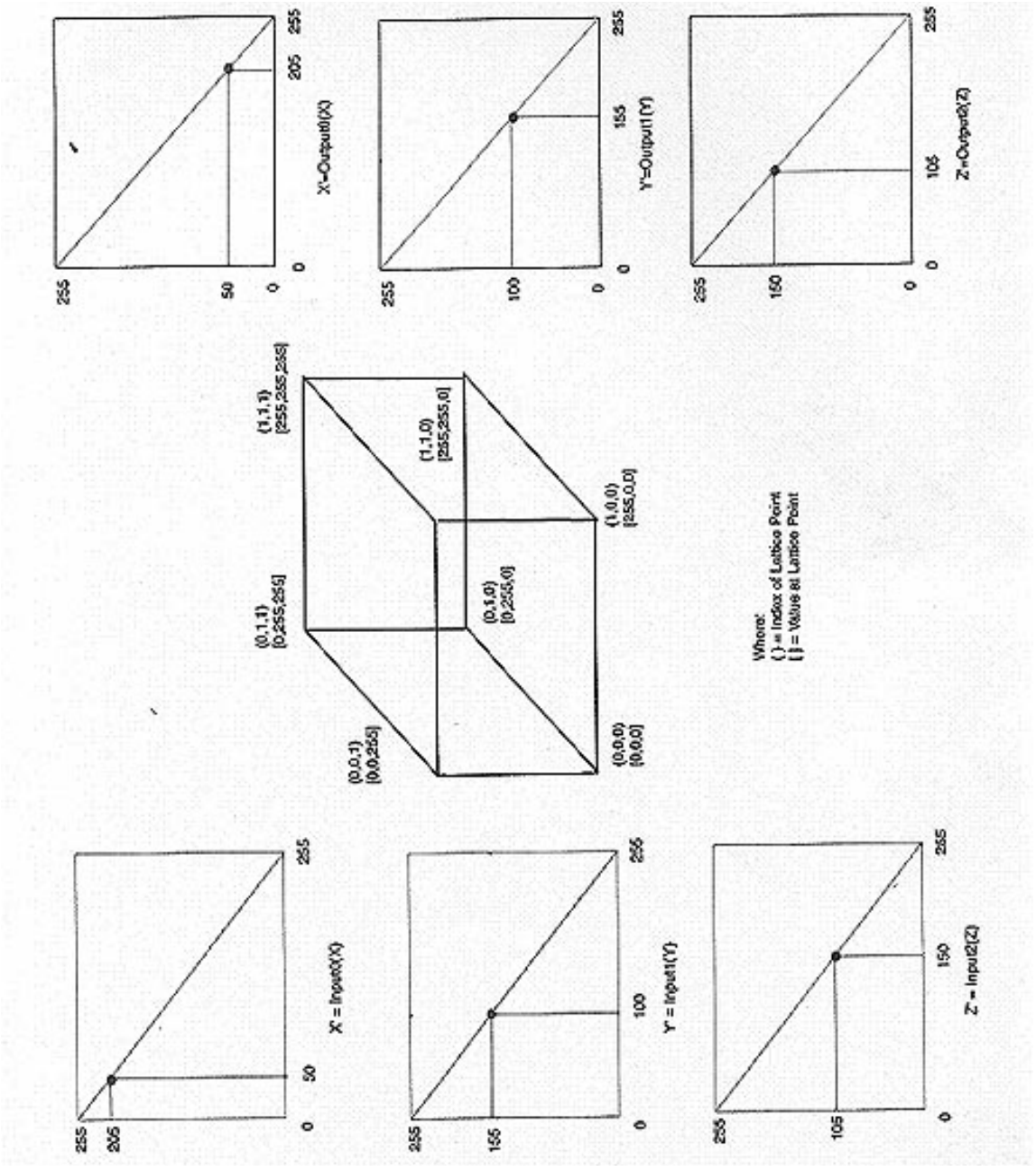

X = Input0(X)
Y = Input1(Y)
Z = Input2(Z)
X'=Output0(X)
Y'=Output1(Y)
Z'=Output2(Z)
(0,0,0) [0,0,0]
(1,0,0) [255,0,0]
(0,1,0) [0,255,0]
(1,1,0) [255,255,0]
(0,0,1) [0,0,255]
(0,1,1) [0,255,255]
(1,1,1) [255,255,255]
Where:
( ) = Index of Lattice Point
[ ] = Value at Lattice Point

FIGURE 9. Data Flow through the CLUT

If there are 2 lattice point in each dimension, the storage appears as follows:

Byte#	lattice indices(x,y,z)
0	0,0,0
1	0,0,1
2	0,1,0
3	0,1,1
4	1,0,0
5	1,0,1
6	1,1,0
7	1,1,1

The output tables are also ordered sequentially. The 8-bit CLUT maps directly to the output table indices. The 16-bit CLUT must be interpolated assuming 4096 entries. CLUT entry 0 maps to output table index 0. CLUT entry 65535 maps to output table index 4095.

The following is an example of numbers flowing through an identity CLUT - see Figure 9, “Data Flow through the CLUT,” on page 160.

Suppose the Input table function is 1-x,

the lattice function is an identity, i.e.

lattice0(x,y,z) = x

lattice1(x,y,z) = y

lattice2(x,y,z) = z

And the Output table function is also 1-x.

Given (x,y,z) = (50, 100, 150)

Input Table (1-x) is: InputX = 205, InputY = 155, InputZ = 105

Lattice: x 205, y 155, z 105

TABLE 1.

Lattice	Index	Interpolant
x	0	(205*1)/255 = 0.80392
y	0	(155*1)/255 = 0.60784
z	0	(105*1)/255 = 0.41176

Output Table (1-x) is: OutputX = 50, OutputY = 100, OutputZ = 150

The series of figures below depict a possible optimization of the CLUT use. The function we will use is represented in Figure 10, “Z = X**1/3 * Y**3 Function Graph,” on page 162.

Note: The graph axles are x increasing to the right, y increasing to the left,and z increasing to the top. The values of x and y are normalized to a range of 0.0-10.0. The graph isn’t perfect but is useful for illustrating the example.

Given the example of:

X = 205

Y = 155

the value of X at position 205 is 205/255*10=8.0392

the value of Y at position 155 is 155/255*10=6.0784

x^1/3 * y^3 = 449.16

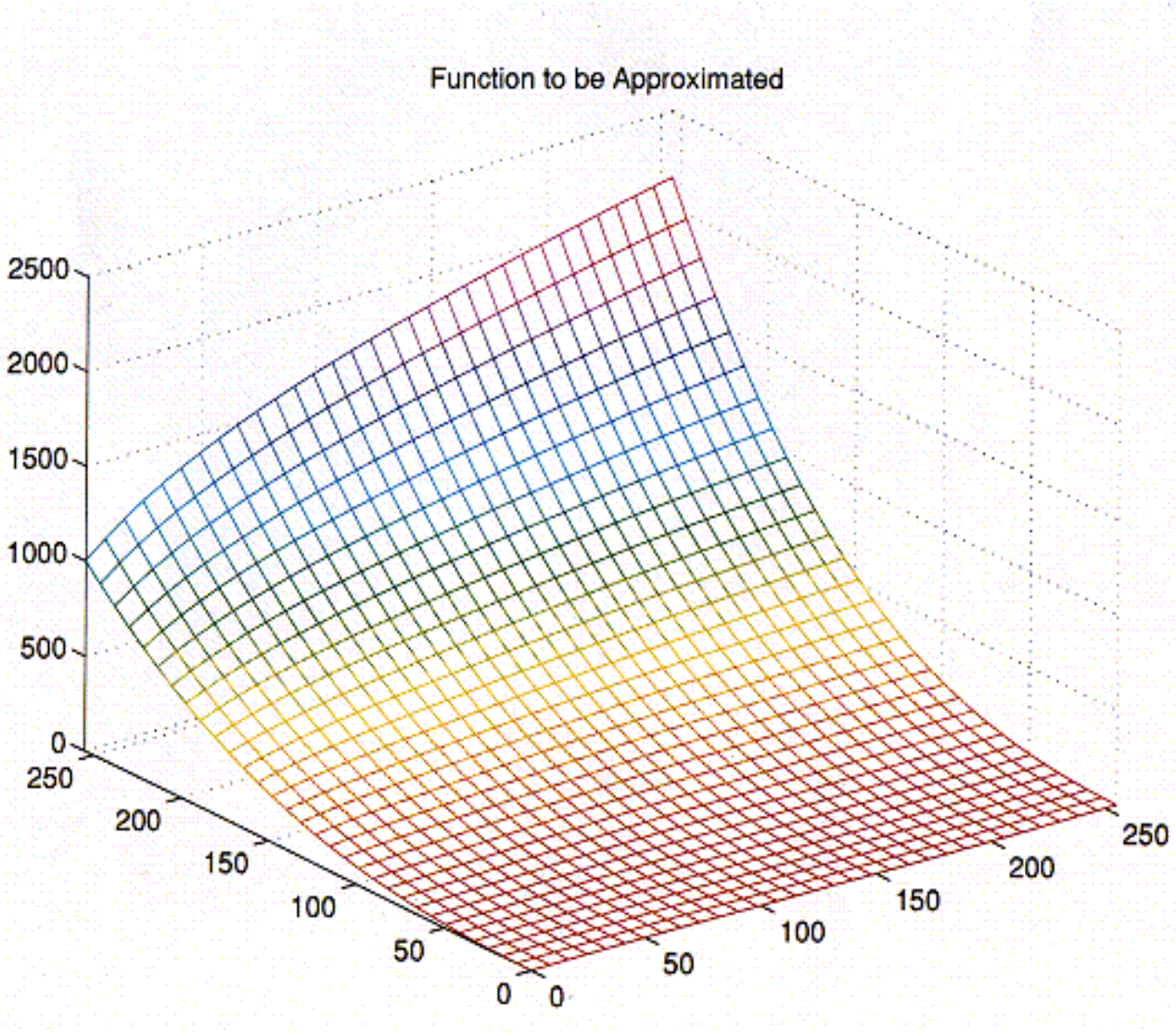


FIGURE 10. Z = X1/3 * Y**3 Function Graph**

To represent this accurately in the table would require a very large number of points. Figure 11, “Z = X**1/3 * Y**3 (represented with sparse grid points),” on page 164 shows the amount of error which can result if one represented this function with a sparse number of grid points.

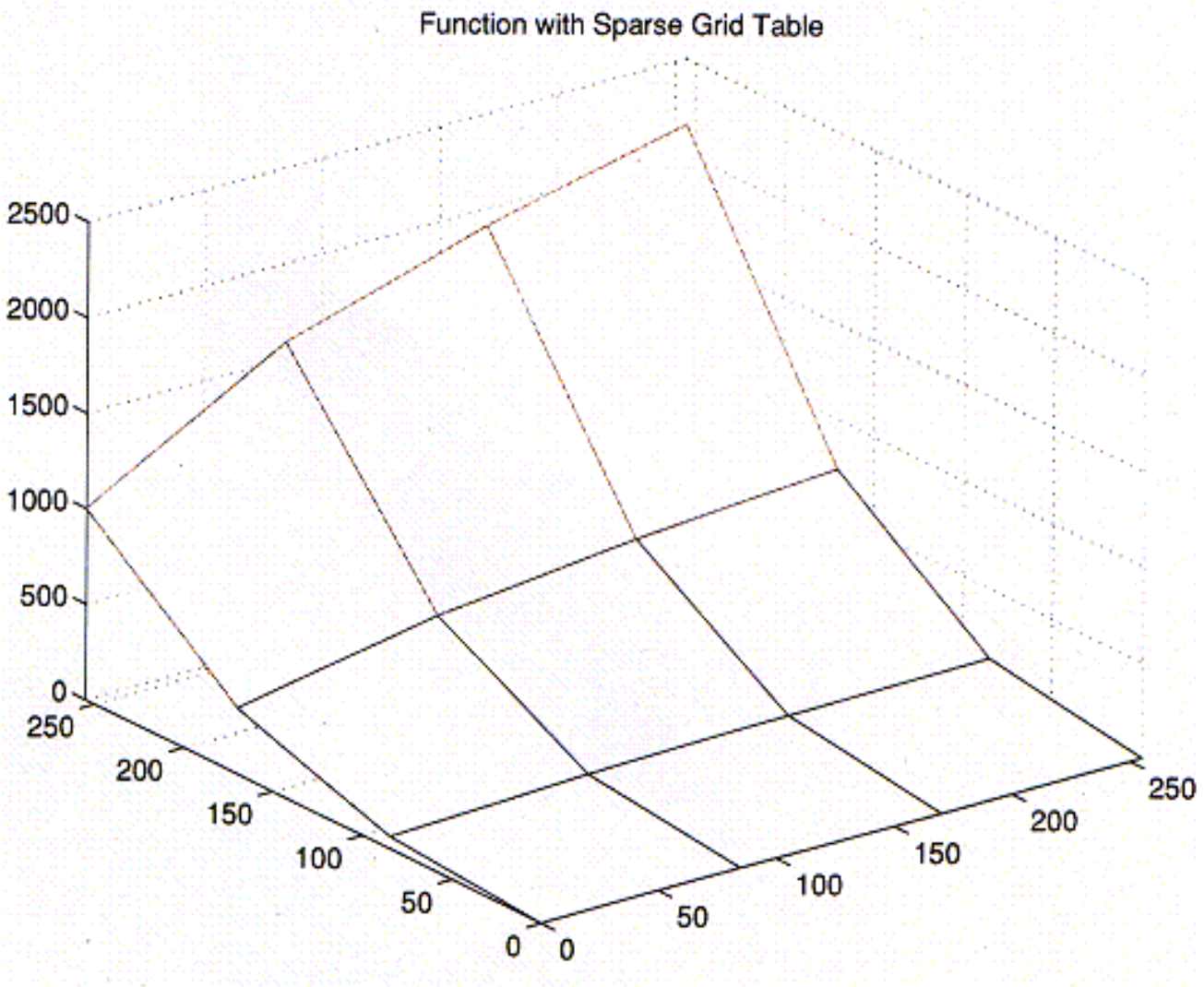


FIGURE 11. Z = X**1/3 * Y**3 (represented with sparse grid points)

To simplify the function and maintain accuracy one could convert the values to log10. The input table and output table could provide the conversion into and out of log10. The graphs of these log conversions are shown in Figure 12, “Into-Log and Outof-Log Curves,” on page 165 and represent the input and output tables, respectively.

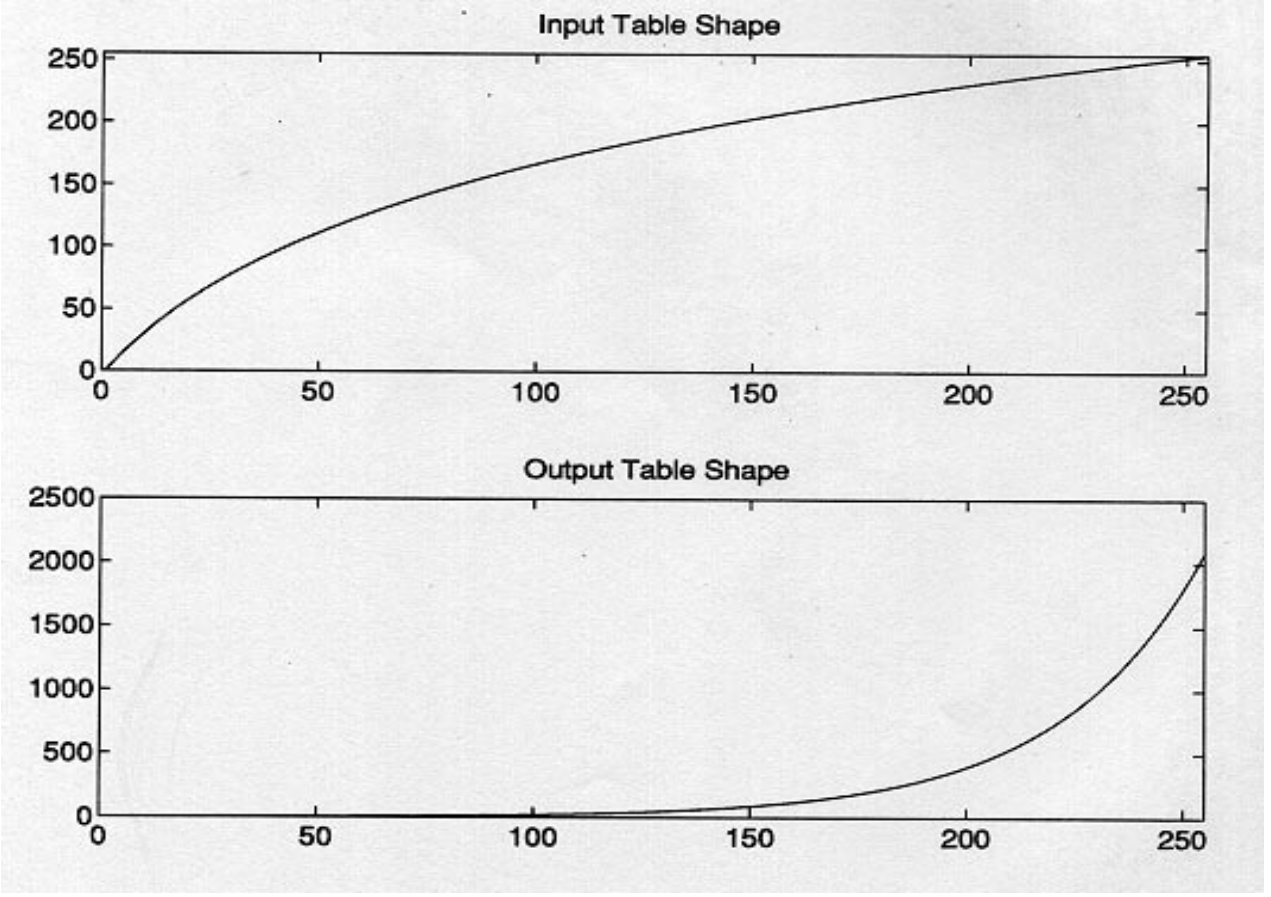


FIGURE 12. Into-Log and Outof-Log Curves

This has the affect of converting the function to z=1/3X + 3y, a nice linear function which can be accurately represented with very few grid points, as show in Figure 13, “Z = X**1/3 * Y**3 (represented as a log function z=1/3X + 3y),” on page 167. This example is used in an AtoB tag in Chapter 6 on dissecting a scanner profile.

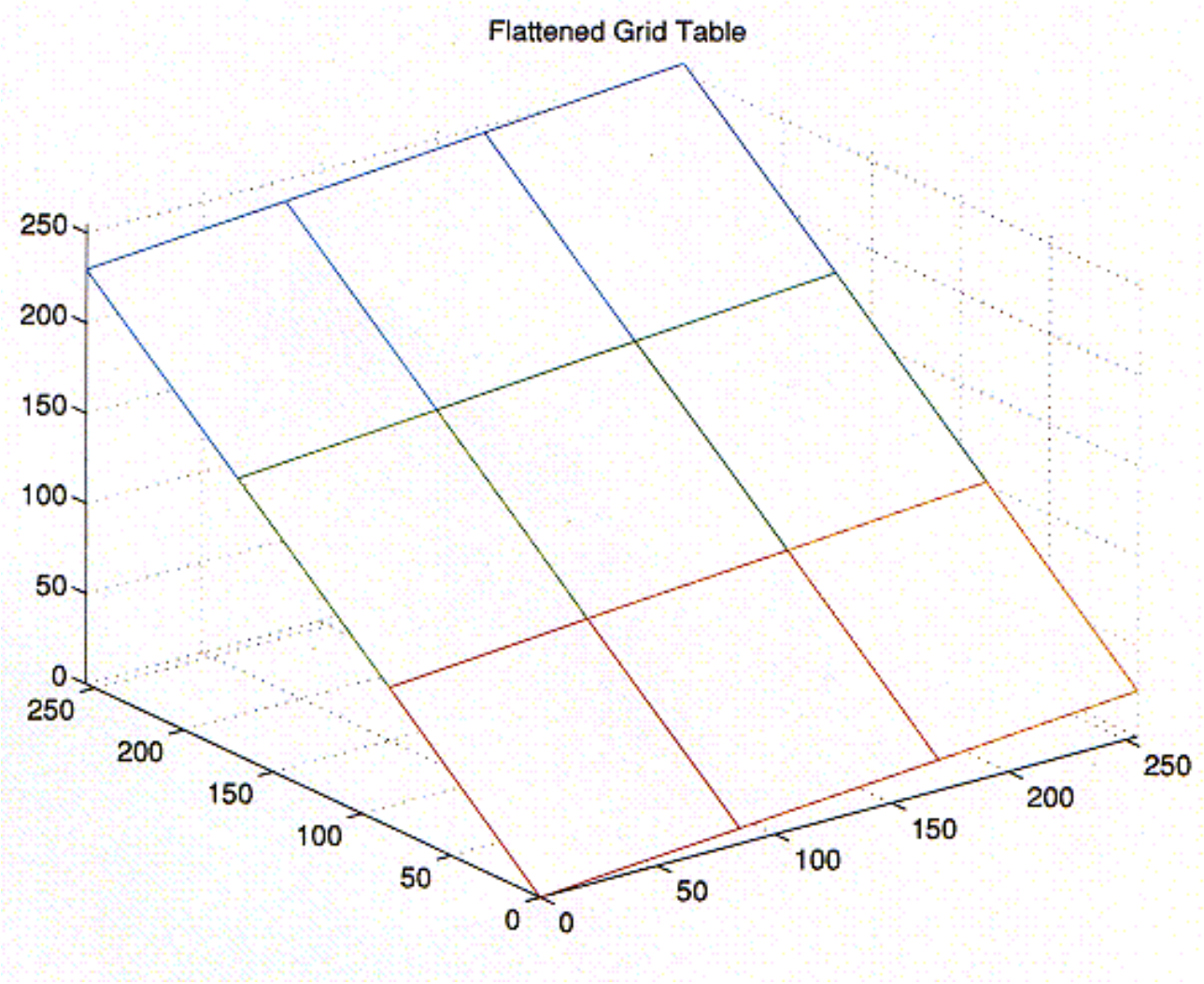


FIGURE 13. Z = X**1/3 * Y**3 (represented as a log function **z=1/3X + 3y)**

CHAPTER 5

Dissecting Display Profiles

There are two types of display profiles which have different required tags. The monochrome display profile has the same required tags the the monochrome input and monchrome output profiles. A sample of this type is shown in the next chapter on input profiles. The other type, dissected below, is the RGB display type.

The hexadecimal file below is a listing of a monitor (display) profile created for testing. The left “count” column is a hexidecimal count of the number of bytes offset, from the beginning of the file, of this line of data. These counters will be used to reference line numbers in the descriptions below. “*” indicates that identical rows have been deleted. These are usually rows of all zeros. The contents of this file is dissected in detail. Later chapters will dissect additional profile types, concentrating on the differences in tags from previous profiles. For reference, the hexidecimal dumps were created with the Unix command "od -X -Ax <filename>". The C program used to decode the profile and print out the contents is located at the same place from which you downloaded this book.

Some of the hex data needs to be converted to more familiiar decimal numbers for explanation. Others, primarily text fields, need to be converted to their ASCII equivalents in order to read them.

All the profile samples in the next few chapters were dumped with icctags.c. This particular sample was created with icctags-mon.c - available at the web site.

RGB Display Profile

This profile contains the required tags for an RGB display profile, plus an exanple of an optional device manufacturer tag(dmnd). The header fields will be shown in detail only for this first profile sample. Subsequent samples are essentially identical except for various signatures.

count hex profile data

0000000 000004ae 4b434d53 21000000 6d6e7472
0000010 52474220 4c616220 07ce0006 0007000f
0000020 002b0038 61637370 53554e57 00000000
0000030 53554e20 31393938 00000001 00000000
0000040 00010000 0000f6d5 00010000 0000d32b
0000050 53554e57 00000000 00000000 00000000
0000060 00000000 00000000 00000000 00000000
*
0000080 0000000a 64657363 000000fc 0000007e
0000090 6258595a 0000017c 00000014 6758595a
00000a0 00000190 00000014 7258595a 000001a4
00000b0 00000014 77747074 000001b8 00000014
00000c0 62545243 000001cc 0000000e 67545243
00000d0 000001dc 00000010 72545243 000001ec
00000e0 0000020c 63707274 000003f8 0000002a
00000f0 646d6e64 00000424 0000008a 64657363
0000100 00000000 00000024 54686973 20697320
0000110 74686520 70726f66 696c6520 64657363
0000120 72697074 696f6e20 74616700 00000000
0000130 00000000 00000000 00000000 00000000
0000140 00000003 39180000 00000000 01040000
0000150 00900000 007e0000 00000001 0000ef7e
0000160 bf58ef7e d0a40000 1fff0000 00000000
0000170 00040000 00000000 00000000 58595a20

0000180 00000000 000047ad 00003345 0000d727
0000190 58595a20 00000000 00003615 000086ae
00001a0 0000010e 58595a20 00000000 00007ac0
00001b0 0000476c 000010dd 58595a20 00000000
00001c0 0000f6d5 00010000 0000d32b 63757276
00001d0 00000000 00000001 02380000 63757276
00001e0 00000000 00000002 0000ffff 63757276
00001f0 00000000 00000100 00000000 00010003
0000200 0006000a 000f0016 001e0027 0031003d
0000210 004a0058 00680079 008c00a0 00b600cd
0000220 00e60100 011c013a 01590179 019c01c0
0000230 01e6020d 02360261 028d02bb 02eb031d
0000240 03510386 03bd03f6 0430046d 04ab04eb
0000250 052d0571 05b705fe 06480693 06e0072f
0000260 078007d3 0828087f 08d80933 098f09ee
0000270 0a4f0ab1 0b160b7c 0be50c50 0cbc0d2b
0000280 0d9c0e0f 0e830efa 0f730fee 106b10eb
0000290 116c11ef 127512fc 13861411 149f152f
00002a0 15c11656 16ec1785 181f18bc 195b19fd
00002b0 1aa01b46 1bed1c97 1d431df2 1ea21f55
00002c0 200a20c1 217b2236 22f423b5 2477253c
00002d0 260226cc 27972865 29352a07 2adb2bb2
00002e0 2c8b2d67 2e442f24 300730eb 31d232bc
00002f0 33a73495 35853678 376d3864 395e3a5a
0000300 3b583c59 3d5c3e61 3f694073 4180428f
0000310 43a044b4 45ca46e2 47fd491b 4a3a4b5c
0000320 4c814da8 4ed14ffd 512c525c 538f54c5
0000330 55fd5738 587559b4 5af65c3a 5d815eca
0000340 60166164 62b56408 655e66b6 6811696e
0000350 6acd6c2f 6d946efb 706571d1 734074b1
0000360 7625779b 79147a8f 7c0d7d8e 7f118096

0000370 821e83a9 853686c6 885889ed 8b848d1e
0000380 8ebb905a 91fc93a0 954796f1 989d9a4b
0000390 9bfd9db0 9f67a120 a2dca49a a65ba81e
00003a0 a9e5abad ad79af47 b117b2eb b4c0b699
00003b0 b874ba52 bc32be16 bffbc1e4 c3cfc5bd
00003c0 c7adc9a0 cb96cd8e cf89d187 d387d58a
00003d0 d790d999 dba4ddb2 dfc2e1d5 e3ebe604
00003e0 e81fea3d ec5eee81 f0a7f2d0 f4fcf72a
00003f0 f95bfb8f fdc5ffff 74657874 00000000
0000400 436f7079 72696768 743a2020 53756e20
0000410 4d696372 6f737973 74656d73 20313939
0000420 38000000 64657363 00000000 00000030
0000430 54686973 20697320 74686520 64657669
0000440 6365206d 616e7566 61637475 72657220
0000450 64657363 72697074 696f6e20 74616700
0000460 00000000 00000000 00000000 00000000
0000470 00003f80 00000003 39180000 00000000
0000480 042c0000 00fc0000 008a0000 00000001
0000490 0000ef7e bf58ef7e d0a40000 1fff0000
00004a0 01000000 00040000 00000000 00000000
00004ae

The first 6 printed (and two deleted) lines constitute the profile header. Each field is shown below in hex and in the decoded decimal or ASCII (shown as hex/decimal). The "size in bytes" are in hex.

line at byte 0

Size in bytes: 4ae, decimal = 1198 (size of the profile)

CMM Id: 4b434d53, ascii = KCMS

Version number: 21,

Field meaning = major version 2 and minor version 1

deviceClass: 6d6e7472, ascii = display

lines at bytes 10 and 20

colorspace: 52474220, ascii = RGB

profile connection space: 4c616220, ascii = Lab

date: hex / decimal = 07ce / 1998, 0006 / 6, 0007 / 7 = (07/06/1998)

time: hex / decimal = 0009/ 9, 0031 / 51, 0032/ 50 = (15:43:56)

magic number: 61637370, ascii = acsp

platform: 53554e57, ascii = SUNW (Solaris)

bit flags: 00, meaning = Non-embedded profile

OK to strip embedded profile out and use independently

line at byte 30

manufacturer: 53554e20 ascii = SUN

model: 31393938, ascii = 1998

device attributes: 00000001,

meaning = tranparency, glossy, positive, color

line at byte 40

rendering intent: 1, meaning = Relative Colorimetric

Illuminant (s15Fixed16Number)

0000f6d5, = 63189/65536 = 0.964188(X)

00010000, = 65536/65536 = 1.000000 (Y)

0000d32c, = 54060/65536 = 0.824875 (Z)

creator: 64657363 ascii = SUNW

remaining lines up to byte 80 are reserved, set to zero

The remainder of the file's lines constitute the profile tag directory and tag data. Each of the tags in the directory consist of three 32-bit words; the tag signature, an offset into the file pointing to the tag's data, and the number of bytes of tag data to read at that offset. The first 4 underlined bytes at byte 80 are the number of tags in the file. After decoding a tag in the directory, we will follow the offset to the actual data, and decode the tag contents.

line at byte 128

Number of tags = a/10
signature = 0x64657363 signatureId = desc, offset = fc/252 size = 7e/126
Scan down to the underlined word at byte fc
64657363 in ascii = text. This is a text description type.
The next word is reserved = 0. The next is 24/36, which indicates that the text string is 36 bytes long. Following that is an ASCII text string which decodes as: "This is the profile description tag"
What follows is the rest of this rather complicated tag - the Unicode and Scriptcode equivalents to the ASCII string. This profile does not include these strings, but must account for space for them. So the next 78 bytes should be skipped.
Note that this tag does not end on a 4 byte boundary. Two bytes are padded with 0 so that the next tag will begin on a 4 byte boundary.

Back up in the tag directory, the next tag is:
signature = 0x6258595a signatureId = bXYZ, offset = 17c/380 size = 14/20
Scan down to the underlined word at byte 17c/380.
58595a20 in ascii XYZ . This is an XYZ type.
The next word is reserved = 0, and the next 3 4-byte words are the XYZ values of the blue colorant tag in U16Fixed16Number format (see Appendix A). Decoded, these values are:
X=0.279984, Y=0.200272, Z=0.840439

The same procedure can be followed to trace the green and red colorant tags.

signature = 0x6758595a signatureId = gXYZ, offset = 190/400 size = 14/20
XYZ type
X=0.211258, Y=0.526093, Z=0.004120

signature = 0x7258595a signatureId = rXYZ, offset = 1a4/420 size = 14/20
XYZ type
X=0.479492, Y=0.278992, Z=0.0658722

signature = 0x77747074 signatureId = wtpt, offset = 1b8/440 size = 14/20
XYZ type
X=0.964188, Y=1.000000, Z=0.824875

For the blue, green, and red tone reproduction curves, I am showing an example of each of the three possible formats. A profile would not normally be created this way, however. The bTRC has only one entry, so it is expected to be a U8Fixed8Number containing a gamma value. The gTRC has two entries, so is expect to contain the UInt16Number for the minimum and maximun values for the curve. The CMM would interpret this as a ramp for all other values. The rTRC contains a full curve of 256 values - the CMM has no interpolation to do in this case. If the curve contains > 2 and < 256 values, then the CMM is expected to interpolate to map the missing values.

signature = 0x62545243 signatureId = bTRC, offset = 1cc/460 size = e/14
Scan down to the underlined word at byte 1cc/460
Curve type, curve count = 1
Count = 1 Curve is a gamma of 2.21875

signature = 0x67545243 signatureId = gTRC, offset = 1dc/476 size = 10/16
Curve type, curve count = 2
Count = 2 Line Start = 0 End = 65535

signature = 0x72545243 signatureId = rTRC, offset = 1ec/492size = 20c/524
Curve type, curve count = 256

0 0 1 3 6 10 15 22 30 39 49 61 74 88 104 121 140
160 182 205 230 256 284 314 345 377 412 448 486 525 566 609 653
699 747 797 849 902 957 1014 1072
1133 1195 1259 1325 1393 1463 1534 1608
1683 1760 1839 1920 2003 2088 2175
2264 2355 2447 2542 2639 2737 2838 2940 3045
3152 3260 3371 3484 3599 3715 3834
3955 4078 4203 4331 4460 4591 4725 4860 4998
5137 5279 5423 5569 5718 5868 6021
6175 6332 6491 6653 6816 6982 7149 7319 7491
7666 7842 8021 8202 8385 8571 8758
8948 9141 9335 9532 9730 9932 10135 10341 10549
10759 10971 11186 11403 11623 11844 12068
12295 12523 12754 12988 13223 13461 13701 13944 14189
14436 14686 14938 15192 15449 15708 15969
16233 16499 16768 17039 17312 17588 17866 18146 18429
18715 19002 19292 19585 19880 20177 20477
20780 21084 21391 21701 22013 22328 22645 22964 23286
23610 23937 24266 24598 24932 25269 25608
25950 26294 26641 26990 27341 27695 28052 28411 28773
29137 29504 29873 30245 30619 30996 31375 31757
32142 32529 32918 33310 33705 34102 34502 34904
35309 35716 36126 36539 36954 37372 37792 38215
38641 39069 39499 39933 40368 40807 41248 41692
42138 42587 43038 43493 43949 44409 44871 45335
45803 46272 46745 47220 47698 48178 48662 49147
49636 50127 50621 51117 51616 52118 52622 53129
53639 54151 54666 55184 55705 56228 56754 57282
57813 58347 58884 59423 59965 60510 61057 61607
62160 62716 63274 63835 64399 64965 65535

signature = 0x63707274 signatureId = cprt, offset = 3f8/1016 size = 2a/42
Text type
Copyright: Sun Microsystems 1998

signature = 0x646d6e64 signatureId = dmnd, offset = 424/1060 size = 8a/138
Text description
This is the device manufacturer description tag

CHAPTER 6

Dissecting Input Profiles

Input profiles may be of several flavors. The 3-component shaper/matrix-based input profile requires the same tags as the RGB display profile in the previous chapter. The only difference you would see in my sample profile would be the signatures identifying the profile as an input scanner or digital camera profile. This chapter includes samples of a monochrome and an n-component lut-based input profile.

Monochrome Input Profile

Besides the required tags, this profile includes sample tags for calibration date/time (calt) and target (targ).

count hex profile data

0000000 000001e6 4b434d53 21000000 73636e72
0000010 52474220 4c616220 07ce0006 00090010
0000020 00210024 61637370 53554e57 00000000
0000030 53554e20 31393938 00000002 00000000
0000040 00010000 0000f6d5 00010000 0000d32b

0000050 53554e57 00000000 00000000 00000000
0000060 00000000 00000000 00000000 00000000
*
0000080 00000006 77747074 000000cc 00000014
0000090 6b545243 000000e0 0000002c 63616c74
00000a0 0000010c 00000014 74617267 00000120
00000b0 0000001a 63707274 0000013c 0000002a
00000c0 64657363 00000168 0000007e 58595a20
00000d0 00000000 0000f6d5 00010000 0000d32b
00000e0 63757276 00000000 00000010 000001c0
00000f0 061b0caa 15421fc5 2c1c3a37 4a095b84
0000100 6ea18356 999ab168 cab9e587 64617461
0000110 00000000 07ce0006 00090010 00210024
0000120 74657874 00000000 414e5349 20495438
0000130 2e372f31 2d313939 33000000 74657874
0000140 00000000 436f7079 72696768 743a2020
0000150 53756e20 4d696372 6f737973 74656d73
0000160 20313939 38000000 64657363 00000000
0000170 00000024 54686973 20697320 74686520
0000180 70726f66 696c6520 64657363 72697074
0000190 696f6e20 74616700 00000000 00000000
00001a0 00000000 00000000 00003f70 f0f10003
00001b0 2b080000 00000000 01700000 00cc0000
00001c0 007e0000 00000001 0000ef7e bf58ef7e
00001d0 d0a40000 1fff0000 00100000 00040000
00001e0 00000000 00000000 00000000 00000000
*
00001f2

Size in bytes = 1e6/486

CMM Id = KCMS
Version number = 0x21000000
deviceClass = input
colorspace = RGB
profile connection space = Lab
date = 9/6/1998, time = 16:33:36
magic number = acsp
platform = Solaris
Non-embedded profile
OK to strip embedded profile out and use independently
manufacturer = SUN
model = 1998
Attributes = reflective, matte, positive, color
rendering intent = Relative Colorimetric
Illuminat X=0.964188 Y=1.000000 Z=0.824875
creator = SUNW

Number of tags in this profile = 6
signature = 0x77747074 signatureId = wtpt, offset = ce/204 size = 14/20
XYZ type
X=0.964188, Y=1.000000, Z=0.824875

signature = 0x6b545243 signatureId = kTRC, offset = e0/224 size = 2c/44
Curve type, curve count = 16
0 448 1563 3242 5442 8133 11292 14903 18953 23428
28321 33622 39322 45416 51897 58759

signature = 0x63616c74 signatureId = calt, offset = 10c/268 size = 14/20
Date type
Date = 9/6/1998 Time = 16:33:36

signature = 0x74617267 signatureId = targ, offset = 120/288 size = 1a/26

Text type

ANSI IT8.7/1-1993

signature = 0x63707274 signatureId = cprt, offset = 13c/316 size = 2a/2

Text type

Copyright: Sun Microsystems 1998

signature = 0x64657363 signatureId = desc, offset = 168/360 size = 7e/126

Text description

This is the profile description tag

N-Component LUT-based Input Profile

This profile includes the required tags. Only the AtoB0 tag is required and not its inverse (BtoA0) because it is deemed unlikely that one would need to provide a preview capability for a scanner. That capability is normally only needed to preview an output (printer) result on a display.

This particular example is also illustrated in Chapter 4, the CLUT section. The tag's matrix is an identity. The 3 input tables convert the input to natural log,. The Clut table implements the function z=x**1/3 + y**3, however, since the values have been converted to log, the function becomes a linear table, z=x/3 + 3y. One needs the minimum number of points for interpolation in this case. The 1 channel output lut for this tag converts the values from log back to their normal range.

count hex profile data

0000000 000005ae 4b434d53 21000000 73636e72

0000010 52474220 58595a20 07ce0006 000d000f

0000020 00280013 61637370 53554e57 00000000

0000030 53554e20 31393938 00000001 00000000

0000040 00010000 0000f6d5 00010000 0000d32b

0000050 53554e57 00000000 00000000 00000000
0000060 00000000 00000000 00000000 00000000
*
0000080 00000004 77747074 000000b4 00000014
0000090 41324230 000000c8 0000043c 63707274
00000a0 00000504 0000002a 64657363 00000530
00000b0 0000007e 58595a20 00000000 0000f6d5
00000c0 00010000 0000d32b 6d667431 00000000
00000d0 03010200 00010000 00000000 00000000
00000e0 00000000 00010000 00000000 00000000
00000f0 00000000 00010000 ffffdfcc bfb4aca5
0000100 9f999590 8c898582 7f7c7a77 7572706e
0000110 6c6a6967 65646261 5f5e5c5b 5a585756
0000120 55545352 504f4e4d 4c4c4b4a 49484746
0000130 45454443 42414140 3f3f3e3d 3c3c3b3a
0000140 3a393938 37373636 35343433 33323231
0000150 3130302f 2f2e2e2d 2d2c2c2b 2b2a2a29
0000160 29282828 27272626 26252524 24242323
0000170 22222221 21202020 1f1f1f1e 1e1e1d1d
0000180 1d1c1c1c 1b1b1b1a 1a1a1919 19181818
0000190 17171717 16161615 15151514 14141313
00001a0 13131212 12121111 11101010 100f0f0f
00001b0 0f0e0e0e 0e0d0d0d 0d0c0c0c 0c0c0b0b
00001c0 0b0b0a0a 0a0a0909 09090908 08080808
00001d0 07070707 06060606 06050505 05050404
00001e0 04040403 03030303 02020202 02020101
00001f0 01010100 00000000 ffffdfcc bfb4aca5
0000200 9f999590 8c898582 7f7c7a77 7572706e
0000210 6c6a6967 65646261 5f5e5c5b 5a585756
0000220 55545352 504f4e4d 4c4c4b4a 49484746
0000230 45454443 42414140 3f3f3e3d 3c3c3b3a

0000240 3a393938 37373636 35343433 33323231
0000250 3130302f 2f2e2e2d 2d2c2c2b 2b2a2a29
0000260 29282828 27272626 26252524 24242323
0000270 22222221 21202020 1f1f1f1e 1e1e1d1d
0000280 1d1c1c1c 1b1b1b1a 1a1a1919 19181818
0000290 17171717 16161615 15151514 14141313
00002a0 13131212 12121111 11101010 100f0f0f
00002b0 0f0e0e0e 0e0d0d0d 0d0c0c0c 0c0c0b0b
00002c0 0b0b0a0a 0a0a0909 09090908 08080808
00002d0 07070707 06060606 06050505 05050404
00002e0 04040403 03030303 02020202 02020101
00002f0 01010100 00000000 ffffdfcc bfb4aca5
0000300 9f999590 8c898582 7f7c7a77 7572706e
0000310 6c6a6967 65646261 5f5e5c5b 5a585756
0000320 55545352 504f4e4d 4c4c4b4a 49484746
0000330 45454443 42414140 3f3f3e3d 3c3c3b3a
0000340 3a393938 37373636 35343433 33323231
0000350 3130302f 2f2e2e2d 2d2c2c2b 2b2a2a29
0000360 29282828 27272626 26252524 24242323
0000370 22222221 21202020 1f1f1f1e 1e1e1d1d
0000380 1d1c1c1c 1b1b1b1a 1a1a1919 19181818
0000390 17171717 16161615 15151514 14141313
00003a0 13131212 12121111 11101010 100f0f0f
00003b0 0f0e0e0e 0e0d0d0d 0d0c0c0c 0c0c0b0b
00003c0 0b0b0a0a 0a0a0909 09090908 08080808
00003d0 07070707 06060606 06050505 05050404
00003e0 04040403 03030303 02020202 02020101
00003f0 01010100 00000000 0003fd00 0003fd00
0000400 fff9f4ee e9e4dfda d6d1cdc8 c4c0bcb8
0000410 b4b0aca8 a5a19e9a 9794908d 8a878481
0000420 7f7c7977 74726f6d 6a686664 615f5d5b

0000430 59575554 52504e4d 4b494846 45434240
0000440 3f3e3c3b 3a383736 35343331 302f2e2d
0000450 2c2b2a29 29282726 25242423 22212120
0000460 1f1e1e1d 1c1c1b1b 1a191918 18171716
0000470 16151514 14141313 12121111 11101010
0000480 0f0f0f0e 0e0e0d0d 0d0c0c0c 0c0b0b0b
0000490 0b0a0a0a 0a090909 09090808 08080808
00004a0 07070707 07070606 06060606 06050505
00004b0 05050505 05040404 04040404 04040404
00004c0 03030303 03030303 03030303 03020202
00004d0 02020202 02020202 02020202 02020201
00004e0 01010101 01010101 01010101 01010101
00004f0 01010101 01010101 01010101 01010100
0000500 00000000 74657874 00000000 436f7079
0000510 72696768 743a2020 53756e20 4d696372
0000520 6f737973 74656d73 20313939 38000000
0000530 64657363 00000000 00000024 54686973
0000540 20697320 74686520 70726f66 696c6520
0000550 64657363 72697074 696f6e20 74616700
0000560 00000000 00000000 00000000 00000000
0000570 0000437f 0000bcb2 243c0003 46280000
0000580 00000000 05380000 00b40000 007e0000
0000590 00000001 0000ef7e bf580000 01000000
00005a0 1fff0000 00030000 04080000 00000000
00005b0 00000000 00000000 00000000
00005ba

Size in bytes = 5ae/1454
CMM Id = KCMS
Version number = 0x21000000
deviceClass = input

colorspace = RGB
profile connection space = XYZ
date = 13/6/1998, time = 15:40:19
magic number = acsp
platform = Solaris
Non-embedded profile
OK to strip embedded profile out and use independently
manufacturer = SUN
model = 1998
Attributes = transparent, glossy, positive, color
rendering intent = Relative Colorimetric
Illuminat X=0.964188 Y=1.000000 Z=0.824875
creator = SUNW

Number of tags in this profile = 4
signature = 0x77747074 signatureId = wtpt, offset = b4/180 size = 14/20
XYZ type
X=0.964188, Y=1.000000, Z=0.824875

signature = 0x41324230 signatureId = A2B0, offset = c8/200 size = 43c/1084
8 bit lut type
Lut 8 type
3 1 2
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
input tables: #channels * 256 = 768
255 255 223 204 191 180 172 165 159
153 149 144 140 137 133 130 127 124
122 119 117 114 112 110 108 106 105
103 101 100 98 97 95 94 92 91

90 88 87 86 85 84 83 82 80
79 78 77 76 76 75 74 73 72
71 70 69 69 68 67 66 65 65
64 63 63 62 61 60 60 59 58
58 57 57 56 55 55 54 54 53
52 52 51 51 50 50 49 49 48
48 47 47 46 46 45 45 44 44
43 43 42 42 41 41 40 40 40
39 39 38 38 38 37 37 36 36
36 35 35 34 34 34 33 33 32
32 32 31 31 31 30 30 30 29
29 29 28 28 28 27 27 27 26
26 26 25 25 25 24 24 24 23
23 23 23 22 22 22 21 21 21
21 20 20 20 19 19 19 19 18
18 18 18 17 17 17 16 16 16
16 15 15 15 15 14 14 14 14
13 13 13 13 12 12 12 12 12
11 11 11 11 10 10 10 10 9
9 9 9 9 8 8 8 8 8
7 7 7 7 6 6 6 6 6
5 5 5 5 5 4 4 4 4
4 3 3 3 3 3 2 2 2
2 2 2 1 1 1 1 1 0
0 0 0 0 255 255 223 204 191
180 172 165 159 153 149 144 140 137
133 130 127 124 122 119 117 114 112
110 108 106 105 103 101 100 98 97
95 94 92 91 90 88 87 86 85
84 83 82 80 79 78 77 76 76
75 74 73 72 71 70 69 69 68

67 66 65 65 64 63 63 62 61
60 60 59 58 58 57 57 56 55
55 54 54 53 52 52 51 51 50
50 49 49 48 48 47 47 46 46
45 45 44 44 43 43 42 42 41
41 40 40 40 39 39 38 38 38
37 37 36 36 36 35 35 34 34
34 33 33 32 32 32 31 31 31
30 30 30 29 29 29 28 28 28
27 27 27 26 26 26 25 25 25
24 24 24 23 23 23 23 22 22
22 21 21 21 21 20 20 20 19
19 19 19 18 18 18 18 17 17
17 16 16 16 16 15 15 15 15
14 14 14 14 13 13 13 13 12
12 12 12 12 11 11 11 11 10
10 10 10 9 9 9 9 9 8
8 8 8 8 7 7 7 7 6
6 6 6 6 5 5 5 5 5
4 4 4 4 4 3 3 3 3
3 2 2 2 2 2 2 1 1
1 1 1 0 0 0 0 0 255
255 223 204 191 180 172 165 159 153
149 144 140 137 133 130 127 124 122
119 117 114 112 110 108 106 105 103
101 100 98 97 95 94 92 91 90
88 87 86 85 84 83 82 80 79
78 77 76 76 75 74 73 72 71
70 69 69 68 67 66 65 65 64
63 63 62 61 60 60 59 58 58
57 57 56 55 55 54 54 53 52

52 51 51 50 50 49 49 48 48
47 47 46 46 45 45 44 44 43
43 42 42 41 41 40 40 40 39
39 38 38 38 37 37 36 36 36
35 35 34 34 34 33 33 32 32
32 31 31 31 30 30 30 29 29
29 28 28 28 27 27 27 26 26
26 25 25 25 24 24 24 23 23
23 23 22 22 22 21 21 21 21
20 20 20 19 19 19 19 18 18
18 18 17 17 17 16 16 16 16
15 15 15 15 14 14 14 14 13
13 13 13 12 12 12 12 12 11
11 11 11 10 10 10 10 9 9
9 9 9 8 8 8 8 8 7
7 7 7 6 6 6 6 6 5
5 5 5 5 4 4 4 4 4
3 3 3 3 3 2 2 2 2
2 2 1 1 1 1 1 0 0
0 0 0

clut: #clut points**#input channels * #output channels = 8
0 3 253 0 0 3 253 0
output tables: #channels * 256 = 256
255 249 244 238 233 228 223 218 214
209 205 200 196 192 188 184 180 176
172 168 165 161 158 154 151 148 144
141 138 135 132 129 127 124 121 119
116 114 111 109 106 104 102 100 97
95 93 91 89 87 85 84 82 80
78 77 75 73 72 70 69 67 66

64 63 62 60 59 58 56 55 54
53 52 51 49 48 47 46 45 44
43 42 41 41 40 39 38 37 36
36 35 34 33 33 32 31 30 30
29 28 28 27 27 26 25 25 24
24 23 23 22 22 21 21 20 20
20 19 19 18 18 17 17 17 16
16 16 15 15 15 14 14 14 13
13 13 12 12 12 12 11 11 11
11 10 10 10 10 9 9 9 9
9 8 8 8 8 8 8 7 7
7 7 7 7 6 6 6 6 6
6 6 5 5 5 5 5 5 5
5 4 4 4 4 4 4 4 4
4 4 4 3 3 3 3 3 3
3 3 3 3 3 3 3 2 2
2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 0

signature = 0x63707274 signatureId = cprt, offset = 504/1284 size = 2a/42
Text type
Copyright: Sun Microsystems 1998

signature = 0x64657363 signatureId = desc, offset = 530/1328 size = 7e/126
Text description
This is the profile description tag

CHAPTER 7

Dissecting Printer Profiles

The hexadecimal file below is a listing of a printer profile.. The contents of this file are dissected below, concentrating on the differences in tags between this and previous profiles. The entire profile is much too large to include in its entirety. The portions which have been left out are indicated.

```
count     hex profile data
------    -----------------
0000000 0005b82c 4b434d53 02000000 70727472
0000016 52474220 4c616220 07cd0004 0009000e
0000032 00040004 61637370 53554e57 00000000
0000048 53554e20 6e6f6e65 00000000 00000000
0000064 00000000 0000f6d5 00010000 0000d32b
0000080 4b4f4441 00000000 00000000 00000000
0000096 00000000 00000000 00000000 00000000
*
0000128 00000029 63707274 00000270 0000003f
0000144 4b303331 000002b0 0000000e 4b303138
0000160 000002c0 0000000c 646d6e64 000002cc
```

0000176 00000068 646d6464 00000334 0000008c
0000192 4b303037 000003c0 00000077 64657363
0000208 00000438 00000098 4b303139 000004d0
0000224 00000083 4b303231 000002c0 0000000c
0000240 4b303133 000002c0 0000000c 4b303136
0000256 00000554 0000007a 4b303137 000002c0
0000272 0000000c 4b303130 000005d0 00000028
0000288 4b303330 000002c0 0000000c 4b303232
0000304 000005f8 00000028 4b303233 000005f8
0000320 00000028 4b303234 000005f8 00000028
0000336 77747074 00000620 00000014 41324230
0000352 00000634 0000c634 4b303730 0000cc68
0000368 0000000a 4b303238 0000cc74 0000000c
0000384 4b303239 0000cc80 0000000c 62666420
0000400 0000cc8c 00000019 42324130 0000cca8
0000416 0000c634 4b303731 000192dc 0000000a
0000432 42324131 000192e8 0000c634 4b303734
0000448 000192dc 0000000a 42324132 0002591c
0000464 0000c634 4b303737 000192dc 0000000a
0000480 70726530 00031f50 0000c634 4b303732
0000496 0003e584 0000000a 70726531 0003e590
0000512 0000c634 4b303735 0003e584 0000000a
0000528 70726532 0004abc4 0000c634 4b303738
0000544 0003e584 0000000a 67616d74 000571f8
0000560 00004634 41324231 00000634 0000c634
0000576 41324232 00000634 0000c634 4b303733
0000592 0000cc68 0000000a 4b303736 0000cc68
0000608 0000000a 4b303739 0003e584 0000000a

bunch of data deleted

0366384 2a9a1e49 15450fbe 0aaa0545 00000249
0366400 059608a2 0c921259 1c715165 57be3c10
0366416 2aba1f4d 16080f8e 09960649 03df0649
0366432 0aaa0d86 1104169a 1f3c5175 58823c92
0366448 2c102071 170c10b2 0cb20a18 08410aeb
0366464 1061134d 14f31a49 22595020 5a383dd7
0366480 2dd721d7 195513ef 10b20eba 0d750fff
0366496 16591904 18611e8a 270c4934 5c614071
0366512 30202504 1ca2179e 16081514 14821628
0366528 1d4520c3 1e7923ff 2f4d4279 5fcf4441
0366544 34922a08 234d1f2c 1e8a1e79 1e691e28
0366560 21c723cf 25f72b7d 36cb5d65 66aa4bef
0366576 3d143430 314530a2 30b23030 2f8e2e8a
0366592 2e492fdf 326935f7 414579e7 7e496492
0366608 574d4fff 4e494e38 4e284e59 4eaa4f5d
0366624 508251c7 548258f3 61a6976d 00000000
0366640 00000000 00000000 00000000 00000000
*
0366960 00000000 ff0cff0c ff0cff0c ff0cff0c
0366976 ff0cff0c ff0cff0c ff0cff0c ff0cff0c
*
0374828

The tag header decodes to the following. Note the tags beginning with “K0” - these are either private tags or tags which have been registered with the ICC. A list of registered tags may be downloaded from the www.color.org web site.

signature = 0x63707274 signatureId = cprt, offset = 624 size = 63
signature = 0x4b303331 signatureId = K031, offset = 688 size = 14
signature = 0x4b303138 signatureId = K018, offset = 704 size = 12
signature = 0x646d6e64 signatureId = dmnd, offset = 716 size = 104

signature = 0x646d6464 signatureId = dmdd, offset = 820 size = 140
signature = 0x4b303037 signatureId = K007, offset = 960 size = 119
signature = 0x64657363 signatureId = desc, offset = 1080 size = 152
signature = 0x4b303139 signatureId = K019, offset = 1232 size = 131
signature = 0x4b303231 signatureId = K021, offset = 704 size = 12
signature = 0x4b303133 signatureId = K013, offset = 704 size = 12
signature = 0x4b303136 signatureId = K016, offset = 1364 size = 122
signature = 0x4b303137 signatureId = K017, offset = 704 size = 12
signature = 0x4b303130 signatureId = K010, offset = 1488 size = 40
signature = 0x4b303330 signatureId = K030, offset = 704 size = 12
signature = 0x4b303232 signatureId = K022, offset = 1528 size = 40
signature = 0x4b303233 signatureId = K023, offset = 1528 size = 40
signature = 0x4b303234 signatureId = K024, offset = 1528 size = 40
signature = 0x77747074 signatureId = wtpt, offset = 1568 size = 20
signature = 0x41324230 signatureId = A2B0, offset = 1588 size = 50740
signature = 0x4b303730 signatureId = K070, offset = 52328 size = 10
signature = 0x4b303238 signatureId = K028, offset = 52340 size = 12
signature = 0x4b303239 signatureId = K029, offset = 52352 size = 12
signature = 0x62666420 signatureId = bfd , offset = 52364 size = 25
signature = 0x42324130 signatureId = B2A0, offset = 52392 size = 50740
signature = 0x4b303731 signatureId = K071, offset = 103132 size = 10
signature = 0x42324131 signatureId = B2A1, offset = 103144 size = 50740
signature = 0x4b303734 signatureId = K074, offset = 103132 size = 10
signature = 0x42324132 signatureId = B2A2, offset = 153884 size = 50740
signature = 0x4b303737 signatureId = K077, offset = 103132 size = 10
signature = 0x70726530 signatureId = pre0, offset = 204624 size = 50740
signature = 0x4b303732 signatureId = K072, offset = 255364 size = 10
signature = 0x70726531 signatureId = pre1, offset = 255376 size = 50740
signature = 0x4b303735 signatureId = K075, offset = 255364 size = 10
signature = 0x70726532 signatureId = pre2, offset = 306116 size = 50740
signature = 0x4b303738 signatureId = K078, offset = 255364 size = 10

signature = 0x67616d74 signatureId = gamt, offset = 356856 size = 17972
signature = 0x41324231 signatureId = A2B1, offset = 1588 size = 50740
signature = 0x41324232 signatureId = A2B2, offset = 1588 size = 50740
signature = 0x4b303733 signatureId = K073, offset = 52328 size = 10
signature = 0x4b303736 signatureId = K076, offset = 52328 size = 10
signature = 0x4b303739 signatureId = K079, offset = 255364 size = 10

Here is a “dump” of the profile contents.

Size in bytes = 374828
CMM Id = KCMS
Version number = 0x2000000
deviceClass = output
colorspace = RGB
profile connection space = Lab
date = 9/4/1997, time = 14:4:4
magic number = acsp
platform = Solaris
Non-embedded profile
OK to strip embedded profile out and use independently
manufacturer = SUN
model = none
device attributes = 00
rendering intent = Perceptual
Illuminat X=0.964188 Y=1.000000 X=0.824890
creator = KODA

Number of tags = 44

Tag # = 0, Tag Hex = 0x63707274, Tag Ascii = cprt
Text type

Tag # = 1, Tag Hex = 0x4b303331, Tag Ascii = K031
Text type
01.00
Tag # = 2, Tag Hex = 0x4b303138, Tag Ascii = K018
Unsigned 32-bit values
0x1

Tag # = 3, Tag Hex = 0x646d6e64, Tag Ascii = dmnd
Text description
SUN

Tag # = 4, Tag Hex = 0x646d6464, Tag Ascii = dmdd
Text description
SPARCprinter EC

Tag # = 5, Tag Hex = 0x4b303037, Tag Ascii = K007
Text description
raw cmyk

Tag # = 6, Tag Hex = 0x64657363, Tag Ascii = desc
Text description
SUN SPARCprinter EC

Tag # = 7, Tag Hex = 0x4b303139, Tag Ascii = K019
Text description
coated paper

Tag # = 8, Tag Hex = 0x4b303231, Tag Ascii = K021
Unsigned 32-bit values

0x1

Tag # = 9, Tag Hex = 0x4b303133, Tag Ascii = K013
Unsigned 32-bit values
0x1

Tag # = 10, Tag Hex = 0x4b303136, Tag Ascii = K016
Text description
linv2test

Tag # = 11, Tag Hex = 0x4b303137, Tag Ascii = K017
Unsigned 32-bit values
0x1

Tag # = 12, Tag Hex = 0x4b303130, Tag Ascii = K010
Signed 15.16 fixed point data
1.769989
1.769989
1.769989
0.000000
0.000000
0.000000
0.000000
0.000000

Tag # = 13, Tag Hex = 0x4b303330, Tag Ascii = K030
Unsigned 32-bit values
0x1

Tag # = 14, Tag Hex = 0x4b303232, Tag Ascii = K022
Signed 15.16 fixed point data

0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000

Tag # = 15, Tag Hex = 0x4b303233, Tag Ascii = K023
Signed 15.16 fixed point data
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000

Tag # = 16, Tag Hex = 0x4b303234, Tag Ascii = K024
Signed 15.16 fixed point data
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000

Tag # = 17, Tag Hex = 0x77747074, Tag Ascii = wtpt
XYZ type
X=0.819641, Y=0.852615, Z=0.701324

Tag # = 18, Tag Hex = 0x4b303730, Tag Ascii = K070
Unsigned 8-bit values
5
6

Tag # = 19, Tag Hex = 0x4b303238, Tag Ascii = K028
Unsigned 32-bit values
0x2

Tag # = 20, Tag Hex = 0x4b303239, Tag Ascii = K029
Unsigned 32-bit values
0x3

Tag # = 21, Tag Hex = 0x62666420, Tag Ascii = bfd
UCR BG description
UCR curve length = 1
UCR = 299 percent
BG curve length = 1
BG = 30 percent
UcrBg description =

Tag # = 22, Tag Hex = 0x4b303731, Tag Ascii = K071
Unsigned 8-bit values
6
4

Tag # = 23, Tag Hex = 0x4b303734, Tag Ascii = K074

Unsigned 8-bit values
6
4

Tag # = 24, Tag Hex = 0x4b303737, Tag Ascii = K077
Unsigned 8-bit values
6
4

Tag # = 25, Tag Hex = 0x4b303732, Tag Ascii = K072
Unsigned 8-bit values
6
6

Tag # = 26, Tag Hex = 0x4b303735, Tag Ascii = K075
Unsigned 8-bit values
6
6

Tag # = 27, Tag Hex = 0x4b303738, Tag Ascii = K078
Unsigned 8-bit values
6
6

Tag # = 28, Tag Hex = 0x4b303733, Tag Ascii = K073
Unsigned 8-bit values
5
6

Tag # = 29, Tag Hex = 0x4b303736, Tag Ascii = K076
Unsigned 8-bit values

5
6

Tag # = 30, Tag Hex = 0x4b303739, Tag Ascii = K079
Unsigned 8-bit values
6
6

The CLUT data is not printed, due to its size.

Tag # = 31, Tag Hex = 0x41324230, Tag Ascii = A2B0
Size 53138 of Lut 41324230
Lut 16 type

Tag # = 32, Tag Hex = 0x41324231, Tag Ascii = A2B1
Size 53138 of Lut 41324231
Lut 16 type

Tag # = 33, Tag Hex = 0x41324232, Tag Ascii = A2B2
Size 53138 of Lut 41324232
Lut 16 type

Tag # = 34, Tag Hex = 0x42324130, Tag Ascii = B2A0
Size 53137 of Lut 42324130
Lut 16 type

Tag # = 35, Tag Hex = 0x42324131, Tag Ascii = B2A1
Size 53137 of Lut 42324131
Lut 16 type

Tag # = 36, Tag Hex = 0x42324132, Tag Ascii = B2A2

Size 53138 of Lut 42324132
Lut 16 type

Tag # = 37, Tag Hex = 0x70726530, Tag Ascii = pre0
Size 53137 of Lut 70726530
Lut 16 type

Tag # = 38, Tag Hex = 0x70726531, Tag Ascii = pre1
Size 53137 of Lut 70726531
Lut 16 type

Tag # = 39, Tag Hex = 0x70726532, Tag Ascii = pre2
Size 53137 of Lut 70726532
Lut 16 type

Tag # = 40, Tag Hex = 0x67616d74, Tag Ascii = gamt
Size 20264 of Lut 67616d74
Lut 16 type

Tag # = 41, Tag Hex = 0x69636864, Tag Ascii = ichd
Header type

CHAPTER 8

Dissecting Other Profile Types

This chapter presents samples of other profile types. The same sort of hexidecimal listing of a sample profile is provided along with the output of the tag "dumping" program. The tag signatures are underlined in the tag directory and in the tag data to help you in dissecting the profiles.

Each sample profile contains its required tags, but some optional tags are included as examples.

The Device Link Profile Type

This profile contains examples of the 4 required tags for a device link profile. Due to the size of the AtoB tag, some of the lookup table data is deleted.

count hex profile data

0000000 00001014 4b434d53 21000000 6c696e6b

0000010 52474220 4c616220 07ce0006 0008000d

0000020 000c002a 61637370 53554e57 00000000

0000030 53554e20 31393938 00000001 00000000

0000040 00010000 0000f6d5 00010000 0000d32b

0000050 53554e57 00000000 00000000 00000000
0000060 00000000 00000000 00000000 00000000
*
0000080 00000004 41324230 000000b4 00000c64
0000090 64657363 00000d18 0000007e 63707274
00000a0 00000d98 0000002a 70736571 00000dc4
00000b0 00000250 6d667432 00000000 03030800
00000c0 00010000 00000000 00000000 00000000
*
00000e0 00010000 00020002 0000ffff 0000ffff
00000f0 0000ffff 00000000 00000000 00001fe0
0000100 00000000 3fc00000 00005fa0 00000000
0000110 7f800000 00009f60 00000000 bf400000
0000120 0000df20 00001fe0 00000000 1fe01fe0
0000130 00001fe0 3fc00000 1fe05fa0 00001fe0
0000140 7f800000 1fe09f60 00001fe0 bf400000
0000150 1fe0df20 00003fc0 00000000 3fc01fe0
0000160 00003fc0 3fc00000 3fc05fa0 00003fc0
0000170 7f800000 3fc09f60 00003fc0 bf400000
0000180 3fc0df20 00005fa0 00000000 5fa01fe0
0000190 00005fa0 3fc00000 5fa05fa0 00005fa0
00001a0 7f800000 5fa09f60 00005fa0 bf400000
00001b0 5fa0df20 00007f80 00000000 7f801fe0
00001c0 00007f80 3fc00000 7f805fa0 00007f80
data deleted
0000cb0 7f80df20 bf409f60 df20bf40 bf40df20
0000cc0 bf40df20 df20df20 0000df20 df201fe0
0000cd0 df20df20 3fc0df20 df205fa0 df20df20
0000ce0 7f80df20 df209f60 df20df20 bf40df20
0000cf0 df20df20 0000ffff 0000ffff 0000ffff
0000d00 00000000 00000000 00000000 00000000

0000d10 00000000 00000000 64657363 00000000
0000d20 00000024 54686973 20697320 74686520
0000d30 70726f66 696c6520 64657363 72697074
0000d40 696f6e20 74616700 00000000 00000000
0000d50 00000000 00000000 00000000 00000003
0000d60 36900000 00000000 0d200000 009c0000
0000d70 007e0000 00000001 0000ef7e bf58ef7e
0000d80 d0a40000 1fff0000 00000000 060c0000
0000d90 00000000 00000000 74657874 00000000
0000da0 436f7079 72696768 743a2020 53756e20
0000db0 4d696372 6f737973 74656d73 20313939
0000dc0 38000000 70736571 00000000 00000002
0000dd0 53554e57 31393938 00000001 00000000
0000de0 43525420 64657363 00000000 00000030
0000df0 54686973 20697320 74686520 64657669
0000e00 6365206d 616e7566 61637475 72657220
0000e10 64657363 72697074 696f6e20 74616700
0000e20 00000000 00000000 00000000 00000000
0000e30 00000000 00000003 36900000 00000000
0000e40 0de40000 00b40000 02500000 00000001
0000e50 0000ef7e bf58ef7e d0a40000 1fff0000
0000e60 00000000 060c0000 00000000 00006465
0000e70 73630000 00000000 002a5468 69732069
0000e80 73207468 65206465 76696365 206d6f64
0000e90 656c2064 65736372 69707469 6f6e2074
0000ea0 61672000 00000000 00000000 00000000
0000eb0 00000000 00000000 00000003 36900000
0000ec0 00000000 0e6e0000 00b40000 02500000
0000ed0 00000001 0000ef7e bf58ef7e d0a40000
0000ee0 1fff0000 00000000 060c0000 00000000
0000ef0 00005355 4e573139 39370000 00010000

0000f00 0000696a 65746465 73630000 00000000
0000f10 00305468 69732069 73207468 65206465
0000f20 76696365 206d616e 75666163 74757265
0000f30 72206465 73637269 7074696f 6e207461
0000f40 67000000 00000000 00000000 00000000
0000f50 00000000 00000000 00033690 00000000
0000f60 00000f06 000000b4 00000250 00000000
0000f70 00010000 ef7ebf58 ef7ed0a4 00001fff
0000f80 00000000 0000060c 00000000 00000000
0000f90 64657363 00000000 0000002a 54686973
0000fa0 20697320 74686520 64657669 6365206d
0000fb0 6f64656c 20646573 63726970 74696f6e
0000fc0 20746167 20000000 00000000 00000000
0000fd0 00000000 00000000 00000000 00033690
0000fe0 00000000 00000f90 000000b4 00000250
0000ff0 00000000 00010000 ef7ebf58 ef7ed0a4
0001000 00001fff 00000000 0000060c 00000000
0001010 00000000
0001014

Size in bytes = 4116
CMM Id = KCMS
Version number = 0x21000000
deviceClass = link
colorspace = RGB
profile connection space = Lab
date = 8/6/1998, time = 13:12:42
magic number = acsp
platform = Solaris
Non-embedded profile
OK to strip embedded profile out and use independently

manufacturer = SUN

model = 1998

Attributes = transparency, glossy, positive, color

rendering intent = Relative Colorimetric

Illuminat X=0.964188 Y=1.000000 Z=0.824875

creator = SUNW

Number of tags in this profile = 4

This tag is an example containing fictitious data. The ICC specification does not specify how to create this tag to represent the sequence of profiles.

signature = 0x41324230 signatureId = A2B0, offset = b4/180 size = c64/3172

16 bit lut type

Lut 16 type

3 3 8

1.000000 0.000000 0.000000

0.000000 1.000000 0.000000

0.000000 0.000000 1.000000

2 2

input tables: #channels * #input entries = 6

0 65535 0 65535 0 65535

clut: #clut points**#input channels * #output channels *2 = 1536

0 0 0 0 0 8160 0 0 16320

0 0 24480 0 0 32640 0 0 40800

0 0 48960 0 0 57120 0 8160 0

0 8160 8160 0 8160 16320 0 8160 24480

0 8160 32640 0 8160 40800 0 8160 48960

0 8160 57120 0 16320 0 0 16320 8160

0 16320 16320 0 16320 24480 0 16320 32640

0 16320 40800 0 16320 48960 0 16320 57120
0 24480 0 0 24480 8160 0 24480 16320
Clut data deleted ...
57120 57120 24480 57120 57120 32640 57120 57120 40800
57120 57120 48960 57120 57120 57120
output tables: #channels * #output entries = 6
0 65535 0 65535 0 65535

signature = 0x64657363 signatureId = desc, offset = d18/3352 size = 7e/126
Text description
This is the profile description tag

signature = 0x63707274 signatureId = cprt, offset = d98/3480 size = 2a/42
Text type
Copyright: Sun Microsystems 1998

This is the profile sequence tag containing information from each profile in the sequence.
signature = 0x70736571 signatureId = pseq, offset = dc4/3524 size = 250/592
Number of profile descriptions in tag = 2
device maufacturer = SUNW
device model = 1998
device attributes= 1, 0
technology = CRT
Device manufacturer string length and contents = 48, This is the device manufacturer description tag
Device model string length and contents = 42, This is the device model description tag

device maufacturer = SUNW
device model = 1997
device attributes= 1, 0
technology = ijet
Device manufacturer string length and contents = 48, This is the device manufacturer description tag
Device model string length and contents = 42, This is the device model description tag

The Named Color Profile Type

This profile includes samples of the 4 required tags plus 3 optional tags: Ps2CRD0 (psd0), Ps2RenderingIntent (ps2i), and CrdInfoTag (crdi).

count hex profile data
0000000 0000032c 4b434d53 21000000 6d6e7472
0000010 52474220 4c616220 07cf0008 0010000b
0000020 00180002 61637370 53554e57 00000000
0000030 53554e20 31393938 00000009 00000000
0000040 00010000 0000f6d5 00010000 0000d32b
0000050 53554e57 00000000 00000000 00000000
0000060 00000000 00000000 00000000 00000000
*
0000080 00000007 64657363 000000d8 0000007e
0000090 77747074 00000158 00000014 63707274
00000a0 0000016c 0000002a 70736430 00000198
00000b0 0000001a 70733269 000001b4 00000022
00000c0 6e636c32 000001d8 000000ac 63726469
00000d0 00000284 000000a8 64657363 00000000

00000e0 00000024 54686973 20697320 74686520
00000f0 70726f66 696c6520 64657363 72697074
0000100 696f6e20 74616700 00000000 00000000
0000110 00000000 00000000 00000000 00000000
0000120 0000ef6a 5f480000 00000000 00000000
0000130 00e00000 00900000 007eef7e d0a40000
0000140 1fff0000 00000000 00000000 00000000
0000150 00000000 00040000 58595a20 00000000
0000160 0000f6d5 00010000 0000d32b 74657874
0000170 00000000 436f7079 72696768 743a2020
0000180 53756e20 4d696372 6f737973 74656d73
0000190 20313939 38000000 64617461 00000000
00001a0 00000000 50533220 43524420 30207461
00001b0 67000000 64617461 00000000 00000000
00001c0 52656c61 74697665 436f6c6f 72696d65
00001d0 74726963 20000000 64617461 00000000
00001e0 00040000 00000002 00000003 6c696768
00001f0 74000000 00000000 00000000 00000000
0000200 00000000 00000000 00000000 69736800
0000210 00000000 00000000 00000000 00000000
0000220 00000000 00000000 00000000 72656400
0000230 00000000 00000000 00000000 00000000
0000240 00000000 00000000 00000000 00ff0000
0000250 00000080 00030004 626c7565 00000000
0000260 00000000 00000000 00000000 00000000
0000270 00000000 00000000 00000000 00ff000a
0000280 000900c8 63726469 00000000 00000018
0000290 506f7374 53637269 70742070 726f6475
00002a0 6374206e 616d6500 0000001c 52656e64
00002b0 6572696e 6720696e 74656e74 20302043
00002c0 5244206e 616d6500 0000001c 52656e64

00002d0 6572696e 6720696e 74656e74 20312043
00002e0 5244206e 616d6500 0000001c 52656e64
00002f0 6572696e 6720696e 74656e74 20322043
0000300 5244206e 616d6500 0000001c 52656e64
0000310 6572696e 6720696e 74656e74 20332043
0000320 5244206e 616d6500 00000000
000032c

Decoded Header:
Size in bytes = 812
CMM Id = KCMS
Version number = 0x21000000
deviceClass = display
colorspace = RGB
profile connection space = Lab
date = 16/8/1999, time = 11:30:34
magic number = acsp
platform = Solaris
Non-embedded profile
OK to strip embedded profile out and use independently
manufacturer = SUN
model = 1998
Attributes = transparency, glossy, positive, black&white
rendering intent = Relative Colorimetric
Illuminat X=0.964188 Y=1.000000 Z=0.824875
creator = SUNW

Number of tags in this profile = 7
signature = 0x64657363 signatureId = desc, offset = d8/216 size = 7e/126
Text description
This is the profile description tag

signature = 0x77747074 signatureId = wtpt, offset = 158/344 size =14/20
XYZ type
X=0.964188, Y=1.000000, Z=0.824875

signature = 0x63707274 signatureId = cprt, offset = 16c/364 size = 2a/42
Text type
Copyright: Sun Microsystems 1998

signature = 0x70736430 signatureId = psd0, offset = 198/408 size = 1a/26
Data type
Ascii data
PS2 CRD 0 tag

signature = 0x70733269 signatureId = ps2i, offset = 1b4/436 size = 22/34
Data type
Ascii data
RelativeColorimetric

signature = 0x6e636c32 signatureId = ncl2, offset = 1d8/472 size = ac/172
Named color type
Vendor = 262144
Count = 2
Number device coordinates = 3
Color prefix = light
Color suffix = ish

Color 1 Root name = red
PCS Coordinates= 255 0 0

Device Coordinates= 128 3 4

Color 2 Root name = blue
PCS Coordinates= 0 0 255
Device Coordinates= 10 9 200

signature = 0x63726469 signatureId = crdi, offset = 284/644 size = a8/168
CRD Info type
PostScript Product name count and string = 24, PostScript product name
Rendering Intent 0 CRD count and name = 28, Rendering intent 0 CRD name
Rendering Intent 1 CRD count and name = 28, Rendering intent 1 CRD name
Rendering Intent 2 CRD count and name = 28, Rendering intent 2 CRD name
Rendering Intent 3 CRD count and name = 28, Rendering intent 3 CRD name

Colorspace Profile

The colorspace profile below has much of the data deleted due to its size. It is a profile containing real data for colorspace conversions between RGB709 and CIELAB. Because it is a real profile, there are additional tags to be seen which are "private" tags. These tags have special meaning for the profile/CMM creator, but should not be considered required for the proper handling of the profile by another CMM.

count hex profile data
0000000 0001901a 4b434d53 02000000 73706163
0000010 52474220 4c616220 07cc0003 001c0009
0000020 000f0004 61637370 53554e20 00000000
0000030 4b4f4441 6e6f6e65 00000000 00000000
0000040 00000000 0000f6b7 00010000 0000d2f5
0000050 4b4f4441 00000000 00000000 00000000
0000060 00000000 00000000 00000000 00000000
*

0000080 0000000d 63707274 00000120 00000047
0000090 646d6e64 00000168 0000006e 646d6464
00000a0 000001d8 00000077 4b303133 00000250
00000b0 0000000c 4b303331 0000025c 0000000e
00000c0 64657363 0000026c 00000089 4b303139
00000d0 000002f8 00000083 4b303330 0000037c
00000e0 0000000c 77747074 00000388 00000014
00000f0 41324230 0000039c 0000c634 4b303730
0000100 0000c9d0 0000000a 42324130 0000c9dc
0000110 0000c634 4b303731 00019010 0000000a
0000120 74657874 00000000 436f7079 72696768
0000130 74202863 29203139 39362045 6173746d
0000140 616e204b 6f64616b 20436f6d 70616e79
0000150 2c20416c 6c205269 67687473 20526573
0000160 65727665 642e0000 64657363 00000000
0000170 00000006 4b4f4441 4b000000 00000000
0000180 0007feff 004b004f 00440041 004b0000
0000190 0000064b 4f44414b 00005900 54a94000
00001a0 00005900 48f46900 40ad4000 40ad4000
00001b0 40ad4000 40ad4000 54a94000 00005900
00001c0 54a94000 00005900 40ad4000 54a94000
00001d0 00005900 00000000 64657363 00000000
00001e0 00000009 43434952 20373039 00000000
00001f0 00000000 0afeff00 43004300 49005200
0000200 20003700 30003900 00000009 43434952
0000210 20373039 0054a940 00000059 0048f469
0000220 0040ad40 0040ad40 0040ad40 0040ad40
0000230 0054a940 00000059 0054a940 00000059
0000240 0040ad40 0054a940 00000059 00000000
0000250 75693332 00000000 00000001 74657874
0000260 00000000 30312e30 30000000 64657363

0000270 00000000 0000000f 4b4f4441 4b204343
0000280 49522037 30390000 00000000 000010fe
0000290 ff004b00 4f004400 41004b00 20004300
00002a0 43004900 52002000 37003000 39000000
00002b0 000f4b4f 44414b20 43434952 20373039
00002c0 00000003 000000a0 fadf003c fa690000
00002d0 0059007c f7690084 b1001020 fadf00a0
00002e0 fadf003c fa690000 00590098 f7690003
00002f0 b7001020 fa000000 64657363 00000000
0000300 0000000d 43434952 20373039 20524742
0000310 00000000 00000000 0efeff00 43004300
0000320 49005200 20003700 30003900 20005200
0000330 47004200 0000000d 43434952 20373039
0000340 20524742 00000059 0048f469 0040ad40
0000350 0040ad40 0040ad40 0040ad40 0054a940
0000360 00000059 0054a940 00000059 0040ad40
0000370 0054a940 00000059 00000000 75693332
0000380 00000000 00000002 58595a20 00000000
0000390 0000dc03 0000e42c 0000bc5a 6d667432
data deleted
000c9d0 75693038 00000000 01060000 6d667432
000c9e0 00000000 03031000 00010000 00000000
000c9f0 00000000 00000000 00010000 00000000
000ca00 00000000 00000000 00010000 01001000
000ca10 00000031 007800cb 0128018b 01f50264
data deleted
0019010 75693038 00000000 0601ff0c
001901a

Size in bytes = 1901a/102426

CMM Id = KCMS
Version number = 0x2000000
deviceClass = colorspace
colorspace = RGB
profile connection space = Lab
date = 28/3/1996, time = 9:15:4
magic number = acsp
Unknown
Non-embedded profile
OK to strip embedded profile out and use independently
manufacturer = KODA
model = none
Attributes = reflective, glossy, positive, color
rendering intent = Perceptual
Illuminat X=0.963730 Y=1.000000 Z=0.824051
creator = KODA

Number of tags in this profile = 13
signature = 0x63707274 signatureId = cprt, offset = 120/288 size = 47/71
Text type
Copyright (c) 1996 Eastman Kodak Company, All Rights Reserved.

signature = 0x646d6e64 signatureId = dmnd, offset = 168/360 size = 6e/10
Text description
KODAK

signature = 0x646d6464 signatureId = dmdd, offset = 1d8/472 size = 77/119
Text description
CCIR 709

private tag: signature = 0x4b303133 signatureId = K013, offset = 250/592

size = c/12

private tag: signature = 0x4b303331 signatureId = K031, offset = 25c/604
size = e/14

signature = 0x64657363 signatureId = desc, offset = 26c/620 size = 89/137
Text description
KODAK CCIR 709

private tag: signature = 0x4b303139 signatureId = K019, offset = 2f8/760
size = 83/131

private tag: signature = 0x4b303330 signatureId = K030, offset = 37c/892
size = c/12

signature = 0x77747074 signatureId = wtpt, offset = 388/904 size = 14/20
XYZ type
X=0.859421, Y=0.891296, Z=0.735748

signature = 0x41324230 signatureId = A2B0, offset = 39c/924 size = c634/50740
16 bit lut type
Lut 16 type
3 3 16
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
256   4096
input tables: #channels * #input entries = 768
0 516 1032 1548 2064 2579 3095 3611 4127
4643 5159 5663 6139 6588 7015 7422 7812 8185
8544 8891 9225 9528 9848 10166 10482 10796 11109

11421 11731 12039 12346 12651 12955 13258 13559 13859
14158 14455 14751 15046 15340 15633 15924 16215 16504
16792 17079 17365 17650 17934 18217 18500 18781 19061
19340 19618 19896 20172 20448 20723 20997 21270 21542
21814 22085 22355 22624 22892 23160 23427 23693 23958
24223 24487 24751 25013 25275 25536 25797 26057 26316
26575 26833 27091 27347 27604 27859 28114 28369 28623
28876 29129 29381 29632 29883 30134 30384 30633 30882
data deleted
52038 52253 52468 52682 52897 53111 53324 53538 53751
53964 54177 54390 54602 54814 55026 55237 55449 55660
55870 56081 56291 56502 56712 56921 57131 57340 57549
57758 57966 58175 58383 58591 58798 59006 59213 59420
59627 59834 60040 60246 60452 60658 60863 61069 61274
61479 61684 61888 62092 62297 62500 62704 62908 63111
63314 63517 63720 63922 64125 64327 64529 64731 64932
65134 65335 65535

clut: #clut points**#input channels * #output channels *2 = 12288
0 32768 32768 325 33304 30427 698 33922 27924
1267 34881 25518 2113 36295 23275 3267 38066 21243
4778 39367 19439 6566 40179 17846 8370 40960 16253
10191 41772 14661 11995 42552 13084 13799 43333 11475
15603 44113 9898 17408 44925 8305 19228 45706 6712
21032 46486 5120 3104 30557 34555 3429 31077 32199
3803 31695 29793 4372 32670 27452 5217 34019 25258
6290 35352 23194 7525 36636 21243 8874 37839 19358
data deleted
58953 36766 31695 59310 37270 29305 60220 31695 56059
60237 31727 55588 60253 31744 55068 60285 31792 54272
60318 31874 53247 60383 31971 51931 60432 32101 50371

60529 32264 48631 60643 32459 46730 60789 32686 44682
60952 32963 42536 61147 33288 40326 61374 33645 38034
61618 34052 35726 61911 34507 33385 62236 35011 31012
63439 29517 56970 63455 29533 56547 63471 29565 56076
63487 29614 55344 63520 29679 54385 63569 29761 53166
63634 29891 51703 63731 30053 50045 63829 30248 48209
63959 30476 46226 64121 30736 44145 64300 31061 41983
64495 31418 39740 64723 31825 37449 64983 32280 35124
65292 32768 32768
output tables: #channels * #output entries = 12288
0 16 32 48 65 81 97 113 130
146 162 178 195 211 227 243 260 276
292 308 325 341 357 373 390 406 422
438 455 471 487 503 520 536 552 568
585 601 617 633 650 666 682 698 715
731 747 763 780 796 812 828 845 861
877 893 910 926 942 958 975 991 1007
1023 1024 1040 1056 1072 1089 1105 1121 1137
data deleted
64349 64365 64381 64398 64414 64430 64446 64463 64479
64495 64511 64512 64528 64544 64560 64577 64593 64609
64625 64642 64658 64674 64690 64707 64723 64739 64755
64772 64788 64804 64820 64837 64853 64869 64885 64902
64918 64934 64950 64967 64983 64999 65015 65032 65048
65064 65080 65097 65113 65129 65145 65162 65178 65194
65210 65227 65243 65259 65275 65292 65292 65292 65292
65292 65292 65292 65292 65292 65292 65292 65292 65292
65292 65292 65292

private tag: signature = 0x4b303730 signatureId = K070, offset = c9d0/51664
size = a/10

signature = 0x42324130 signatureId = B2A0, offset = c9dc/51676
size = c634/50740
16 bit lut type
Lut 16 type
3 3 16
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
256 4096
input tables: #channels * #input entries = 768
0 49 120 203 296 395 501 612 728
848 973 1101 1233 1368 1506 1648 1792 1939
2088 2241 2395 2552 2711 2872 3036 3201 3369
3538 3709 3882 4057 4234 4412 4592 4774 4958
5142 5329 5517 5706 5897 6090 6283 6479 6675
6873 7072 7273 7475 7678 7882 8088 8294 8502
8711 8922 9133 9346 9559 9774 9990 10207 10425
data deleted
54596 54879 55162 55445 55729 56012 56296 56581 56865
57150 57435 57721 58007 58293 58579 58866 59152 59440
59727 60015 60303 60591 60880 61169 61458 61747 62037
62327 62617 62908 63199 63490 63781 64073 64365 64657
64950 65243 65535

clut: #clut points**#input channels * #output channels *2 = 12288
15831 22365 33564 17066 22397 29647 17976 22414 26754
18610 22414 24706 19049 22430 23340 19325 22430 22495
19456 22446 22007 19553 22446 21731 19634 22446 21471
19732 22446 21195 19813 22446 20935 19894 22446 20658
19976 22446 20431 20057 22446 20171 20138 22446 19927

data deleted
65535 26721 30069 65535 26737 27014 65535 26754 24641
65535 26770 22853 65535 26770 21520 65535 26770 20610
65535 23324 65535 65535 23438 65535 65535 23535 65535
65535 23600 65535 65535 23682 65535 65535 23730 60123
65535 23795 51703 65535 23828 44665 65535 23860 38846
65535 23893 34084 65535 23925 30232 65535 23942 27176
65535 23958 24803 65535 23974 23015 65535 23974 21699
65535 23974 20772
output tables: #channels * #output entries = 12288
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
data deleted
65292 65292 65292 65292 65292 65292 65292 65292 65292
65292 65292 65292 65292 65292 65292 65292 65292 65292
65292 65292 65292 65292 65292 65292 65292 65292 65292
65292 65292 65292 65292 65292 65292 65292 65292 65292
65292 65292 65292

private tag: signature = 0x4b303731 signatureId = K071, offset = c9d0/102416
size = a/10

Abstract Profile

This profile may be used to perform some overall image affect and might normally be used in a link between other device profiles. This particular sample merely gives the image an overall blue cast.

count hex profile data

0000000 0000027a 4b434d53 21000000 4c616220
0000010 4c616220 58595a20 07ce0006 000e0009
0000020 00030001 61637370 53554e57 00000000
0000030 53554e20 31393938 00000002 00000000
0000040 00010000 0000f6d5 00010000 0000d32b
0000050 53554e57 00000000 00000000 00000000
0000060 00000000 00000000 00000000 00000000
*
0000080 00000004 41324230 000000b4 00000106
0000090 77747074 000001bc 00000014 63707274
00000a0 000001d0 0000002a 64657363 000001fc
00000b0 0000007e 6d667432 00000000 03030300
00000c0 00010000 00000000 00000000 00000000
*
00000e0 00010000 00020002 0000ffff 0000ffff
00000f0 0000ffff 00000000 09f60000 00008877
0000100 00000000 ffff0000 798609f6 00007986
0000110 88770000 7986ffff 0000f807 09f60000
0000120 f8078877 0000f807 ffff7986 000009f6
0000130 79860000 88777986 0000ffff 79867986
0000140 09f67986 79868877 79867986 ffff7986
0000150 f80709f6 7986f807 88777986 f807ffff
0000160 f8070000 09f6f807 00008877 f8070000
0000170 fffff807 798609f6 f8077986 8877f807
0000180 7986ffff f807f807 09f6f807 f8078877
0000190 f807f807 ffff0000 ffff0000 ffff0000
00001a0 ffff0000 00000000 00000000 00000000
00001b0 00000000 00000000 00000000 58595a20
00001c0 00000000 0000f6d5 00010000 0000d32b
00001d0 74657874 00000000 436f7079 72696768
00001e0 743a2020 53756e20 4d696372 6f737973

00001f0 74656d73 20313939 38000000 64657363
0000200 00000000 00000024 54686973 20697320
0000210 74686520 70726f66 696c6520 64657363
0000220 72697074 696f6e20 74616700 00000000
0000230 00000000 00000000 00000000 00000000
0000240 00000000 00000002 96500000 00000000
0000250 02040000 00b40000 007eef7e d0a40000
0000260 1fff0000 00000000 00000000 00000000
0000270 00000000 005def7e d1ec0000 00000000
0000280 00000000 00000000
0000286

Size in bytes = 27a/634
CMM Id = KCMS
Version number = 0x21000000
Unknown
colorspace = Lab
profile connection space = XYZ
date = 14/6/1998, time = 9:3:1
magic number = acsp
platform = Solaris
Non-embedded profile
OK to strip embedded profile out and use independently
manufacturer = SUN
model = 1998
Attributes = reflective, matte, positive, color
rendering intent = Relative Colorimetric
Illuminat X=0.964188 Y=1.000000 Z=0.824875
creator = SUNW

Number of tags in this profile = 4
signature = 0x41324230 signatureId = A2B0, offset = b4/180 size = 106/262
16 bit lut type
Lut 16 type
3 3 3
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
2 2
input tables: #channels * #input entries = 6
0 65535 0 65535 0 65535

clut: #clut points**#input channels * #output channels *2 = 81
0 0 2550 0 0 34935 0 0 65535
0 31110 2550 0 31110 34935 0 31110 65535
0 63495 2550 0 63495 34935 0 63495 65535
31110 0 2550 31110 0 34935 31110 0 65535
31110 31110 2550 31110 31110 34935 31110 31110 65535
31110 63495 2550 31110 63495 34935 31110 63495 65535
63495 0 2550 63495 0 34935 63495 0 65535
63495 31110 2550 63495 31110 34935 63495 31110 65535
63495 63495 2550 63495 63495 34935 63495 63495 65535

output tables: #channels * #output entries = 6
0 65535 0 65535 0 65535

signature = 0x77747074 signatureId = wtpt, offset = 1bc/444 size = 14/20
XYZ type
X=0.964188, Y=1.000000, Z=0.824875

signature = 0x63707274 signatureId = cprt, offset = 1d0/464 size = 2a/42
Text type
Copyright: Sun Microsystems 1998

signature = 0x64657363 signatureId = desc, offset = 1fc/508 size = 7e/126
Text description
This is the profile description tag

ICC Format Number Systems A

A.1 Binary Number System

Each digit in a base 2 number is multiplied by a power of 2. To convert a binary number to a decimal number, multiply each digit (d#) by the appropriate power and add the values:

Decimal value = (d3 * 8) + (d2 * 4) + (d1 * 2) + (d0 * 1)

Decimal 1 = binary 1

Decimal 13 = binary 1101

A.2 Hexadecimal Number System

Each digit in a base 16 (hexadecimal, frequently shortened to hex) number is multiplied by a power of 16. To convert a hexadecimal number to a decimal number, multiply each digit (d#) by the appropriate power and add the values:

Decimal value = (d3 * 4096) + (d2 * 256) + (d1 * 16) + (d0 * 1)

The letters A through F represent 10 though 15, respectively.

Decimal 10 = hex A

Decimal 11 = hex B

Decimal 12 = hex C

Decimal 13 = hex D

Decimal 14 = hex E

Decimal 15 = hex F

A hex number is frequently suffixed with an ‘h’ to distinguish it from decimal numbers.

Decimal 1 = 1h

Decimal 13 = Dh

Decimal 31 = 1Fh

A.3 2's Complement Number System

2's complement number system is the way nearly all computers work. A negative value is formed by taking the 1's complement of a positive value and adding 1. The 1's complement exchanges 1 for 0 and 0 for 1 in a binary number.

decimal 5 = binary 0101

1's complement of 0101 = 1010

2's complement = 1011 (added 1 to 1's complement)

therefore

decimal -5 = binary 1011

The positive range for a 4 digit binary number is 0 to 7. The negative range is -8 to -1, one extra value.

There is also a system called the offset 2's complement number system. It is a 2's complement number with a constant offset formed by adding the size of the negative range to the 2's complement number. This is useful for applying signed values to an unsigned operation.

With a negative range of 8 (binary 1000) for a 4 digit binary number, the offset 2's complement appears as follows:

decimal 5 = 010 + 1000 = offset 2's complement 1101

decimal -5 = 1011 + 1000 = 0011

A.4 Fixed Point Number System

Fixed point numbers are integers with an implied and constant multiplier. As an example, assume we have a 4 bit number whose left 2 bits represent the integer portion of the number and the right two bits the fractional portion of the number.

decimal 1 = fixed point 0100

fixed point 0100 = 4 * (2**-2)

The decimal range for this 4 digit fixed point number is 0 to 3.75, in .25 increments.

The ICC specification uses several fixed point number systems. These are listed here along with an explanation of their names.

s15Fixed16Number is a 32 bit 2's complement number with a multiplier of 2**16. The notation is:

s = one sign bit

15 = 15 integer bits

Fixed = this is a fixed point number

16 = 16 fractional bits

Example encodings in hex:

-32768.0 = 80000000h

0 = 00000000h

1.0 = 00010000h

32767 + (65535/65536) = 7FFFFFFFh

u16Fixed16Number is identical except the u = no sign bit, thus can not represent a negative number, and has one additional integer bit to represent larger positive values. Encoding examples are:

0 = 00000000h

1 = 00010000h

65535 + (65535/65536) = FFFFFFFFh

u8Fixed8Number is an unsigned 16 bit number with 8 integer bits and 8 fractional bits. Encoding examples are:

0 = 0000h

1 = 0100h

255 + (255/256) = FFFFh

Other numbers used in the ICC specification are terms for more familiar quantities.

uInt16Number is a generic unsigned 2 byte (16 bit) integer.

uInt32Number is a generic unsigned 4 byte (32 bit) integer.

uInt64Number is a generic unsigned 8 byte (64 bit) integer.

uInt8Number is a generic unsigned 1 byte (8 bit) integer.

A.5 Code to convert into and out of fixed point

```
/*
 * Convert IC fixed point to a double
 */
double
icfixed2double(long val, long type)
{
    double              retval;
    val = swap((long)val);
    switch(type) {
    case icSigS15Fixed16ArrayType :
        retval = (double) (val / 65536.0);
        break;
    case icSigU16Fixed16ArrayType :
        retval = (double)(val/65536.0);
        break;
    default :
        retval  = -1.0;
        break;
    }

    return(retval);
}
/*
 * Convert double to an IC fixed point number
 * is not accurate enough for boundary conditions
 */
long
double2icfixed(double val, long type)
```

```
{
    long                retval;
    short               a;
    ushort      b, c;

    switch(type) {
    case icSigS15Fixed16ArrayType :
        a = (short)(val);
        b = (ushort)((val - a) * 65536.0);
        retval = (long) ((a <<16) | b);
        break;
    case icSigU16Fixed16ArrayType :
        c = (ushort)(val);
        b = (ushort)((val - c) * 65536.0);
        retval = (unsigned long)((c <<16) | b);
        break;
    default :
        retval  = -1.0;
        break;
    }
    retval = swap((long)retval);
    return(retval);
}
```

A.6 Additional Number Types Using Fixed Types

Non-intuitive number types in the specification include:

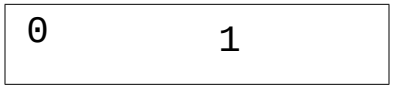
XYZNumber is a set of 3 s15Fixed16Number's used to encode CIEXYZ tristimulus values.

dateTimeNumber is a set of 6 uInt16Number's representing a 4-digit year, month, day, hours, minutes and seconds, respectively.

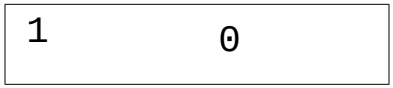
A.7 Big-endian/Little-endian

The ICC specification describes the profile format as a big-endian file. This means that the address of the bytes within a 16, 32, or 64 bit value is from the most significant to the least significant byte, as the byte addresses increase.

Big-endian byte order for 16 bit word

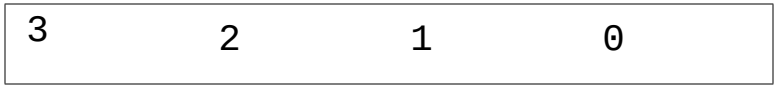


Little-endian byte order for 16 bit word



Big-endian byte order for 32 bit word

Little-endian byte order for 32 bit word



Big-endian byte order for 64 bit word

Little-endian byte order for 64 bit word

A.8 Code to swap bytes between different “endian” platforms.

```
/*
 * Routine to byte swap a long, just returns on big-endian
 */
long
swap(long value)
{
#ifdef _LITTLE_ENDIAN
    char        *ptr, c;

    ptr = (char *)&value;
    c = *(ptr+1);
    *(ptr+1) = *(ptr+2);
    *(ptr+2) = c;

    c = *ptr;
    *ptr = *(ptr+3);
    *(ptr+3) = c;
#endif _LITTLE_ENDIAN

    return(value);
}

/*
 * Routine to byte swap a short, just returns on big-endian
 */
ushort
swap16(ushort value)
{
#ifdef _LITTLE_ENDIAN
    char        *ptr, c;

    ptr = (char *)&value;
    c = *(ptr+1);
    *(ptr+1) = *(ptr);
    *ptr = c;

#endif _LITTLE_ENDIAN

    return(value);
}
```

ICC Header File in C B

```
/* Header file guard bands */
#ifndef ICC_H
#define ICC_H

/*****************************************************************
 Copyright (c) 1994-2000 Sun Microsystems, Inc.

                    Rights Reserved

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the “Software”), to deal in the Software without restrict-
ion, including without limitation the rights to use, copy, modify,
merge, publish distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished
to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND,
```

```
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-
INFRINGEMENT.  IN NO EVENT SHALL SUNSOFT, INC. OR ITS PARENT
COMPANY BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of Sun Microsysems, Inc.
shall not be used in advertising or otherwise to promote the
sale, use or other dealings in this Software without written
authorization from Sun Microsysems Inc.
************************************************************
/

/*
 * This version of the header file corresponds to
 * Specification ICC.1:2000-1
 *
 * All header file entries are pre-fixed with “ic” to help
 * avoid name space collisions. Signatures are pre-fixed with
 * icSig.
 *
 * The structures defined in this header file were created to
 * represent a description of an ICC profile on disk. Rather
 * than use pointers a technique is used where a single byte array
 * was placed at the end of each structure. This allows us in “C”
 * to extend the structure by allocating more data than is needed
 * to account for variable length structures.
 *
 * This also ensures that data following is allocated
 * contiguously and makes it easier to write and read data from
 * the file.
 *
```

```
* For example to allocate space for a 256 count length UCR
* and BG array, and fill the allocated data.  Note strlen + 1
* to remember NULL terminator.
*
       icUcrBgCurve    *ucrCurve, *bgCurve;
       int             ucr_nbytes, bg_nbytes, string_bytes;
       icUcrBg         *ucrBgWrite;
       char            ucr_string[100], *ucr_char;

       strcpy(ucr_string, "Example ucrBG curves");
       ucr_nbytes = sizeof(icUInt32Number) +
                (UCR_CURVE_SIZE * sizeof(icUInt16Number));
       bg_nbytes = sizeof(icUInt32Number) +
                (BG_CURVE_SIZE * sizeof(icUInt16Number));
       string_bytes = strlen(ucr_string) + 1;

       ucrBgWrite = (icUcrBg *)malloc(
                        (ucr_nbytes + bg_nbytes + string_bytes));

       ucrCurve = (icUcrBgCurve *)ucrBgWrite->data;
       ucrCurve->count = UCR_CURVE_SIZE;
       for (i=0; i<ucrCurve->count; i++)
               ucrCurve->curve[i] = (icUInt16Number)i;

       bgCurve = (icUcrBgCurve *)((char *)ucrCurve + ucr_nbytes);
       bgCurve->count = BG_CURVE_SIZE;
       for (i=0; i<bgCurve->count; i++)
               bgCurve->curve[i] = 255 - (icUInt16Number)i;

       ucr_char = (char *)((char *)bgCurve + bg_nbytes);
       memcpy(ucr_char, ucr_string, string_bytes);
*
*/
```

```
/*
 * Many of the structures contain variable length arrays. This
 * is represented by the use of the convention.
 *
 *      type    data[icAny];
 */

/*---------------------------------------------------------------
--------*/
/*
 * Defines used in the specification
 */
#define icMagicNumber                   0x61637370L     /* 'acsp' */
#define icVersionNumber              0x02300000L    /* 2.3.0, BCD */

#define ICC_4BYTE_ENUM0x7FFFFFFFL/* reserve 4 bytes */

/* Screening Encodings */
#define icPrtrDefaultScreensFalse       0x00000000L     /* Bit pos
0 */
#define icPrtrDefaultScreensTrue      0x00000001L    /* Bit pos 0 */
#define icLinesPerInch               0x00000002L    /* Bit pos 1 */
#define icLinesPerCm                 0x00000000L    /* Bit pos 1 */

/*
 * Device attributes, currently defined values correspond
 * to the least-significant 4 bytes of the 8 byte attribute
 * quantity, see the header for their location.
 */
#define icReflective                 0x00000000L    /* Bit pos 0 */
#define icTransparency               0x00000001L    /* Bit pos 0 */
#define icGlossy                     0x00000000L    /* Bit pos 1 */
#define icMatte                      0x00000002L    /* Bit pos 1 */
#define icPosMedia0x00000000L/* Bit pos 2 */
```

```
#define icNegMedia0x00000004L/* Bit pos 2 */
#define icColorMedia0x00000000L/* Bit pos 3 */
#define icBWMedia0x00000008L/* Bit pos 3 */
/*
 * Profile header flags, the least-significant 16 bits are reserved
 * for consortium use.
 */
#define icEmbeddedProfileFalse        0x00000000L    /* Bit pos 0 */
#define icEmbeddedProfileTrue         0x00000001L    /* Bit pos 0 */
#define icUseAnywhere                 0x00000000L    /* Bit pos 1 */
#define icUseWithEmbeddedDataOnly       0x00000002L     /* Bit pos
1 */

/* Ascii or Binary data */
#define icAsciiData                     0x00000000L
#define icBinaryData                    0x00000001L

/*
 * Define used to indicate that this is a variable length array
 */
#define icAny                           1


/*---------------------------------------------------------------
--------*/
/*
 * Use this area to translate platform definitions of long
 * etc into icXXX form. The rest of the header uses the icXXX
 * typedefs. Signatures are 4 byte quantities.
 */
#ifdef __sgi
#include “sgidefs.h”

typedef __int32_t       icSignature;
```

```
/*
 * Number definitions
 */

/* Unsigned integer numbers */
typedef unsigned char   icUInt8Number;
typedef unsigned short  icUInt16Number;
typedef __uint32_t      icUInt32Number;
typedef __uint32_t      icUInt64Number[2];

/* Signed numbers */
typedef char            icInt8Number;
typedef short           icInt16Number;
typedef __int32_t       icInt32Number;
typedef __int32_t       icInt64Number[2];

/* Fixed numbers */
typedef __int32_t       icS15Fixed16Number;
typedef __uint32_t      icU16Fixed16Number;

#else   /* default definitions */
#include <sys/isa_defs.h>
#ifdef _LP64  /*64 bit os */

typedef unsigned int    icSignature;

/*
 * Number definitions
 */

/* Unsigned integer numbers */
typedef unsigned char   icUInt8Number;
typedef unsigned short  icUInt16Number;
```

```
typedef unsigned int    icUInt32Number;
typedef unsigned int    icUInt64Number[2];

/* Signed numbers */
typedef char            icInt8Number;
typedef short           icInt16Number;
typedef int             icInt32Number;
typedef int             icInt64Number[2];

/* Fixed numbers */
typedef int             icS15Fixed16Number;
typedef unsigned int    icU16Fixed16Number;

#else   /* default definitions */

typedef long            icSignature;

/*
 * Number definitions
 */

/* Unsigned integer numbers */
typedef unsigned char   icUInt8Number;
typedef unsigned short  icUInt16Number;
typedef unsigned long   icUInt32Number;
typedef unsigned long   icUInt64Number[2];

/* Signed numbers */
typedef char            icInt8Number;
typedef short           icInt16Number;
typedef long            icInt32Number;
typedef long            icInt64Number[2];

/* Fixed numbers */
```

```
typedef long            icS15Fixed16Number;
typedef unsigned long   icU16Fixed16Number;
#endif
#endif  /* default defs */

/*------------------------------------------------------------------
--------*/
/* public tags and sizes */
typedef enum {
    icSigAToB0Tag                    = 0x41324230L,  /* 'A2B0' */
    icSigAToB1Tag                     = 0x41324231L,  /* 'A2B1' */
    icSigAToB2Tag                    = 0x41324232L,  /* 'A2B2' */
    icSigBlueColorantTag              = 0x6258595AL,  /* 'bXYZ' */
    icSigBlueTRCTag                   = 0x62545243L,  /* 'bTRC' */
    icSigBToA0Tag                     = 0x42324130L,  /* 'B2A0' */
    icSigBToA1Tag                     = 0x42324131L,  /* 'B2A1' */
    icSigBToA2Tag                     = 0x42324132L,  /* 'B2A2' */
    icSigCalibrationDateTimeTag       = 0x63616C74L,  /* 'calt' */
    icSigCharTargetTag               = 0x74617267L,  /* 'targ' */
    icSigCopyrightTag                 = 0x63707274L,  /* 'cprt' */
    icSigCrdInfoTag                   = 0x63726469L,  /* 'crdi' */
    icSigDeviceMfgDescTag             = 0x646D6E64L,  /* 'dmnd' */
    icSigDeviceModelDescTag           = 0x646D6464L,  /* 'dmdd' */
    icSigDeviceSettingsTag            = 0x64657673L,  /* 'devs' */
    icSigGamutTag                     = 0x67616D74L,  /* 'gamt' */
    icSigGrayTRCTag                   = 0x6b545243L,  /* 'kTRC' */
    icSigGreenColorantTag             = 0x6758595AL,  /* 'gXYZ' */
    icSigGreenTRCTag                  = 0x67545243L,  /* 'gTRC' */
    icSigLuminanceTag                 = 0x6C756d69L,  /* 'lumi' */
    icSigMeasurementTag               = 0x6D656173L,  /* 'meas' */
    icSigMediaBlackPointTag           = 0x626B7074L,  /* 'bkpt' */
    icSigMediaWhitePointTag           = 0x77747074L,  /* 'wtpt' */
    icSigNamedColorTag                 = 0x6E636f6CL,  /* 'ncol'
                                            * OBSOLETE, use ncl2 */
```

```
    icSigNamedColor2Tag                = 0x6E636C32L,  /* 'ncl2' */
    icSigOutputResponseTag             = 0x72657370L,  /* 'resp' */
    icSigPreview0Tag                   = 0x70726530L,  /* 'pre0' */
    icSigPreview1Tag                   = 0x70726531L,  /* 'pre1' */
    icSigPreview2Tag                   = 0x70726532L,  /* 'pre2' */
    icSigProfileDescriptionTag         = 0x64657363L,  /* 'desc' */
    icSigProfileSequenceDescTag        = 0x70736571L,  /* 'pseq' */
    icSigPs2CRD0Tag                    = 0x70736430L,  /* 'psd0' */
    icSigPs2CRD1Tag                    = 0x70736431L,  /* 'psd1' */
    icSigPs2CRD2Tag                    = 0x70736432L,  /* 'psd2' */
    icSigPs2CRD3Tag                    = 0x70736433L,  /* 'psd3' */
    icSigPs2CSATag                     = 0x70733273L,  /* 'ps2s' */
    icSigPs2RenderingIntentTag         = 0x70733269L,  /* 'ps2i' */
    icSigRedColorantTag                = 0x7258595AL,  /* 'rXYZ' */
    icSigRedTRCTag                     = 0x72545243L,  /* 'rTRC' */
    icSigScreeningDescTag              = 0x73637264L,  /* 'scrd' */
    icSigScreeningTag                  = 0x7363726EL,  /* 'scrn' */
    icSigTechnologyTag                 = 0x74656368L,  /* 'tech' */
    icSigUcrBgTag                      = 0x62666420L,  /* 'bfd ' */
    icSigViewingCondDescTag            = 0x76756564L,  /* 'vued' */
    icSigViewingConditionsTag          = 0x76696577L,  /* 'view' */
    icSigChromaticityTag= 0x6368726DL,/* 'chrm' */
    icMaxEnumTag                        = 0x7FFFFFFFL
} icTagSignature;

/* technology signature descriptions */
typedef enum {
    icSigDigitalCamera                 = 0x6463616DL,  /* 'dcam' */
    icSigFilmScanner                   = 0x6673636EL,  /* 'fscn' */
    icSigReflectiveScanner             = 0x7273636EL,  /* 'rscn' */
    icSigInkJetPrinter                = 0x696A6574L,  /* 'ijet' */
    icSigThermalWaxPrinter             = 0x74776178L,  /* 'twax' */
    icSigElectrophotographicPrinter    = 0x6570686FL,  /* 'epho' */
    icSigElectrostaticPrinter          = 0x65737461L,  /* 'esta' */
```

```
    icSigDyeSublimationPrinter         = 0x64737562L,  /* 'dsub' */
    icSigPhotographicPaperPrinter      = 0x7270686FL,  /* 'rpho' */
    icSigFilmWriter                    = 0x6670726EL,  /* 'fprn' */
    icSigVideoMonitor                  = 0x7669646DL,  /* 'vidm' */
    icSigVideoCamera                   = 0x76696463L,  /* 'vidc' */
    icSigProjectionTelevision          = 0x706A7476L,  /* 'pjtv' */
    icSigCRTDisplay                    = 0x43525420L,  /* 'CRT ' */
    icSigPMDisplay                     = 0x504D4420L,  /* 'PMD ' */
    icSigAMDisplay                     = 0x414D4420L,  /* 'AMD ' */
    icSigPhotoCD                       = 0x4B504344L,  /* 'KPCD' */
    icSigPhotoImageSetter              = 0x696D6773L,  /* 'imgs' */
    icSigGravure                       = 0x67726176L,  /* 'grav' */
    icSigOffsetLithography             = 0x6F666673L,  /* 'offs' */
    icSigSilkscreen                    = 0x73696C6BL,  /* 'silk' */
    icSigFlexography                   = 0x666C6578L,  /* 'flex' */
    icMaxEnumTechnology                 = 0x7FFFFFFFL
} icTechnologySignature;

/* type signatures */
typedef enum {
    icSigCurveType                     = 0x63757276L,  /* 'curv' */
    icSigDataType                      = 0x64617461L,  /* 'data' */
    icSigDateTimeType                  = 0x6474696DL,  /* 'dtim' */
    icSigDeviceSettingsType            = 0x64657673L,  /* 'devs' */
    icSigLut16Type                     = 0x6d667432L,  /* 'mft2' */
    icSigLut8Type                      = 0x6d667431L,  /* 'mft1' */
    icSigMeasurementType               = 0x6D656173L,  /* 'meas' */
    icSigNamedColorType                 = 0x6E636f6CL,  /* 'ncol'
                                                        * OBSOLETE, use ncl2 */
    icSigProfileSequenceDescType       = 0x70736571L,  /* 'pseq' */
    icSigResponseCurveSet16Type        = 0x72637332L,  /* 'rcs2' */
    icSigS15Fixed16ArrayType           = 0x73663332L,  /* 'sf32' */
    icSigScreeningType                 = 0x7363726EL,  /* 'scrn' */
    icSigSignatureType                 = 0x73696720L,  /* 'sig ' */
```

```
    icSigTextType                       = 0x74657874L,  /* 'text' */
    icSigTextDescriptionType            = 0x64657363L,  /* 'desc' */
    icSigU16Fixed16ArrayType            = 0x75663332L,  /* 'uf32' */
    icSigUcrBgType                      = 0x62666420L,  /* 'bfd ' */
    icSigUInt16ArrayType                = 0x75693136L,  /* 'ui16' */
    icSigUInt32ArrayType                = 0x75693332L,  /* 'ui32' */
    icSigUInt64ArrayType                = 0x75693634L,  /* 'ui64' */
    icSigUInt8ArrayType                 = 0x75693038L,  /* 'ui08' */
    icSigViewingConditionsType          = 0x76696577L,  /* 'view' */
    icSigXYZType                        = 0x58595A20L,  /* 'XYZ ' */
    icSigXYZArrayType                   = 0x58595A20L,  /* 'XYZ ' */
    icSigNamedColor2Type                = 0x6E636C32L,  /* 'ncl2' */
    icSigCrdInfoType                    = 0x63726469L,  /* 'crdi' */
    icSigChromaticityType= 0x6368726DL,/* 'chrm' */
    icMaxEnumType                        = 0x7FFFFFFFL
} icTagTypeSignature;

/*
 * Color Space Signatures
 * Note that only icSigXYZData and icSigLabData are valid
 * Profile Connection Spaces (PCSs)
 */
typedef enum {
    icSigXYZData                        = 0x58595A20L,  /* 'XYZ ' */
    icSigLabData                        = 0x4C616220L,  /* 'Lab ' */
    icSigLuvData                        = 0x4C757620L,  /* 'Luv ' */
    icSigYCbCrData                      = 0x59436272L,  /* 'YCbr' */
    icSigYxyData                        = 0x59787920L,  /* 'Yxy ' */
    icSigRgbData                        = 0x52474220L,  /* 'RGB ' */
    icSigGrayData                       = 0x47524159L,  /* 'GRAY' */
    icSigHsvData                        = 0x48535620L,  /* 'HSV ' */
    icSigHlsData                        = 0x484C5320L,  /* 'HLS ' */
    icSigCmykData                       = 0x434D594BL,  /* 'CMYK' */
    icSigCmyData                        = 0x434D5920L,  /* 'CMY ' */
```

```
    icSig2colorData                 = 0x32434C52L,  /* '2CLR' */
    icSig3colorData                 = 0x33434C52L,  /* '3CLR' */
    icSig4colorData                 = 0x34434C52L,  /* '4CLR' */
    icSig5colorData                 = 0x35434C52L,  /* '5CLR' */
    icSig6colorData                 = 0x36434C52L,  /* '6CLR' */
    icSig7colorData                 = 0x37434C52L,  /* '7CLR' */
    icSig8colorData                 = 0x38434C52L,  /* '8CLR' */
    icSig9colorData                 = 0x39434C52L,  /* '9CLR' */
    icSig10colorData                = 0x41434C52L,  /* 'ACLR' */
    icSig11colorData                = 0x42434C52L,  /* 'BCLR' */
    icSig12colorData                = 0x43434C52L,  /* 'CCLR' */
    icSig13colorData                = 0x44434C52L,  /* 'DCLR' */
    icSig14colorData                = 0x45434C52L,  /* 'ECLR' */
    icSig15colorData                = 0x46434C52L,  /* 'FCLR' */
    icMaxEnumData                    = 0x7FFFFFFFL
} icColorSpaceSignature;

/* profileClass enumerations */
typedef enum {
    icSigInputClass                 = 0x73636E72L,  /* 'scnr' */
    icSigDisplayClass               = 0x6D6E7472L,  /* 'mntr' */
    icSigOutputClass                = 0x70727472L,  /* 'prtr' */
    icSigLinkClass                  = 0x6C696E6BL,  /* 'link' */
    icSigAbstractClass              = 0x61627374L,  /* 'abst' */
    icSigColorSpaceClass            = 0x73706163L,  /* 'spac' */
    icSigNamedColorClass            = 0x6e6d636cL,  /* 'nmcl' */
    icMaxEnumClass                   = 0x7FFFFFFFL
} icProfileClassSignature;

/* Platform Signatures */
typedef enum {
    icSigMacintosh                  = 0x4150504CL,  /* 'APPL' */
    icSigMicrosoft                  = 0x4D534654L,  /* 'MSFT' */
    icSigSolaris                    = 0x53554E57L,  /* 'SUNW' */
```

```
    icSigSGI                            = 0x53474920L,  /* 'SGI ' */
    icSigTaligent                       = 0x54474E54L,  /* 'TGNT' */
    icMaxEnumPlatform                    = 0x7FFFFFFFL
} icPlatformSignature;

/*---------------------------------------------------------------------
--------*/
/*
 * Other enums
 */

/* Measurement Geometry, used in the measurmentType tag */
typedef enum {
    icGeometryUnknown                  = 0x00000000L,  /* Unknown */
   icGeometry045or450                 = 0x00000001L,  /* 0/45, 45/0 */
   icGeometry0dord0                   = 0x00000002L,  /* 0/d or d/0 */
    icMaxGeometry                        = 0x7FFFFFFFL
} icMeasurementGeometry;

/* Rendering Intents, used in the profile header */
typedef enum {
    icPerceptual                        = 0,
    icRelativeColorimetric              = 1,
    icSaturation                        = 2,
    icAbsoluteColorimetric              = 3,
    icMaxEnumIntent                     = 0x7FFFFFFFL
} icRenderingIntent;

/* Different Spot Shapes currently defined, used for screeningType
*/
typedef enum {
    icSpotShapeUnknown                  = 0,
    icSpotShapePrinterDefault           = 1,
    icSpotShapeRound                    = 2,
```

```
    icSpotShapeDiamond                  = 3,
    icSpotShapeEllipse                  = 4,
    icSpotShapeLine                     = 5,
    icSpotShapeSquare                   = 6,
    icSpotShapeCross                    = 7,
    icMaxEnumSpot                       = 0x7FFFFFFFL
} icSpotShape;

/* Standard Observer, used in the measurmentType tag */
typedef enum {
    icStdObsUnknown                   = 0x00000000L,  /* Unknown */
    icStdObs1931TwoDegrees              = 0x00000001L,  /* 2 deg */
    icStdObs1964TenDegrees             = 0x00000002L,  /* 10 deg */
    icMaxStdObs                         = 0x7FFFFFFFL
} icStandardObserver;

/* Pre-defined illuminants, used in measurement and viewing
conditions type */
typedef enum {
    icIlluminantUnknown                 = 0x00000000L,
    icIlluminantD50                     = 0x00000001L,
    icIlluminantD65                     = 0x00000002L,
    icIlluminantD93                     = 0x00000003L,
    icIlluminantF2                      = 0x00000004L,
    icIlluminantD55                     = 0x00000005L,
    icIlluminantA                       = 0x00000006L,
    icIlluminantEquiPowerE              = 0x00000007L,
    icIlluminantF8                      = 0x00000008L,
    icMaxEnumIluminant                  = 0x7FFFFFFFL
} icIlluminant;

/* media type for icSigDeviceSettingsTag */
typedef enum {
    icStandard                          = 1,
```

```
    icTrans                             = 2,  /* transparency */
    icGloss                             = 3,
    icUser1                             = 256,
    icMaxDeviceMedia                    = 0x7FFFFFFFL
} icDeviceMedia;

/* halftone settings for icSigDeviceSettingTag  */
typedef enum {
    icNone                              = 1,
    icCoarse                            = 2,
    icFine                              = 3,
    icLineArt                           = 4,
    icErrorDiffusion                    = 5,
    icReserved6                         = 6,
    icReserved7                         = 7,
    icReserved8                         = 8,
    icReserved9                         = 9,
    icGrayScale                         = 10,
    icUser2                             = 256,
    icMaxDither                         = 0x7FFFFFFFL
} icDeviceDither;

/* signatures for icSigDeviceSettingsTag  */
typedef enum {
    icSigResolution                   = 0x72736c6eL,  /* 'rsln' */
    icSigMedia                        = 0x6d747970L,  /* 'mtyp' */
    icSigHalftone                     = 0x6866746eL,  /* 'hftn' */
    icMaxSettings                      = 0x7FFFFFFFL
} icSettingsSig;

/* measurement units for the icResponseCurveSet16Type */
typedef enum {
    icStaA                            = 0x53746141L,  /* 'StaA' */
    icStaE                            = 0x53746145L,  /* 'StaE' */
```

```
    icStaI                              = 0x53746149L,  /* 'StaI' */
    icStaT                              = 0x53746154L,  /* 'StaT' */
    icStaM                              = 0x5374614dL,  /* 'StaM' */
    icDN                                = 0x444e2020L,  /* 'DN  ' */
    icDNP                               = 0x444e2050L,  /* 'DN P' */
    icDNN                               = 0x444e4e20L,  /* 'DNN ' */
    icDNNP                              = 0x444e4e50L,  /* 'DNNP' */
    icMaxUnits                           = 0x7FFFFFFFL
} icMeasUnitsSig;

typedef enum {
    icUnknown  = 0x0000L,/* unknown */
    icITURBT709= 0x0001L,/* ITU-R BT.709 */
    icSMPTERP1451994= 0x0002L,/* SMPTE RP145-1994 */
    icEBUTech3213E= 0x0003L,/* EBU Tech.3213-E */
    icP22      = 0x0004L,/* P22 */
    icMaxPhosCol= 0xFFFFL
} icPhosColType;

/*---------------------------------------------------------------
--------*/
/*
 * Arrays of numbers
 */

/* Int8 Array */
typedef struct {
   icInt8Number       data[icAny];    /* Variable array of values */
} icInt8Array;

/* UInt8 Array */
typedef struct {
   icUInt8Number      data[icAny];    /* Variable array of values */
} icUInt8Array;
```

```
/* uInt16 Array */
typedef struct {
   icUInt16Number     data[icAny];   /* Variable array of values */
} icUInt16Array;

/* Int16 Array */
typedef struct {
   icInt16Number      data[icAny];   /* Variable array of values */
} icInt16Array;

/* uInt32 Array */
typedef struct {
   icUInt32Number     data[icAny];   /* Variable array of values */
} icUInt32Array;

/* Int32 Array */
typedef struct {
   icInt32Number      data[icAny];   /* Variable array of values */
} icInt32Array;

/* UInt64 Array */
typedef struct {
   icUInt64Number     data[icAny];   /* Variable array of values */
} icUInt64Array;

/* Int64 Array */
typedef struct {
   icInt64Number      data[icAny];   /* Variable array of values */
} icInt64Array;

/* u16Fixed16 Array */
typedef struct {
   icU16Fixed16Number  data[icAny];   /* Variable array of values */
```

```
} icU16Fixed16Array;

/* s15Fixed16 Array */
typedef struct {
   icS15Fixed16Number  data[icAny];   /* Variable array of values */
} icS15Fixed16Array;

/* The base date time number */
typedef struct {
    icUInt16Number      year;
    icUInt16Number      month;
    icUInt16Number      day;
    icUInt16Number      hours;
    icUInt16Number      minutes;
    icUInt16Number      seconds;
} icDateTimeNumber;

/* XYZ Number  */
typedef struct {
    icS15Fixed16Number  X;
    icS15Fixed16Number  Y;
    icS15Fixed16Number  Z;
} icXYZNumber;

/* XYZ Array */
typedef struct {
    icXYZNumber         data[icAny];    /* Variable array of XYZ
numbers */
} icXYZArray;

/* Curve */
typedef struct {
    icUInt32Number      count;          /* Number of entries */
   icUInt16Number     data[icAny];   /* The actual table data, real
```

```
                                        * number is determined by count
                                        * Interpretation depends on how
                                        * data is used with a given tag
                                            */
} icCurve;

/* Data */
typedef struct {
    icUInt32Number      dataFlag;       /* 0 = ascii, 1 = binary */
    icInt8Number        data[icAny];    /* Data, size from tag */
} icData;

/* lut16 */
typedef struct {
   icUInt8Number      inputChan;     /* Number of input channels */
   icUInt8Number      outputChan;    /* Number of output channels */
    icUInt8Number       clutPoints;     /* Number of grid points */
   icInt8Number       pad;          /* Padding for byte alignment */
    icS15Fixed16Number  e00;            /* e00 in the 3 * 3 */
    icS15Fixed16Number  e01;            /* e01 in the 3 * 3 */
    icS15Fixed16Number  e02;            /* e02 in the 3 * 3 */
    icS15Fixed16Number  e10;            /* e10 in the 3 * 3 */
    icS15Fixed16Number  e11;            /* e11 in the 3 * 3 */
    icS15Fixed16Number  e12;            /* e12 in the 3 * 3 */
    icS15Fixed16Number  e20;            /* e20 in the 3 * 3 */
    icS15Fixed16Number  e21;            /* e21 in the 3 * 3 */
    icS15Fixed16Number  e22;            /* e22 in the 3 * 3 */
   icUInt16Number     inputEnt;      /* Num of in-table entries */
   icUInt16Number     outputEnt;     /* Num of out-table entries */
    icUInt16Number      data[icAny];    /* Data follows see spec */
/*
 *  Data that follows is of this form
 *
 *  icUInt16Number    inputTable[inputChan][icAny];   * The in-table
```

```
 *  icUInt16Number      clutTable[icAny];               * The clut
 *  icUInt16Number      outputTable[outputChan][icAny]; * The out-
table
 */
} icLut16;

/* lut8, input & output tables are always 256 bytes in length */
typedef struct {
    icUInt8Number       inputChan;      /* Num of input channels */
    icUInt8Number       outputChan;     /* Num of output channels */
    icUInt8Number       clutPoints;     /* Num of grid points */
    icInt8Number        pad;
    icS15Fixed16Number  e00;            /* e00 in the 3 * 3 */
    icS15Fixed16Number  e01;            /* e01 in the 3 * 3 */
    icS15Fixed16Number  e02;            /* e02 in the 3 * 3 */
    icS15Fixed16Number  e10;            /* e10 in the 3 * 3 */
    icS15Fixed16Number  e11;            /* e11 in the 3 * 3 */
    icS15Fixed16Number  e12;            /* e12 in the 3 * 3 */
    icS15Fixed16Number  e20;            /* e20 in the 3 * 3 */
    icS15Fixed16Number  e21;            /* e21 in the 3 * 3 */
    icS15Fixed16Number  e22;            /* e22 in the 3 * 3 */
    icUInt8Number       data[icAny];    /* Data follows see spec */
/*
 *  Data that follows is of this form
 *
 *  icUInt8Number      inputTable[inputChan][256];    * The in-table
 *  icUInt8Number       clutTable[icAny];               * The clut
 *  icUInt8Number       outputTable[outputChan][256];   * The out-
table
 */
} icLut8;

/* Measurement Data */
typedef struct {
```

```
   icStandardObserver        stdObserver;   /* Standard observer */
    icXYZNumber                backing;       /* XYZ for backing */
    icMeasurementGeometry      geometry;      /* Meas. geometry */
    icU16Fixed16Number        flare;        /* Measurement flare */
    icIlluminant               illuminant;     /* Illuminant */
} icMeasurement;

/* Named color */

/*
 * icNamedColor2 takes the place of icNamedColor
 */
typedef struct {
   icUInt32Number     vendorFlag;    /* Bottom 16 bits for IC use */
    icUInt32Number      count;          /* Count of named colors */
   icUInt32Number     nDeviceCoords;  /* Num of device coordinates
*/
   icInt8Number      prefix[32];    /* Prefix for each color name */
   icInt8Number      suffix[32];    /* Suffix for each color name */
   icInt8Number       data[icAny];   /* Named color data follows */
/*
 *  Data that follows is of this form
 *
 * icInt8Number       root1[32];           * Root name for 1st color
 * icUInt16Number      pcsCoords1[icAny];      * PCS coords of 1st
color
 * icUInt16Number      deviceCoords1[icAny];   * Dev coords of 1st
color
 * icInt8Number       root2[32];           * Root name for 2nd color
 * icUInt16Number      pcsCoords2[icAny];      * PCS coords of 2nd
color
 * icUInt16Number      deviceCoords2[icAny];   * Dev coords of 2nd
color
 *                     :
 *                     :
```

```
 * Repeat for name and PCS and device color coordinates up to (count-
1)
 *
 * NOTES:
 * PCS and device space can be determined from the header.
 *
 * PCS coordinates are icUInt16 numbers and are described in Annex
A of
 * the ICC spec. Only 16 bit L*a*b* and XYZ are allowed. The number of
 * coordinates is consistent with the headers PCS.
 *
 * Device coordinates are icUInt16 numbers where 0x0000 represents
 * the minimum value and 0xFFFF represents the maximum value.
 * If the nDeviceCoords value is 0 this field is not given.
 */
} icNamedColor2;

/* Profile sequence structure */
typedef struct {
    icSignature               deviceMfg;      /* Dev Manufacturer */
     icSignature                 deviceModel;    /* Dev Model */
    icUInt64Number             attributes;     /* Dev attributes */
    icTechnologySignature      technology;     /* Technology sig */
   icInt8Number              data[icAny];    /* Desc text follows */
/*
 *  Data that follows is of this form, this is an icInt8Number
 *  to avoid problems with a compiler generating  bad code as
 *  these arrays are variable in length.
 *
 * icTextDescriptionType        deviceMfgDesc;  * Manufacturer text
 * icTextDescriptionType        modelDesc;      * Model text
 */
} icDescStruct;
```

```
/* Profile sequence description */
typedef struct {
    icUInt32Number      count;          /* Number of descriptions */
    icUInt8Number       data[icAny];    /* Array of desc structs */
} icProfileSequenceDesc;

/* textDescription */
typedef struct {
    icUInt32Number      count;          /* Description length */
    icInt8Number        data[icAny];    /* Descriptions follow */
/*
 *  Data that follows is of this form
 *
 * icInt8Number       desc[count]    * NULL terminated ascii string
 * icUInt32Number       ucLangCode;     * UniCode language code
 * icUInt32Number      ucCount;        * UniCode description length
 * icInt16Number        ucDesc[ucCount];* The UniCode description
 * icUInt16Number       scCode;         * ScriptCode code
 * icUInt8Number        scCount;        * ScriptCode count
 * icInt8Number         scDesc[67];     * ScriptCode Description
 */
} icTextDescription;

/* Screening Data */
typedef struct {
    icS15Fixed16Number  frequency;      /* Frequency */
    icS15Fixed16Number  angle;          /* Screen angle */
   icSpotShape       spotShape;     /* Spot Shape encodings below */
} icScreeningData;

typedef struct {
    icUInt32Number      screeningFlag;  /* Screening flag */
    icUInt32Number      channels;       /* Number of channels */
   icScreeningData    data[icAny];    /* Array of screening data */
```

```
} icScreening;

/* Text Data */
typedef struct {
    icInt8Number       data[icAny];    /* Variable array of chars */
} icText;

/* Structure describing either a UCR or BG curve */
typedef struct {
    icUInt32Number      count;          /* Curve length */
   icUInt16Number     curve[icAny];   /* The array of curve values */
} icUcrBgCurve;

/* Under color removal, black generation */
typedef struct {
    icInt8Number       data[icAny];           /* The Ucr BG data */
/*
 *  Data that follows is of this form, this is a icInt8Number
 *  to avoid problems with a compiler generating  bad code as
 *  these arrays are variable in length.
 *
 * icUcrBgCurve         ucr;            * Ucr curve
 * icUcrBgCurve         bg;             * Bg curve
 * icInt8Number         string;         * UcrBg description
 */
} icUcrBg;

/* viewingConditionsType */
typedef struct {
   icXYZNumber        illuminant;    /* In candelas per sq. meter */
   icXYZNumber        surround;      /* In candelas per sq. meter */
   icIlluminant       stdIluminant;   /* See icIlluminant defines */
} icViewingCondition;
```

```
/* CrdInfo type */
typedef struct {
    icUInt32Number     count;         /* Char count includes NULL */
    icInt8Number       data[icAny];    /* Null terminated string */
} icCrdInfo;

/* support structures for the icSigDeviceSettingsTag */
typedef struct {
    icUInt32Number      numPlatforms;   /* number of platforms */
    icUInt32Number      data[icAny];
}icSettingsData;

/* where data is “numPlatforms” of the following structure
 *
 *typedef struct {
 *  icPlatformSignature platform;
 *  icUInt32Number      size;         total size of all settings
 *  icUInt32Number      combCount;    # of settings
 *  icSettingsStruct    data[icAny];
 *};
 *
 *  where data is “combCount” of the following structure
 *
 *typedef struct {
 *  icUInt32Number      structSize;   size in bytes of entire
structure
 *  icUInt32Number    numStructs;  # of setting structures inlcuded
 *  icSettings          data[icAny];
 *}icSettingsStruct;
 *
 *  where data is “numStructs” of the following structure
 *
 *typedef struct {
 *  icSettingsSig       settingSig;
```

```
 *  icUInt32Number     size;        size in bytes per setting value
 *  icUInt32Number      numSettings;  number of seting values
 *  icUInt32Number      data[icAny];
 *}icSettings;
 *
 *  where data is “numsettings” of one of the following:
 *  icUInt64Number      resolution;
 *  icDeviceMedia       media;
 *  icDeviceDither      halftone;
*/

/* for use with the icResponseCurveSet16Type */
typedef struct {
    icUInt16Number      channels;       /* number of channels */
    icUInt16Number      numTypes;       /* count of meas. types */
    icUInt32Number      data[icAny];
}icResponse;

/* where data is “numTypes” of the following
 *  icMeasUnitsSig      sigType;
 *  icUInt32Number      numMeas;   one entry for each “channels”
 *  icXYZNumber        meas;     one xyz entry for each “channels”
 *                                     respective “numMeas”
 *  icResponse16Number  respNum;   one structure for each “channels”
 *                                     respective “numMeas”
 */

typedef struct {
   icUInt16Number    interval;     /* device value scaled 0-FFFF */
    icUInt16Number      pad;            /* 0 */
   icS15Fixed16Number  measurement;   /* actual measurement value */
} icResponse16Number;

typedef struct {
```

```
    icUInt16Numberchannels;/* number of channels */
    icUInt16Numbertype;/* phosphor/colorant type */
    icU16Fixed16Number data[icAny];/* array of x and y coordinates
*/
} icChromaticity;


/*---------------------------------------------------------------
--------*/
/*
 * Tag Type definitions
 */

/*
 * Many of the structures contain variable length arrays. This
 * is represented by the use of the convention.
 *
 *      type    data[icAny];
 */

/* The base part of each tag */
typedef struct {
    icTagTypeSignature  sig;            /* Signature */
    icInt8Number        reserved[4];    /* Reserved, set to 0 */
} icTagBase;

/* curveType */
typedef struct {
    icTagBase           base;           /* Signature, “curv” */
    icCurve             curve;          /* The curve data */
} icCurveType;

/* dataType */
typedef struct {
```

```
    icTagBase           base;           /* Signature, “data” */
    icData              data;           /* The data structure */
} icDataType;

/* dateTimeType */
typedef struct {
    icTagBase           base;           /* Signature, “dtim” */
    icDateTimeNumber    date;           /* The date */
} icDateTimeType;

/* lut16Type */
typedef struct {
    icTagBase           base;           /* Signature, “mft2” */
    icLut16             lut;            /* Lut16 data */
} icLut16Type;

/* lut8Type, input & output tables are always 256 bytes in length */
typedef struct {
    icTagBase           base;           /* Signature, “mft1” */
    icLut8              lut;            /* Lut8 data */
} icLut8Type;

/* Measurement Type */
typedef struct {
    icTagBase           base;           /* Signature, “meas” */
    icMeasurement       measurement;    /* Measurement data */
} icMeasurementType;

/* Named color type */
/* icNamedColor2Type, replaces icNamedColorType */
typedef struct {
    icTagBase           base;           /* Signature, “ncl2” */
    icNamedColor2       ncolor;         /* Named color data */
} icNamedColor2Type;
```

```
/* Profile sequence description type */
typedef struct {
    icTagBase                   base;   /* Signature, “pseq” */
    icProfileSequenceDesc       desc;   /* The seq description */
} icProfileSequenceDescType;

/* textDescriptionType */
typedef struct {
    icTagBase                   base;   /* Signature, “desc” */
    icTextDescription           desc;   /* The description */
} icTextDescriptionType;

/* s15Fixed16Type */
typedef struct {
    icTagBase           base;           /* Signature, “sf32” */
    icS15Fixed16Array   data;           /* Array of values */
} icS15Fixed16ArrayType;

typedef struct {
    icTagBase           base;           /* Signature, “scrn” */
    icScreening         screen;         /* Screening structure */
} icScreeningType;

/* sigType */
typedef struct {
    icTagBase           base;           /* Signature, “sig” */
    icSignature         signature;      /* The signature data */
} icSignatureType;

/* textType */
typedef struct {
    icTagBase           base;           /* Signature, “text” */
    icText             data;          /* Variable array of chars */
```

```
} icTextType;

/* u16Fixed16Type */
typedef struct {
    icTagBase           base;           /* Signature, “uf32” */
    icU16Fixed16Array   data;         /* Variable array of values */
} icU16Fixed16ArrayType;

/* Under color removal, black generation type */
typedef struct {
    icTagBase           base;           /* Signature, “bfd “ */
    icUcrBg             data;           /* ucrBg structure */
} icUcrBgType;

/* uInt16Type */
typedef struct {
    icTagBase           base;           /* Signature, “ui16” */
    icUInt16Array      data;         /* Variable array of values */
} icUInt16ArrayType;

/* uInt32Type */
typedef struct {
    icTagBase           base;           /* Signature, “ui32” */
    icUInt32Array      data;         /* Variable array of values */
} icUInt32ArrayType;

/* uInt64Type */
typedef struct {
    icTagBase           base;           /* Signature, “ui64” */
    icUInt64Array      data;         /* Variable array of values */
} icUInt64ArrayType;

/* uInt8Type */
typedef struct {
```

```
    icTagBase           base;           /* Signature, “ui08” */
    icUInt8Array       data;          /* Variable array of values */
} icUInt8ArrayType;

/* viewingConditionsType */
typedef struct {
    icTagBase           base;           /* Signature, “view” */
    icViewingCondition  view;           /* Viewing conditions */
} icViewingConditionType;

/* XYZ Type */
typedef struct {
    icTagBase           base;           /* Signature, “XYZ” */
    icXYZArray        data;         /* Variable array of XYZ nums */
} icXYZType;

/* CRDInfoType where [0] is the CRD product name count and string and
 * [1] -[5] are the rendering intents 0-4 counts and strings
 */
typedef struct {
    icTagBase           base;           /* Signature, “crdi” */
    icCrdInfo         info;         /* 5 sets of counts & strings */
}icCrdInfoType;
     /*   icCrdInfo      productName;     PS product count/string */
     /*   icCrdInfo       CRDName0;        CRD name for intent 0 */
     /*   icCrdInfo       CRDName1;        CRD name for intent 1 */
     /*   icCrdInfo       CRDName2;        CRD name for intent 2 */
     /*   icCrdInfo       CRDName3;        CRD name for intent 3 */

typedef struct {
    icTagBase           base;           /* Signature, ‘devs’ */
    icSettingsData      data;
} icDeviceSettingsType;
```

```
typedef struct {
    icTagBase           base;           /* Signature, 'rcs2' */
    icResponse          data;
} icResponseCurveSet16Type;

/* where data is structured as follows
 * icUInt16Number  channels;          number of channels
 * icUInt16Number  numTypes;          count of measurement types
 * icUInt32Number  offset[numTypes];  offset from byte 0 of tag to
each
 *                                      response data set
 *
 * plus one or more of the following structures
 * typedef struct {
 *   icMeasUnitsSig     measurementUnit;       sig of the meas. unit
 *   icUInt32Number     perChannel[channels];   # of meas's per chan
 *   icXYZNumber       measure[channels];     measurements of patch
 *                                             w/max colorant value
 *    icResponse16Number  response[channels][perChannel[channels]];
 *    }
 */

typedef struct {
    icTagBasebase;/* Signature, 'chrm' */
    icChromaticitychromaticity;/* Chromaticity data */
} icChromaticityType;

/*-----------------------------------------------------------------
--------*/

/*
 * Lists of tags, tags, profile header and profile structure
 */
```

```
/* A tag */
typedef struct {
    icTagSignature      sig;            /* The tag signature */
    icUInt32Number      offset;         /* Start of tag relative to
                                         * start of header, Spec
                                         * Clause 5 */
    icUInt32Number      size;           /* Size in bytes */
} icTag;

/* A Structure that may be used independently for a list of tags */
typedef struct {
    icUInt32Number      count;          /* Num tags in the profile */
    icTag               tags[icAny];    /* Variable array of tags */
} icTagList;

/* The Profile header */
typedef struct {
    icUInt32Number            size;          /* Prof size in bytes */
    icSignature               cmmId;           /* CMM for profile */
    icUInt32Number            version;         /* Format version */
    icProfileClassSignature   deviceClass;    /* Type of profile */
    icColorSpaceSignature     colorSpace;    /* Clr space of data */
    icColorSpaceSignature     pcs;             /* PCS, XYZ or Lab */
    icDateTimeNumber          date;            /* Creation Date */
    icSignature               magic;           /* icMagicNumber */
    icPlatformSignature       platform;      /* Primary Platform */
    icUInt32Number            flags;           /* Various bits */
    icSignature               manufacturer;   /* Dev manufacturer */
    icUInt32Number            model;         /* Dev model number */
    icUInt64Number            attributes;    /* Device attributes */
    icUInt32Number            renderingIntent;/* Rendering intent */
    icXYZNumber               illuminant;    /* Profile illuminant */
    icSignature               creator;       /* Profile creator */
```

```
    icInt8Number                reserved[44];   /* Reserved */
} icHeader;

/*
 * A profile,
 * we can't use icTagList here because its not at the end of the
structure
 */
typedef struct {
    icHeader            header;         /* The header */
    icUInt32Number     count;          /* Num tags in the profile */
    icInt8Number       data[icAny];    /* The tagTable and tagData */
/*
 * Data that follows is of the form
 *
 * icTag        tagTable[icAny];        * The tag table
 * icInt8Number tagData[icAny];         * The tag data
 */
} icProfile;

/*-----------------------------------------------------------------
--------*/
/* Obsolete entries */

/* icNamedColor was replaced with icNamedColor2 */
typedef struct {
   icUInt32Number     vendorFlag;    /* Bottom 16 bits for IC use */
    icUInt32Number      count;          /* Count of named colors */
    icInt8Number       data[icAny];    /* Named color data follows */
/*
 *  Data that follows is of this form
 *
 * icInt8Number         prefix[icAny];  * Prefix
 * icInt8Number         suffix[icAny];  * Suffix
```

```
 * icInt8Number         root1[icAny];   * Root name
 * icInt8Number         coords1[icAny]; * Color coordinates
 * icInt8Number         root2[icAny];   * Root name
 * icInt8Number         coords2[icAny]; * Color coordinates
 *                      :
 *                      :
 * Repeat for root name and color coordinates up to (count-1)
 */
} icNamedColor;

/* icNamedColorType was replaced by icNamedColor2Type */
typedef struct {
    icTagBase           base;           /* Signature, “ncol” */
    icNamedColor        ncolor;         /* Named color data */
} icNamedColorType;

#endif /* ICC_H */
```

Index