ICC Profile Format Specification

Version 3.4 -- August 15, 1997

International Color Consortium

Copyright Notice

Licenses and Trademarks

For additional information on the ICC:

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

Tim Kohler, ICC Chairman

Canon Information Systems
20300 Stevens Creek Blvd., Suite 100
Cupertino, CA 95014

Phone:(408)-342-2223
Fax: (408)-342-2260
Email: tkohler@canon.com

Founding Members

Adobe Systems Inc.
Agfa-Gevaert N.V.
Apple Computer, Inc.
Eastman Kodak Company
FOGRA (Honorary)
Microsoft Corporation
Silicon Graphics, Inc.
Sun Microsystems, Inc.
Taligent, Inc. (resigned)

Regular Members

Anitec Imaging Products
Barco Graphics N.V.
Canon Information Systems
ColorAge, Inc.
Color Savvy Systems, Inc.
Color Solutions, Inc.
Corbis Corporation
Cymbolic Sciences International
Dainippon Screen Corporate
DuPont Electronic Imaging
DynaLab, Inc.
Fuji Photo Film U.S.A., Inc.
Fujifilm Electronic Imaging, Inc.
Fujitsu Laboratories Ltd.
GretagMacbeth
Harlequin, Inc.
Hewlett Packard Company
Imation Enterprises Corp.
Konica Corporation
Lexmark International
Light Source, Inc.
Linotype-CPS
LOGO
Management Graphics, Inc.
MGI Software Corp.
NEC Corporation
Okidata
Onyx Graphics Corp.
Pantone, Inc.
PhotoDisc Inc.
Polaroid Corporation
Praxisoft, Inc.
Scitex Corporation Ltd.
Seiko Epson Corp.
Sonnetech Ltd.
Sony Corporation
Toppan Printing Co., Ltd.
Toyo Ink Mfg. Co., Ltd.
Trumatch, Inc.
ViewSonic Corp.
VISU Technologies
X-Rite, Incorporated
Xerox Corporation
Xionics Document Technologies

Note: This list is correct at time of publication. See the ICC Web Site for the most up-to-date member list.

Table of Contents

0 Introduction

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

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

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

0.1 Intended Audience

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

0.2 Organizational Description of This Specification

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

Clause 1 describes the scope of this specification.

Clause 2 provides the normative references for this specification.

Clause 3 describes the conformance requirements for this specification.

Clause 4 provides general definitions used within this standard.

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

Clause 6 describes the requirements of this specification. Sub-clause 6.1: “Header Description” describes the format header definition. Sub-clause 6.2: “Tag Table Definition” describes the tag table. Sub-clause 6.3: “Required Tags for Profiles” provides a top level view of what tags are required for each type of profile classification and a brief description of the algorithmic models associated with these classes. Four additional color transformation formats are also described: device link, color space conversion, abstract transformations, and named color transforms. Sub-clause 6.4: “Tag Descriptions” is a detailed algorithmic and intent description of all of the tagged elements described in the previous clauses. Sub-clause 6.5: “Tag Type Definitions” provides a byte stream definition of the structures that make up the tags in sub-clause 6.4.

Annex A: “Color Spaces” describes the color spaces used in this specification. Annex B: “Embedding Profiles” provides the necessary details to embed profiles into PICT, EPS, TIFF, and JFIF files. Annex C: “C Header File Example” provides cross-platform ANSI-C compatible header file example for each of the device profile and color transform formats. Annex D: “PostScript Level 2 Tags” provides a general description of the PostScript Level 2 tags used in this specification. Annex E: “Profile Connection Space Explanation” is a paper describing details of the profile connection space. Annex F: “Summary of Spec Changes” is a summary of the changes made since the last version of the spec.

0.3 International Color Consortium

Considering the potential impact of this standard on various industries, a consortium has been formed that will administer this specification and the registration of tag signatures and descriptions. The founding members of this consortium include; Adobe Systems Inc., Agfa-Gevaert N.V., Apple Computer, Inc., Eastman Kodak Company, FOGRA (Honorary), Microsoft Corporation, Silicon Graphics, Inc., Sun Microsystems, Inc., and Taligent, Inc (resigned). These companies have committed to fully support this specification in their operating systems, platforms and applications.

0.4 Device Profiles

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

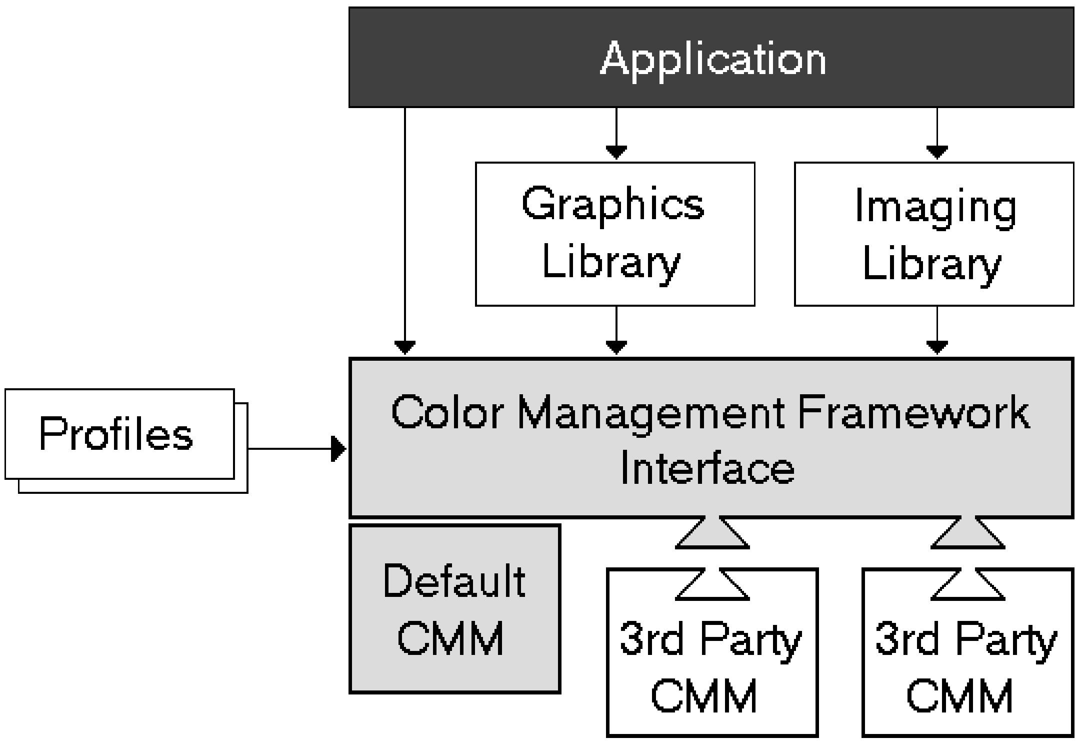


FIGURE 1.

0.5 Profile Element Structure

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

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

In addition to the required tags for each device profile, a number of optional tags are defined that can be used for enhanced color transformations. Examples of these tags include PostScript Level 2 support, calibration support, and others. In the case of required and optional tags, all of the signatures, an algorithmic description, and intent are registered with the International Color Consortium.

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

0.6 Embedded Profiles

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

0.7 Registration Authority

This standard requires that signatures for CMM type, device manufacturer, device model, profile tags and profile tag types be registered to insure that all profile data is uniquely defined. The registration authority for these data is the ICC Technical Secretary:

Michael Has
FOGRA
Streitfieldstrasse 19
D-81673 Munich, Germany
Tel: 49-89-4318241
Fax: 49-89-431-6896
Email: 100273.524@compuserve.com

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

0.8 Redundant Data Arbitration

There are several methods of color rendering described in the following structures that can function within a single CMM. If data for more than one method are included in the same profile, the following selection algorithm should be used by the software implementation: if an 8 bit or 16 bit lookup table is present, it should be used; if a lookup table is not present (and not required), the appropriate default modeling parameters are used.

1 Scope

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

2 Normative references

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

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

ISO 5/1:1984, "Photography (sensitometry) - Density measurements - Terms, symbols and notation"

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

ISO 5/4:1983, "Photography - Density measurements - Part 4: Geometric conditions for reflection density"

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

ISO 3664:1975, "Photography - Illumination conditions for viewing colour transparencies and their reproductions" *(Need to check against ANSI PH2.30-1989)*

ISO 12641:199X, "Graphic technology - Prepress digital data exchange - Colour targets for input scanner calibration" (Equals IT8.7/1 and IT8.7/2). Approved 1995, awaiting publication by ISO.

ISO 12642:199X, "Graphic technology - Prepress digital data exchange - Input data for characterization of 4-colour process printing" (Equals IT8.7/3). Approved 1995, awaiting publication by ISO.

ISO 13655:199X, "Graphic technology - Spectral measurement and colorimetric computation for images" (Equals CGATS.5). Approved 1995, awaiting publication by ISO.

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

TIFF 6.0 Specification, published by Adobe Systems Incorporated.

ISO/DIS 12639, "Graphic Technology - Prepress digital data exchange - Tag image file format for image technology (TIFF/IT)" (Incorporates ANSI IT8.8-1993 and adds additional compliance options. Final approval expected late 1996.)

ISO/WD 12234, "Photography - Electronic still picture cameras - Removeable memory (TIFF/EP)"

PICT Standard Specifications, published by Apple Computer, Inc.

3 Conformance

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

4 Definitions

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

4.1 aligned

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

4.2 ASCII string

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

4.3 big-endian

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

4.4 byte

An eight-bit unsigned binary integer.

4.5 byte offset

The number of bytes from the beginning of a field.

4.6 fixed point representation

A method of encoding a real number into binary by putting an implied binary point at a fixed bit position. See Table 2 for an example.

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

4.7 NULL

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

4.8 offset

An address within an ICC profile, relative to byte zero of the file.

4.9 perceptual intent

A rendering intent that specifies the full gamut of the image is compressed or expanded to fill the gamut of the destination device. Gray balance is preserved but colorimetric accuracy might not be preserved.

4.10 profile connection space (PCS)

An abstract color space used to connect the source and destination profiles. See Annex A.1: "Profile Connection Spaces" for a full description.

4.11 rendering intent

Rendering intent specifies the style of reproduction to be used during the evaluation of this profile in a sequence of profiles. It applies specifically to that profile in the sequence and not to the entire sequence. Typically, the user or application will set the rendering intent dynamically at runtime or embedding time.

4.12 saturation intent

A rendering intent that specifies the saturation of the pixels in the image is preserved perhaps at the expense of accuracy in hue and lightness.

5 Notation, symbols and abbreviations

5.1 Notations

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

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

5.2 Symbols and abbreviations

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

ANSI	American National Standards Institute
BCD	Binary Coded Decimal
BG	Black Generation
CIE	*Commission Internationale de l'Éclairage* (International Commission on Illumination)
CLUT	Color Lookup Table (multidimensional)
CMM	Color Management Module
CMS	Color Management System
CMY	Cyan, Magenta, Yellow
CMYK	Cyan, Magenta, Yellow, Key (black)
CRD	Color Rendering Dictionary
CRT	Cathode-Ray Tube
EPS	Encapsulated PostScript
ICC	International Color Consortium
ISO	International Standards Organization
LCD	Liquid Crystal Display
LUT	Lookup Table
PCS	Profile Connection Space
PPD	PostScript Printer Description
RGB	Red, Green, Blue

TIFF	Tagged Image File Format
TRC	Tone Reproduction Curve
UCR	Under Color Removal

5.3 Basic Numeric Types

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

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

TABLE 1.

5.3.2 s15Fixed16Number: This type represents a fixed signed 4 byte/32 bit quantity which has 16 fractional bits. An example of this encoding is:

-32768.0	80000000h
0	00000000h
1.0	00010000h
32767 + (65535/65536)	7FFFFFFFh

TABLE 2.

5.3.3 u16Fixed16Number: This type represents a fixed unsigned 4 byte/32 bit quantity which has 16 fractional bits. An example of this encoding is:

0	00000000h
1.0	00010000h
65535 + (65535/65536)	FFFFFFFFh

TABLE 3.

5.3.4 u8Fixed8Number: This type represents a fixed unsigned 2 byte/16 bit quantity which has 8 fractional bits. An example of this encoding is:

0	0000h
1.0	0100h
255 + (255/256)	FFFFh

TABLE 4.

5.3.5 uInt16Number: This type represents a generic unsigned 2 byte/16 bit quantity.

5.3.6 uInt32Number: This type represents a generic unsigned 4 byte/32 bit quantity.

5.3.7 uInt64Number: This type represents a generic unsigned 8 byte/64 bit quantity.

5.3.8 uInt8Number: This type represents a generic unsigned 1 byte/8 bit quantity.

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

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

TABLE 5.

For relative tristimulus values, the XYZNumbers are scaled such that a perfect reflecting diffuser has a Y value of 1.0 and NOT 100.0. Tristimulus values must be non-negative.

5.3.10 Seven Bit ASCII:

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

TABLE 6.

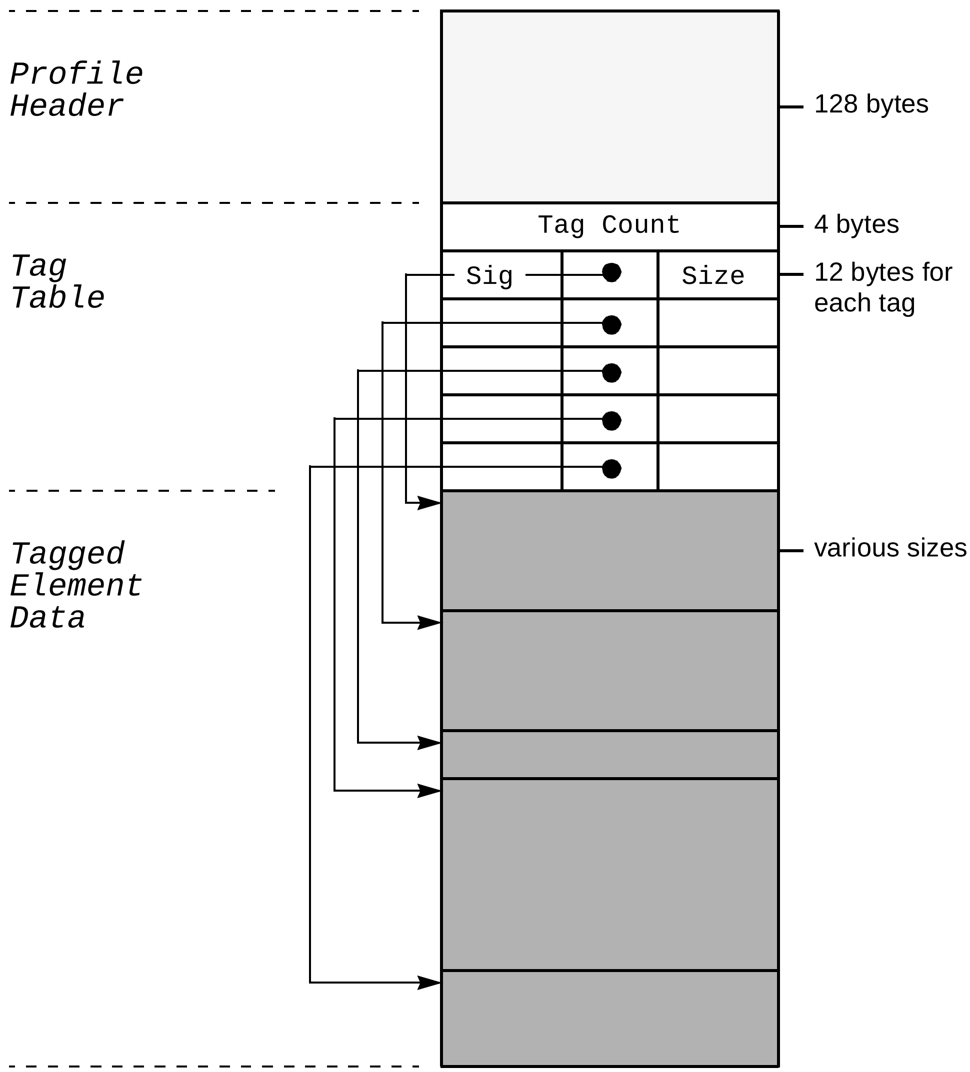
Decimal															
0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	“	35	#	36	$	37	%	38	&	39	‘
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47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

TABLE 7.

6 Requirements

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

FIGURE 2. **Profile Map**



First, the 128 byte file header as defined in clause 6.1: “Header Description”.

Second, the tag table as defined in clause 6.2: “Tag Table Definition”.

Third, the tag data elements in accordance with the requirements of clauses 6.3: “Required Tags for Profiles”, 6.4: “Tag Descriptions” and 6.5: “Tag Type Definitions”.

Note: The information necessary to understand and create the Tag Data Elements is arranged within this standard as follows. Each class, and subclass, of device (e.g.: input, RGB) requires the use of specific tags and allows other optional tags. These relationships are described in clause 6.3: “Required Tags for Profiles”. Tags themselves are described in clause 6.4: “Tag Descriptions”. However tag descriptions draw upon a series of commonly used “tag types” which are defined in clause 6.5: “Tag Type Definitions”. The definition of the basic number types used for data encoding are found in clause 5.3: “Basic Numeric Types”.

All profile data must be encoded as big-endian.

All color spaces used in this standard shall be in accordance with Annex A: “Color Spaces”.

6.1 Header Description

The profile header provides the necessary information to allow a receiving system to properly search and sort ICC profiles. Table 8 gives the byte position, content and encoding of the profile header.

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

Byte Offset	Content	Encoded as...
0-3	Profile size	uInt32Number
4-7	CMM Type	see below
8-11	Profile version number	see below
12-15	Profile/Device class	see below
16-19	Color space of data (possibly a derived space) [i.e. “the canonical input space”]	see below
20-23	Profile connection space [i.e. “the canonical output space”]	see below
24-35	Date and time this profile was first created	dateTimeNumber
36-39	’acsp’ (61637370h) profile file signature	
40-43	Primary platform target for the profile	see below

TABLE 8.

44-47	Flags to indicate various options for the CMM such as distributed processing and caching options	see below
48-51	Device manufacturer of the device for which this profile is created	see below
52-55	Device model of the device for which this profile is created	see below
56-63	Device attributes unique to the particular device setup such as media type	see below
64-67	Rendering Intent	see below
68-79	The XYZ values of the illuminant of the profile connection space. This must correspond to D50. It is explained in more detail in Annex A.1: "Profile Connection Spaces".	XYZNumber
80-83	Identifies the creator of the profile	see below
84-127	44 bytes reserved for future expansion	

TABLE 8.

6.1.1 Profile size

The total size of the profile in bytes.

6.1.2 CMM Type

Identifies the preferred CMM to be used. The signatures must be registered in order to avoid conflicts (see clause 0.7: "Registration Authority").

6.1.3 Profile Version

Profile version number where the first 8 bits are the major version number and the next 8 bits are for the minor version number. The major and minor version numbers are set by the International Color Consortium and will match up with the profile format revisions. The current version number is 02h with a minor version number of 10h.

The encoding is such that:

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

TABLE 9.

A major version change can only happen if there is an incompatible change. An example of a major version change may be the addition of new required tags. A minor version change can happen with compatible changes. An example of a minor version number change may be the addition of new optional tags.

6.1.4 Profile/Device class

There are three basic classifications (classes) of device profiles: input, display and output profiles.

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

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

TABLE 10.

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

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

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

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

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

These profiles have the following signatures:

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

TABLE 11.

6.1.5 Color Space Signature

The encoding is such that:

Color Space	**Signature**	**Hex Encoding**
XYZData	'XYZ '	58595A20h
labData	'Lab '	4C616220h
luvData	'Luv '	4C757620h
YCbCrData	'YCbr '	59436272h
YxyData	'Yxy '	59787920h
rgbData	'RGB '	52474220h
grayData	'GRAY'	47524159h
hsvData	'HSV '	48535620h
hlsData	'HLS '	484C5320h
cmykData	'CMYK'	434D594Bh
cmyData	'CMY '	434D5920h

TABLE 12.

2colorData	'2CLR'	32434C52h
3colorData (if not listed above)	'3CLR'	33434C52h
4colorData (if not listed above)	'4CLR'	34434C52h
5colorData	'5CLR'	35434C52h
6colorData	'6CLR'	36434C52h
7colorData	'7CLR'	37434C52h
8colorData	'8CLR'	38434C52h
9colorData	'9CLR'	39434C52h
10colorData	'ACLR'	41434C52h
11colorData	'BCLR'	42434C52h
12colorData	'CCLR'	43434C52h
13colorData	'DCLR'	44434C52h
14colorData	'ECLR'	45434C52h
15colorData	'FCLR'	46434C52h

TABLE 12.

6.1.6 Profile Connection Space Signature

The encoding is such that:

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

TABLE 13.

When the profile is a DeviceLink profile, the Profile Connection Space Signature is taken from the Color Space Signatures table. (See clause 6.1.5)

6.1.7 Primary Platform Signature

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

The encoding is such that:

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

TABLE 14.

6.1.8 Profile Flags

Flags to indicate various hints for the CMM such as distributed processing and caching options. The first 16 bits (low word in big-endian notation) are reserved for the Profile Consortium.

The encoding is such that:

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

TABLE 15.

6.1.9 Device manufacturer and model

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

6.1.10 Attributes

Attributes unique to the particular device setup such as media type. The first 32 bits (lower two words in big-endian notation) of this 64 bit value are reserved for the ICC.

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

Attribute	Bit Position
Reflective (off) or Transparency (on)	0
Glossy (off) or Matte (on)	1

TABLE 16.

6.1.11 Rendering Intent

Perceptual, relative colorimetric, saturation and absolute colorimetric are the four intents required to be supported. The first 16 bits (low word in big-endian notation) worth of numbers are reserved for the ICC.

The encoding is such that:

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

TABLE 17.

Note that this flag might not have any meaning until the profile is used in some context, e.g. in a DeviceLink or an embedded source profile.

6.1.12 Profile Creator

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

6.2 Tag Table Definition

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

Individual tag structures within the Tag Table:

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

TABLE 18.

6.2.1 Tag Signature

A four byte value registered with the ICC (see clause 0.7: “Registration Authority”).

6.2.2 Offset

See clause 4.8: “offset”.

6.2.3 Element Size

The number of bytes in the tag data element.

6.2.4 Tag Data Requirements

All tag data is required to start on a 4-byte boundary (relative to the start of the profile header) so that a tag starting with a 32 bit value will be properly aligned without the tag handler needing to know the contents of the tag. This means that the low 2 bits of the beginning offset must be 0. The element size should be for actual data and must not include padding at the end of the tag data. The header is the first element in the file structure encompassing the first 128 bytes. This is immediately followed by the tag table. Tag data elements make up the rest of the file structures. There may be any number of tags and no particular order is required for the data of the tags. Each tag may have any size (up to the limit imposed by the 32 bit offsets). Exactly which tags are required or optional with which profiles is described under sub-clause 6.3: “Required Tags for Profiles”.

6.3 Required Tags for Profiles

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

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

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

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

The interpretation of some tags are context dependent. This dependency is described below in Table 19 .

Profile	**Tag Name**	**Interpretation**
Input Profile	AToB0Tag	none
Display Profile	AToB0Tag	none
Output Profile	BToA0Tag	perceptual rendering
Output Profile	BToA1Tag	colorimetric rendering
Output Profile	BToA2Tag	saturation rendering
Input Profile	grayTRCTag	depends on intent
Display Profile	grayTRCTag	additive
Output Profile	grayTRCTag	subtractive

TABLE 19.

6.3.1 Input Profile

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

6.3.1.1 Monochrome Input Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve (TRC)
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 20.

The mathematical model implied by this data is:

$$connection = grayTRC[device]$$

This represents a simple tone reproduction curve adequate for most monochrome input devices. The *connection* values in this equation should represent the achromatic channel of the profile connection space. If the inverse of this is desired, then the following equation is used:

$$device = grayTRC^{-1}[connection]$$

Multidimensional tables are not allowed to be included in monochrome profiles.

6.3.1.2 Three-Component Matrix-Based Input Profiles

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

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
redColorantTag	Red colorant XYZ relative tristimulus values
greenColorantTag	Green colorant XYZ relative tristimulus values
blueColorantTag	Blue colorant XYZ relative tristimulus values
redTRCTag	Red channel tone reproduction curve
greenTRCTag	Green channel tone reproduction curve
blueTRCTag	Blue channel tone reproduction curve
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 21.

The forward mathematical model implied by this data is:

$$linear_r = redTRC[device_r]$$
$$linear_g = greenTRC[device_g]$$
$$linear_b = blueTRC[device_b]$$

$$\begin{bmatrix} connection_X \\ connection_Y \\ connection_Z \end{bmatrix} = \begin{bmatrix} redColorant_X & greenColorant_X & blueColorant_X \\ redColorant_Y & greenColorant_Y & blueColorant_Y \\ redColorant_Z & greenColorant_Z & blueColorant_Z \end{bmatrix} \begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix}$$

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

$$\begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix} = \begin{bmatrix} redColorant_X & greenColorant_X & blueColorant_X \\ redColorant_Y & greenColorant_Y & blueColorant_Y \\ redColorant_Z & greenColorant_Z & blueColorant_Z \end{bmatrix}^{-1} \begin{bmatrix} connection_X \\ connection_Y \\ connection_Z \end{bmatrix}$$

$$device_r = redTRC^{-1}[linear_r]$$
$$device_g = greenTRC^{-1}[linear_g]$$
$$device_b = blueTRC^{-1}[linear_b]$$

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

6.3.1.3 N-Component LUT-Based Input Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Device to PCS: 8 or 16 bit data
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 22.

The AToB0Tag represents a device model described by the lut8Type or lut16Types. This tag provides the parameter data for an algorithm that includes a set of non-interdependent per-channel tone reproduction curves, a multi-dimensional lookup table and a set of non-interdependent per-channel linearization curves. The mathematical model implied by this data is described in detail in clauses 6.5.5 and 6.5.6 that specify the general lookup table tag element structures.

6.3.2 Display Profile

This profile represents display devices such as monitors.

6.3.2.1 Monochrome Display Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 23.

The mathematical model implied by this data is:

$$connection = grayTRC[device]$$

This represents a simple tone reproduction curve adequate for most monochrome display devices. The *connection* values in this equation should represent the achromatic channel of the profile connection space. If the inverse of this is desired, then the following equation is used:

$$device = grayTRC^{-1}[connection]$$

Multidimensional tables are not allowed to be included in monochrome profiles.

6.3.2.2 RGB Display Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
redColorantTag	Relative XYZ values of red phosphor
greenColorantTag	Relative XYZ values of green phosphor
blueColorantTag	Relative XYZ values of blue phosphor
redTRCTag	Red channel tone reproduction curve
greenTRCTag	Green channel tone reproduction curve
blueTRCTag	Blue channel tone reproduction curve
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 24.

This model is based on a three non-interdependent per-channel tone reproduction curves to convert between linear and non-linear RGB values and a 3x3 matrix to convert between linear RGB values and relative XYZ values. The mathematical model implied by this data is:

$$linear_r = redTRC[device_r]$$
$$linear_g = greenTRC[device_g]$$
$$linear_b = blueTRC[device_b]$$

$$\begin{bmatrix} connection_X \\ connection_Y \\ connection_Z \end{bmatrix} = \begin{bmatrix} redColorant_X & greenColorant_X & blueColorant_X \\ redColorant_Y & greenColorant_Y & blueColorant_Y \\ redColorant_Z & greenColorant_Z & blueColorant_Z \end{bmatrix} \begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix}$$

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

$$\begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix} = \begin{bmatrix} redColorant_X & greenColorant_X & blueColorant_X \\ redColorant_Y & greenColorant_Y & blueColorant_Y \\ redColorant_Z & greenColorant_Z & blueColorant_Z \end{bmatrix}^{-1} \begin{bmatrix} connection_X \\ connection_Y \\ connection_Z \end{bmatrix}$$

$$device_r = redTRC^{-1}[linear_r]$$

$$device_g = greenTRC^{-1}[linear_g]$$

$$device_b = blueTRC^{-1}[linear_b]$$

Only the CIEXYZ encoding of the profile connection space can be used with matrix/TRC models. A multidimensional table tag must be included if the CIELAB encoding of the profile connection space is to be used.

6.3.3 Output Profile

This profile represents output devices such as printers and film recorders. The LUT tags that are required by the printer profiles contain either the 8 bit or the 16 bit LUTs as described in clauses 6.5.5: “lut16Type” and 6.5.6: “lut8Type”. The LUT algorithm for profile connection space to device space transformations process data sequentially through a matrix, input tables, a color LUT, and output tables.

6.3.3.1 Monochrome Output Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 25.

The mathematical model implied by this data is:

$$connection = grayTRC[device]$$

This represents a simple tone reproduction curve adequate for most monochrome output devices. The *connection* values in this equation should represent the achromatic channel of the profile connection space. If the inverse of this is desired, then the following equation is used:

$$device = grayTRC^{-1}[connection]$$

Multidimensional tables are not allowed to be included in monochrome profiles.

6.3.3.2 RGB and CMYK Output Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Device to PCS: 8 or 16 bit data: intent of 0
BToA0Tag	PCS to Device space: 8 or 16 bit data: intent of 0
gamutTag	Out of Gamut: 8 or 16 bit data
AToB1Tag	Device to PCS: 8 or 16 bit data: intent of 1
BToA1Tag	PCS to Device space: 8 or 16 bit data: intent of 1
AToB2Tag	Device to PCS: 8 or 16 bit data: intent of 2
BToA2Tag	PCS to Device space: 8 or 16 bit data: intent of 2
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 26.

These tags represent a device model described in clause 6.5.6: “lut8Type” or clause 6.5.5: “lut16Type”. The intent values described in these tags directly correlate to the value of the rendering intent header flag of the source profile in the color modeling session.

Rendering Intent	Value
perceptual	0
relative colorimetric	1
saturation	2
absolute colorimetric	3

TABLE 27.

Each of the first three intents are associated with a specific tag. The fourth intent, absolute colorimetry, is obtained by modifying the relative colorimetric intent tag based on the values which are in the mediaWhitePointTag. It is permissible to reference the same tag for all of these intents and to use the relative colorimetric intent tag when absolute colorimetry is specified. This decision is left to the profile builder.

In essence, each of these tags provides the parameter data for an algorithm that includes a 3x3 matrix, a set of non-interdependent per-channel tone reproduction curves, a multidimensional lookup table and a set of non-interdependent per-channel linearization curves. The algorithmic details of this model and the intent of each tag is given later in clauses 6.5.5: “lut16Type” or 6.5.6: “lut8Type” that specify the general lookup table tag element structures.

6.3.4 Additional Profile Formats

6.3.4.1 DeviceLink Profile

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

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Actual transformation parameter structure; 8 or 16 bit data
profileSequence-DescTag	An array of descriptions of the profile sequence
copyrightTag	7 bit ASCII profile copyright information

TABLE 28.

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

The AToB0Tag represents a device model described in clause 6.5.5: “lut16Type” or clause 6.5.6: “lut8Type”. This tag provides the parameter data for an algorithm that includes a 3x3 matrix, a set of non-interdependent per-channel tone reproduction curves, a multidimensional lookup table and a set of non-interdependent per-channel linearization curves. The algorithmic details of this model and the intent of each tag is given later in clauses 6.5.5: “lut16Type” and 6.5.6: “lut8Type” that specify the general lookup table tag element structures. This is a pre-evaluated transform that cannot be undone.

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

6.3.4.2 ColorSpace Conversion Profile

This profile provides the relevant information to perform a color space transformation between the non-device color spaces and the PCS. It does not represent any device model.

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
BToA0Tag	Inverse transformation parameter structure; 8 or 16 bit data
AToB0Tag	Actual transformation parameter structure; 8 or 16 bit data
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 29.

The AToB0Tag and BToA0Tag represent a model described in clause 6.5.5: “lut16Type” or clause 6.5.6: “lut8Type”. This tag provides the parameter data for an algorithm that includes a 3x3 matrix, a set of non-interdependent per-channel tone reproduction curves, a multidimensional lookup table and a set of non-interdependent per-channel linearization curves. The algorithmic details of this model and the intent of each tag is given later in clauses 6.5.5: “lut16Type” and 6.5.6: “lut8Type” that specify the general lookup table tag element structures.

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

6.3.4.3 Abstract Profile

This profile represents abstract transforms and does not represent any device model. Color transformations using abstract profiles are performed from PCS to

PCS.

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Actual transformation parameter structure; 8 or 16 bit data
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 30.

The AToB0Tag represents a PCS to PCS model described by the lut8Type or lut16Types. This tag provides the parameter data for an algorithm that includes a 3x3 matrix, a set of non-interdependent per-channel tone reproduction curves, a multidimensional lookup table and a set of non-interdependent per-channel linearization curves. The algorithmic details of this model and the intent of each tag is given later in clauses 6.5.5: “lut16Type” and 6.5.6: “lut8Type” that specify the general lookup table tag element structures.

6.3.4.4 Named Color Profile

This profile class is designated by setting the header’s “Profile/Device class” field to ‘nmcl’. Named color profiles can be thought of as sibling profiles to device profiles. For a given device there would be one or more device profiles to handle process color conversions and one or more named color profiles to handle named colors. There might be multiple named color profiles to account for different consumables or multiple named color vendors.

This profile provides a PCS and optional device representation for a list of named colors. Named color profiles are device-specific in that their data is shaped for a particular device.

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
namedColor2Tag	PCS and optional device representation for named colors
mediaWhitePointTag	Media XYZ white point
copyrightTag	7 bit ASCII profile copyright information

TABLE 31.

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

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

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

6.4 Tag Descriptions

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

In addition to the tags listed below, any of the previously defined tags in clause 6.3: "Required Tags for Profiles" can also be used as optional tags if they are not used in the required set for a particular profile and are not specifically excluded in a profile definition.

Tag Name	**General Description**
AToB0Tag	Multidimensional transformation structure
AToB1Tag	Multidimensional transformation structure
AToB2Tag	Multidimensional transformation structure
blueColorantTag	Relative XYZ values of blue phosphor or colorant
blueTRCTag	Blue channel tone reproduction curve
BToA0Tag	Multidimensional transformation structure
BToA1Tag	Multidimensional transformation structure
BToA2Tag	Multidimensional transformation structure
calibrationDateTimeTag	Profile calibration date and time
charTargetTag	Characterization target such as IT8/7.2
copyrightTag	7 bit ASCII profile copyright information
crdInfoTag	names of companion CRDs to the profile

TABLE 32.

deviceMfgDescTag	displayable description of device manufacturer
deviceModelDescTag	displayable description of device model
gamutTag	Out of Gamut: 8 or 16 bit data
grayTRCTag	Gray tone reproduction curve
greenColorantTag	Relative XYZ values of green phosphor or colorant
greenTRCTag	Green channel tone reproduction curve
luminanceTag	Absolute luminance for emissive device
measurementTag	Alternative measurement specification information
mediaBlackPointTag	Media XYZ black point
mediaWhitePointTag	Media XYZ white point
preview0Tag	Preview transformation: 8 or 16 bit data
preview1Tag	Preview transformation: 8 or 16 bit data
preview2Tag	Preview transformation: 8 or 16 bit data
profileDescriptionTag	profile description for display
profileSequence-DescTag	profile sequence description from source to destination
ps2CRD0Tag	PostScript Level 2 color rendering dictionary: perceptual
ps2CRD1Tag	PostScript Level 2 color rendering dictionary: colorimetric
ps2CRD2Tag	PostScript Level 2 color rendering dictionary: saturation
ps2CRD3Tag	PostScript Level 2 color rendering dictionary: absolute
ps2CSATag	PostScript Level 2 color space array
ps2RenderingIntentTag	PostScript Level 2 Rendering Intent
redColorantTag	Relative XYZ values of red phosphor or colorant
redTRCTag	Red channel tone reproduction curve
screeningDescTag	Screening attributes description
screeningTag	Screening attributes such as frequency, angle and spot
technologyTag	Device technology information such as LCD, CRT, Dye Sublimation, etc.
ucrbgTag	Under color removal and black generation description

TABLE 32.

viewingCondDescTag	Specifies viewing condition description
viewingConditionsTag	Specifies viewing condition parameters

TABLE 32.

6.4.1 AToB0Tag

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

Device to PCS: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: “lut16Type” and 6.5.6: “lut8Type”.

6.4.2 AToB1Tag

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

Device to PCS: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: “lut16Type” and 6.5.6: “lut8Type”.

6.4.3 AToB2Tag

Tag Type: lut8Type or lut16Type
Tag Signature: ‘A2B2’ (41324232h)

Device to PCS: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: “lut16Type” and 6.5.6: “lut8Type”.

6.4.4 blueColorantTag

Tag Type: XYZType
Tag Signature: ‘bXYZ’ (6258595Ah)

The relative XYZ values of blue phosphor or colorant.

6.4.5 blueTRCTag

Tag Type: curveType
Tag Signature: ‘bTRC’ (62545243h)

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

The count value specifies the number of entries in the curve table except as follows:

when count is 0, then a linear response (slope equal to 1.0) is assumed,

when count is 1, then the data entry is interpreted as a simple gamma value encoded as a u8Fixed8Number.

Gamma is interpreted canonically and not as an inverse.

6.4.6 BToA0Tag

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

PCS to Device space: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: "lut16Type" and 6.5.6: "lut8Type".

6.4.7 BToA1Tag

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

PCS to Device space: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: "lut16Type" and 6.5.6: "lut8Type".

6.4.8 BToA2Tag

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

PCS to Device space: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: "lut16Type" and 6.5.6: "lut8Type".

6.4.9 calibrationDateTimeTag

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

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

6.4.10 charTargetTag

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

This tag contains the measurement data for a characterization target such as IT8.7/2. This tag is provided so that distributed utilities can create transforms "on the fly" or check the current performance against the original device performance. The tag embeds the exact data file format defined in the ANSI or ISO standard which is applicable to the device being characterized.

Examples are the data formats described in ANSI IT8.7/1-1993 section 4.10, ANSI IT8.7/2-1993 section 4.10 and ANSI IT8.7/3 section 4.10. Each of these file formats contains an identifying character string as the first few bytes of the format, allowing an external parser to determine which data file format is being used. This provides the facilities to include a wide range of targets using a variety of measurement specifications in a standard manner.

> Note: The IT8 specifications do not currently have a keyword which identifies the set as being reference data as opposed to device response data. An addition to enable this additional data set is being considered by the IT8 committee.

6.4.11 copyrightTag

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

This tag contains the 7 bit ASCII text copyright information for the profile.

6.4.12 crdInfoTag

Tag Type: crdInfoType
Tag Signature: 'crdi' (63726469h)

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.

See Annex D.1 "Synchronizing Profiles and CRDs" for information about using this tag.

6.4.13 deviceMfgDescTag

Tag Type: textDescriptionType

Tag Signature: ‘dmnd’ (646D6E64h)

Structure containing invariant and localizable versions of the device manufacturer for display. The content of this structure is described in clause 6.5.14: “textDescriptionType”.

6.4.14 deviceModelDescTag

Tag Type: textDescriptionType
Tag Signature: ‘dmdd’ (646D6464h)

Structure containing invariant and localizable versions of the device model for display. The content of this structure is described in clause 6.5.14: “textDescriptionType”.

6.4.15 gamutTag

Tag Type: lut8Type or lut16Type
Tag Signature: ‘gamt’ (67616D74h)

Out of Gamut tag: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: “lut16Type” and 6.5.6: “lut8Type”.

The CLUT tag has a single output. If the output value is 0, the input color is in gamut. If the output is non-zero, the input color is out of gamut, with the number “n+1” being at least as far out of the gamut as the number “n”.

6.4.16 grayTRCTag

Tag Type: curveType
Tag Signature: ‘kTRC’ (6B545243h)

Gray tone reproduction curve. The tone reproduction curve provides the necessary information to convert between a single device channel and the CIEXYZ encoding of the profile connection space. The first element represents no colorant (white) or phosphors (black) and the last element represents 100 percent colorant (black) or 100 percent phosphor (white).

The count value specifies the number of entries in the curve table except as follows:

> when count is 0, then a linear response (slope equal to 1.0) is assumed,
>
> when count is 1, then the data entry is interpreted as a simple gamma value encoded as a u8Fixed8Number.

Gamma is interpreted canonically and not as an inverse.

6.4.17 greenColorantTag

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

Relative XYZ values of green phosphor or colorant.

6.4.18 greenTRCTag

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

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

The count value specifies the number of entries in the curve table except as follows:

> when count is 0, then a linear response (slope equal to 1.0) is assumed,
>
> when count is 1, then the data entry is interpreted as a simple gamma value encoded as a u8Fixed8Number.

Gamma is interpreted canonically and not as an inverse.

6.4.19 luminanceTag

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

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.

6.4.20 measurementTag

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

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

6.4.21 mediaBlackPointTag

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

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 this tag is not present, it is assumed to be (0,0,0).

6.4.22 mediaWhitePointTag

Tag Type: XYZType
Tag Signature: 'wtpt' (77747074h)

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.

6.4.23 namedColorTag

Tag Type: namedColorType
Tag Signature: 'ncol' (6E636F6Ch)

NOTE: This tag is obsolete, and should not be used in new profiles. Use namedColor2Tag instead.

Named color reference transformation for converting between named color sets and the profile connection space or device color spaces.

6.4.24 namedColor2Tag

Tag Type: namedColor2Type
Tag Signature: 'ncl2' (6E636C32h)

Named color information providing a PCS and optional device representation for a list of named colors. The namedColorTag should no longer be used.

6.4.25 preview0Tag

Tag Type: lut8Type or lut16Type
Tag Signature: 'pre0' (70726530h)

Preview transformation from PCS to device space and back to the PCS: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5:

"lut16Type" and 6.5.6: "lut8Type".

6.4.26 preview1Tag

Tag Type: lut8Type or lut16Type
Tag Signature: 'pre1' (70726531h)

Preview transformation from the PCS to device space and back to the PCS: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: "lut16Type" and 6.5.6: "lut8Type".

6.4.27 preview2Tag

Tag Type: lut8Type or lut16Type
Tag Signature: 'pre2' (70726532h)

Preview transformation from PCS to device space and back to the PCS: 8 bit or 16 bit data. The processing mechanisms are described in clauses 6.5.5: "lut16Type" and 6.5.6: "lut8Type".

6.4.28 profileDescriptionTag

Tag Type: textDescriptionType
Tag Signature: 'desc' (64657363h)

Structure containing invariant and localizable versions of the profile description for display. The content of this structure is described in clause 6.5.14: "textDescriptionType". This invariant description has no fixed relationship to the actual profile disk file name.

6.4.29 profileSequenceDescTag

Tag Type: profileSequenceDescType
Tag Signature: 'pseq' (70736571h)

Structure containing a description of the profile sequence from source to destination, typically used with the DeviceLink profile. The content of this structure is described in clause 6.5.10: "profileSequenceDescType".

6.4.30 ps2CRD0Tag

Tag Type: dataType
Tag Signature: 'psd0' (70736430h)

PostScript Level 2 Type 1 color rendering dictionary (CRD) for the Perceptual rendering intent. This tag provides the dictionary operand to the setcolorrendering operator. This tag can be used in conjunction with the setcolorrendering operator on any PostScript Level 2 device.

See Annex D: “PostScript Level 2 Tags”, for the relationship between the ICC profile data and PostScript Tags.

6.4.31 ps2CRD1Tag

Tag Type: dataType
Tag Signature: ‘psd1’ (70736431h)

PostScript Level 2 Type 1 CRD for the RelativeColorimetric rendering intent. This tag provides the dictionary operand to the setcolorrendering operator. This tag can be used in conjunction with the setcolorrendering operator on any PostScript Level 2 device.

See Annex D: “PostScript Level 2 Tags”, for the relationship between the ICC profile data and PostScript Tags.

6.4.32 ps2CRD2Tag

Tag Type: dataType
Tag Signature: ‘psd2’ (70736432h)

PostScript Level 2 Type 1 CRD for the Saturation rendering intent. This tag provides the dictionary operand to the setcolorrendering operator. This tag can be used in conjunction with the setcolorrendering operator on any PostScript Level 2 device.

See Annex D:: “PostScript Level 2 Tags”, for the relationship between the ICC profile data and PostScript Tags.

6.4.33 ps2CRD3Tag

Tag Type: dataType
Tag Signature: ‘psd3’ (70736433h)

PostScript Level 2 Type 1 CRD for the AbsoluteColorimetric rendering intent. This tag provides the dictionary operand to the setcolorrendering operator. This tag can be used in conjunction with the setcolorrendering operator on any PostScript Level 2 device.

See Annex D: “PostScript Level 2 Tags”, for the relationship between the ICC profile data and PostScript Tags.

6.4.34 ps2CSATag

Tag Type: dataType
Tag Signature: ‘ps2s’ (70733273h)

PostScript Level 2 color space array. This tag provides the array operand to the setcolorspace operator. For color spaces that fit within the original PostScript Level 2 device independent color model no operator verification need be performed. For color spaces that fit only within extensions to this model, operator verification is first required. An example of this would be for Calibrated CMYK input color spaces which are supported via an extension. In such cases where the necessary PostScript Level 2 support is not available, PostScript Level 1 color spaces, such as DeviceCMYK, can be used, or the colors can be converted on the host using a CMS. In the latter case, the PostScript Level 1 color operators are used to specify the device dependent (pre-converted) colors. The PostScript contained in this tag expects the associated color values instantiated either through setcolor or image to be in the range [0, 1].

See Annex D: “PostScript Level 2 Tags”, for the relationship between the ICC profile data and PostScript Tags.

6.4.35 ps2RenderingIntentTag

Tag Type: dataType
Tag Signature: ‘ps2i’ (70733269h)

PostScript Level 2 rendering intent. This tag provides the operand to the findcolorrendering operator. findcolorrendering is not necessarily supported on all PostScript Level 2 devices, hence its existence must first be established. Standard values for ps2RenderingIntentTag are RelativeColorimetric, AbsoluteColorimetric, Perceptual, and Saturation. These intents are meant to correspond to the rendering intents of the profile's header.

See Annex D: “PostScript Level 2 Tags”, for the relationship between the ICC profile data and PostScript Tags.

6.4.36 redColorantTag

Tag Type: XYZType
Tag Signature: ‘rXYZ’ (7258595Ah)

Relative XYZ values of red phosphor or colorant.

6.4.37 redTRCTag

Tag Type: curveType
Tag Signature: ‘rTRC’ (72545243h)

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

The count value specifies the number of entries in the curve table except as follows:

> when count is 0, then a linear response (slope equal to 1.0) is assumed,
>
> when count is 1, then the data entry is interpreted as a simple gamma value encoded as a u8Fixed8Number.

Gamma is interpreted canonically and not as an inverse.

6.4.38 screeningDescTag

Tag Type: textDescriptionType
Tag Signature: ‘scrd’ (73637264h)

Structure containing invariant and localizable versions of the screening conditions. The content of this structure is described in clause 6.5.14: “textDescriptionType”.

6.4.39 screeningTag

Tag Type: screeningType
Tag Signature: ‘scrn’ (7363726Eh)

This tag contains screening information for a variable number of channels.

6.4.40 technologyTag

Tag Type: signatureType
Tag Signature: ‘tech’ (74656368h)

Device technology information such as CRT, Dye Sublimation, etc. The encoding is such that:

Technology	Signature	Hex Encoding
Film Scanner	‘fscn’	6673636Eh
Digital Camera	‘dcam’	6463616Dh
Reflective Scanner	‘rscn’	7273636Eh
Ink Jet Printer	‘ijet’	696A6574h
Thermal Wax Printer	‘twax’	74776178h
Electrophotographic Printer	‘epho’	6570686Fh
Electrostatic Printer	‘esta’	65737461h
Dye Sublimation Printer	‘dsub’	64737562h
Photographic Paper Printer	‘rpho’	7270686Fh
Film Writer	‘fprn’	6670726Eh
Video Monitor	‘vidm’	7669646Dh
Video Camera	‘vidc’	76696463h
Projection Television	‘pjtv’	706A7476h
Cathode Ray Tube Display	‘CRT ‘	43525420h
Passive Matrix Display	‘PMD ‘	504D4420h
Active Matrix Display	‘AMD ‘	414D4420h
Photo CD	‘KPCD’	4B504344h
PhotoImageSetter	‘imgs’	696D6773h
Gravure	‘grav’	67726176h
Offset Lithography	‘offs’	6F666673h
Silkscreen	‘silk’	73696C6Bh
Flexography	‘flex’	666C6578h

TABLE 33.

6.4.41 ucrbgTag

Tag Type: ucrbgType
Tag Signature: ‘bfd ’ (62666420h)

Under color removal and black generation specification. This tag contains curve information for both under color removal and black generation in addition to a general description. The content of this structure is described in clause 6.5.17: “ucrbgType”.

This tag provides descriptive information only and is not involved in the processing model.

6.4.42 viewingCondDescTag

Tag Type: textDescriptionType
Tag Signature: ‘vued’ (76756564h)

Structure containing invariant and localizable versions of the viewing conditions. The content of this structure is described in clause 6.5.14: “textDescriptionType”.

6.4.43 viewingConditionsTag

Tag Type: viewingConditionsType
Tag Signature: ‘view’ (76696577h)

Viewing conditions parameters. The content of this structure is described in clause 6.5.22: “viewingConditionsType”.

6.5 Tag Type Definitions

This clause specifies the type and structure definitions used to create all of the individual tagged elements in the ICC Profile Format. The data type description identifiers are indicated at the right margin of each data or structure definition. An effort was made to make sure one-byte, two-byte and four-byte data lies on one-byte, two-byte and four-byte boundaries respectively. This required occasionally including extra spaces indicated with “reserved for padding” in some tag type definitions. Value 0 is defined to be of “unknown value” for all enumerated data structures.

All tags, including private tags, have as their first four bytes (0-3) a tag signature (a 4 byte sequence) to identify to profile readers what kind of data is contained within a tag. This encourages tag type reuse and allows profile parsers to reuse code when tags use common tag types. The second four bytes (4-7) are reserved for future expansion and must be set to 0 in this version of the specification. Each new tag signature and tag type signature must be registered with the International Color Consortium (see clause 0.7: “Registration Authority”) in order to prevent signature collisions.

Where not specified otherwise, the low 16 bits of all 32 bit flags in the type descriptions below are reserved for use by the International Color Consortium.

When 7 bit ASCII text representation is specified in types below, each individual character is encoded in 8 bits with the high bit set to zero. The details are presented in clause 5.3.10: “Seven Bit ASCII”.

6.5.1 crdInfoType

This type 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.

Byte Offset	Content	Encoded as...
0-3	‘crdi‘(63726469h) type descriptor	
4-7	reserved, must be set to 0	
8-11	PostScript product name character count, including terminating null	uInt32Number
12 - m-1	PostScript product name string in 7 bit ASCII	
m- m+3	Rendering intent 0 CRD character count, including terminating null	uInt32Number
m+4 - n-1	Rendering intent 0 CRD name string in 7 bit ASCII	
n - n+3	Rendering intent 1 CRD character count, including terminating null	uInt32Number
n+4 - p-1	Rendering intent 1 CRD name string in 7 bit ASCII	
p -p+3	Rendering intent 2 CRD character count, including terminating null	uInt32Number
p+4 - q-1	Rendering intent 2 CRD name string in 7 bit ASCII	
q -q+3	Rendering intent 3 CRD character count, including terminating null	uInt32Number
q+4 - r	Rendering intent 3 CRD name string in 7 bit ASCII	

TABLE 34.

If a companion CRD is not available for a given profile, then the character count field is zero and there is no string.

See Annex D.1 “Synchronizing Profiles and CRDs” for more information.

6.5.2 curveType

The curveType contains a 4 byte count value and a one-dimensional table of 2 byte values. The byte stream is given below.

Byte Offset	Content	Encoded as...
0-3	‘curv‘(63757276h) type descriptor	
4-7	reserved, must be set to 0	
8-11	count value specifying number of entries that follow	uInt32Number
12-end	actual curve values starting with the zeroth entry and ending with the entry count-1.	uInt16Number[]

TABLE 35.

Unless otherwise specified (see clauses 6.4.5: “blueTRCTag”, 6.4.16: “grayTRCTag”, 6.4.18: “greenTRCTag”, and 6.4.37: “redTRCTag”) these 16 bit unsigned integers in the range 0 to 65535 linearly map to curve values in the interval [0.0, 1.0].

6.5.3 dataType

The dataType is a simple data structure that contains either 7 bit ASCII or binary data, i.e. textType data or transparent 8-bit bytes. The length of the string is obtained by subtracting 12 from the element size portion of the tag itself. If this type is used for ASCII data, it must be terminated with a 00h byte.

Byte Offset	Content
0-3	‘data‘(64617461h) type descriptor
4-7	reserved, must be set to 0
8-11	data flag, 00000000h represents ASCII data, 00000001h represents binary data, other values are reserved for future use
12-n	a string of (element size - 12) ASCII characters or (element size - 12) bytes

TABLE 36.

6.5.4 dateTimeType

This dateTimeType is a 12 byte value representation of the time and date. The actual values are encoded as a dateTimeNumber described in clause 5.3.1.

Byte Offset	Content	Encoded as...
0-3	‘dtim‘(6474696Dh) type descriptor	
4-7	reserved, must be set to 0	
8-19	date and time	dateTimeNumber

TABLE 37.

6.5.5 lut16Type

This structure converts an input color into an output color using tables with 16 bit precision. This type contains four processing elements: a 3 by 3 matrix (only used when the input color space is XYZ), a set of one dimensional input lookup tables, a multidimensional lookup table, and a set of one dimensional output tables. Data is processed using these elements via the following sequence:

(matrix) -> (1d input tables) -> (multidimensional lookup table) -> (1d output tables).

Byte Offset	Content	Encoded as...
0-3	‘mft2‘(6D667432h) [multi-function table with 2 byte precision] type descriptor	
4-7	reserved, must be set to 0	
8	Number of Input Channels	uInt8Number
9	Number of Output Channels	uInt8Number
10	Number of CLUT grid points (identical for each side)	uInt8Number
11	Reserved for padding (required to be 00h)	
12-15	Encoded e00 parameter	s15Fixed16Number
16-19	Encoded e01 parameter	s15Fixed16Number
20-23	Encoded e02 parameter	s15Fixed16Number
24-27	Encoded e10 parameter	s15Fixed16Number
28-31	Encoded e11 parameter	s15Fixed16Number
32-35	Encoded e12 parameter	s15Fixed16Number
36-39	Encoded e20 parameter	s15Fixed16Number
40-43	Encoded e21 parameter	s15Fixed16Number
44-47	Encoded e22 parameter	s15Fixed16Number
48-49	Number of input table entries	uInt16Number
50-51	Number of output table entries	uInt16Number
52-n	input tables	uInt16Number[]
n+1-m	CLUT values	uInt16Number[]
m+1-o	output tables	uInt16Number[]

TABLE 38.

The matrix is organized as a 3 by 3 array. The dimension corresponding to the matrix rows varies least rapidly and the dimension corresponding to the matrix columns varies most rapidly and is shown in matrix form below.

$$\begin{bmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{bmatrix}$$

When using the matrix of an output profile, and the input data is XYZ, we have

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Each input X, Y or Z is an unsigned 1.15 number and each matrix entry is a signed 15.16 number. Therefore, each multiplication in the matrix multiply is 1.15 * s15.16 = s16.31 and the final sum is also s16.31 (48 bits). From this sum we take bits 31-16 as the unsigned integer result for X′, Y′, or Z′. These are then used as the inputs to the input tables of the multidimensional LUT. This normalization is used since the number of fractional bits in the input data must be maintained by the matrix operation.

The matrix is mandated to be an identity matrix unless the input is in the XYZ color space.

The input tables are arrays of 16 bit unsigned values. Each input table consists of a minimum of two and a maximum of 4096 two byte integers. Each input table entry is appropriately normalized to the range 0-65535. The inputTable is of size (InputChannels * inputTableEntries * 2) bytes. When stored in this tag, the one-dimensional lookup tables are assumed to be packed one after another in the order described below.

The CLUT is organized as an n-dimensional array with a given number of grid points in each dimension, where n is the number of input channels (input tables) in the transform. The dimension corresponding to the first input channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value contains m two-byte integers, where m is the number of output channels. The first sequential two byte integer of the entry contains the function value for the first output function, the second sequential two byte integer of the entry contains the function value for the second output function, and so on until all the output functions have been supplied. Each two byte integer in the CLUT is appropriately normalized to the range of 0 - 65535. The equation for computing the byte size of the CLUT is:

$$CLUTSize = GridPoints^{InputChannels} \bullet OutputChannels \bullet 2$$

The output tables are arrays of 16 bit unsigned values. Each output table consists of a minimum of two and a maximum of 4096 two byte integers. Each output table entry is appropriately normalized to the range 0 - 65535. The outputTable is of size (OutputChannels * outputTableEntries * 2) bytes. When stored in this tag, the one-dimensional lookup tables are assumed to be packed one after another in the order described in the following paragraph.

When using this type, it is necessary to assign each color space component to an input and output channel. The following table shows these assignments. The channels are numbered according to the order in which their table occurs. Note that additional color spaces can be added simply by defining the signature, channel assignments, and creating the tables.

Color Space	Channel 1	Channel 2	Channel 3	Channel 4
'XYZ'	X	Y	Z	
'Lab'	L	a	b	
'Luv'	L	u	v	
'YCbr'	Y	Cb	Cr	
'Yxy'	Y	x	y	
'RGB'	R	G	B	
'GRAY'	K			
'HSV'	H	S	V	
'HLS'	H	L	S	
'CMYK'	C	M	Y	K
'CMY'	C	M	Y	
'2CLR'	Ch. 1	Ch. 2		
'3CLR'	Ch. 1	Ch. 2	Ch. 3	
'4CLR'	Ch. 1	Ch. 2	Ch. 3	Ch. 4

TABLE 39.

6.5.6 lut8Type

This structure converts an input color into an output color using tables of 8 bit precision. This type contains four processing elements: a 3 by 3 matrix (only used when the input color space is XYZ), a set of one dimensional input lookup tables, a multidimensional lookup table, and a set of one dimensional output tables. Data is processed using these elements via the following sequence:

(matrix) -> (1d input tables) -> (multidimensional lookup table) -> (1d output tables).

Byte Offset	Content	Encoded as...
0-3	‘mft1‘(6D667431h) [multi-function table with 1 byte precision] type descriptor	
4-7	reserved, must be set to 0	
8	Number of Input Channels	uInt8Number
9	Number of Output Channels	uInt8Number
10	Number of CLUT grid points (identical for each side)	uInt8Number
11	Reserved for padding (fill with 00h)	
12-15	Encoded e00 parameter	s15Fixed16Number
16-19	Encoded e01 parameter	s15Fixed16Number
20-23	Encoded e02 parameter	s15Fixed16Number
24-27	Encoded e10 parameter	s15Fixed16Number
28-31	Encoded e11 parameter	s15Fixed16Number
32-35	Encoded e12 parameter	s15Fixed16Number
36-39	Encoded e20 parameter	s15Fixed16Number
40-43	Encoded e21 parameter	s15Fixed16Number
44-47	Encoded e22 parameter	s15Fixed16Number
48-m	input tables	uInt8Number[]
m+1-n	CLUT values	uInt8Number[]
n+1-o	output tables	uInt8Number[]

TABLE 40.

The matrix is organized as a 3 by 3 array. The dimension corresponding to the matrix rows varies least rapidly and the dimension corresponding to the matrix columns varies most rapidly and is shown in matrix form below.

$$\begin{bmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{bmatrix}$$

When using the matrix of an output profile, and the input data is XYZ, we have

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Each input X, Y or Z is an unsigned 1.15 number and each matrix entry is a signed 15.16 number. Therefore, each multiplication in the matrix multiply is 1.15 * s15.16 = s16.31 and the final sum is also s16.31 (48 bits). From this sum we take bits 31-16 as the unsigned integer result for X′, Y′, or Z′. These are then scaled to the range 0-255 and used as the inputs to the input tables of the multidimensional LUT. This normalization is used since the number of fractional bits in the input data must be maintained by the matrix operation.

The matrix is mandated to be an identity matrix unless the input is in the XYZ color space.

The input tables are arrays of 8 bit unsigned values. Each input table consists of 256 one byte integers. Each input table entry is appropriately normalized to the range 0-255. The inputTable is of size (InputChannels * 256) bytes. When stored in this tag, the one-dimensional lookup tables are assumed to be packed one after another in the order described below.

The CLUT is organized as an n-dimensional array with a given number of grid points in each dimension, where n is the number of input channels (input tables) in the transform. The dimension corresponding to the first input channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is an m-byte array, where m is the number of output channels. The first sequential byte of the entry contains the function value for the first output function, the second sequential byte of the entry contains the function value for the second output function, and so on until all the output functions have been supplied. Each byte in the CLUT is appropriately normalized to the range 0 - 255. The equation for computing the byte size of the CLUT is:

$$CLUTSize = GridPoints^{InputChannels} \bullet OutputChannels$$

The output tables are arrays of 8 bit unsigned values. Each output table consists of 256 one byte integers. Each output table entry is appropriately normalized to the range 0 - 255. The outputTable is of size (OutputChannels * 256) bytes. When stored in this tag, the one-dimensional lookup tables are assumed to be packed one after another in the order described in the following paragraph.

When using this type, it is necessary to assign each color space component to an input and output channel. The following table shows these assignments. The channels are numbered according to the order in which their table occurs. Note that additional color spaces can be added simply by defining the signature, channel assignments, and creating the tables.

Color Space	Channel 1	Channel 2	Channel 3	Channel 4
‘XYZ’	X	Y	Z	
‘Lab’	L	a	b	
‘Luv’	L	u	v	
‘Yxy’	Y	x	y	
‘YCbr’	Y	Cb	Cr	
‘RGB’	R	G	B	
‘GRAY’	K			
‘HSV’	H	S	V	
‘HLS’	H	L	S	
‘CMYK’	C	M	Y	K
‘CMY’	C	M	Y	
‘2CLR’	Ch. 1	Ch. 2		
‘3CLR’	Ch. 1	Ch. 2	Ch. 3	
‘4CLR’	Ch. 1	Ch. 2	Ch. 3	Ch. 4

TABLE 41.

6.5.7 measurementType

The measurementType information refers only to the internal profile data and is meant to provide profile makers an alternative to the default measurement specifications.

Byte Offset	Content	Encoded as...
0-3	‘meas‘(6D656173h) type descriptor	
4-7	reserved, must be set to 0	
8-11	encoded value for standard observer	see below
12-23	XYZ tristimulus values for measurement backing	XYZNumber
24-27	encoded value for measurement geometry	see below
28-31	encoded value for measurement flare	see below
32-35	encoded value for standard illuminant	see below

TABLE 42.

The encoding for the standard observer field is such that:

Standard Observer	Encoded Value
unknown	00000000h
1931 2 degree Observer	00000001h
1964 10 degree Observer	00000002h

TABLE 43.

The encoding for the measurement geometry field is such that:

Geometry	Encoded Value
unknown	00000000h
0/45 or 45/0	00000001h
0/d or d/0	00000002h

TABLE 44.

The encoding for the measurement flare value is shown below and is equivalent to the basic numeric type u16Fixed16Number in clause 5.3.3.

Tristimulus Value	Encoded Value
0 (0%)	00000000h
1.0 (or 100%)	00010000h

TABLE 45.

The encoding for the standard illuminant field is such that:

Standard Illuminant	Encoded Value
unknown	00000000h
D50	00000001h
D65	00000002h
D93	00000003h
F2	00000004h
D55	00000005h
A	00000006h
Equi-Power (E)	00000007h
F8	00000008h

TABLE 46.

6.5.8 namedColorType

NOTE: This type is obsolete, and should not be used in new profiles. Use namedColor2Type instead.

The namedColorType is a count value and array of structures that provide color coordinates for 7 bit ASCII color names. This provides users the ability to create a logo color dictionary between a named color set and a space color specification. The color space is identified by the “color space of data” field of the profile header. In order to maintain maximum portability it is strongly recommended that special characters of the 7 bit ASCII set not be used.

Byte Offset	Content	Encoded as...
0-3	‘ncol‘(6E636F6Ch) type descriptor	
4-7	reserved, must be set to 0	
8-11	vendor specific flag (lower 16 bits reserved for Consortium use)	
12-15	count of named colors	uInt32Number
15-t	prefix for each color name (maximum of 32 bytes) 7 bit ASCII, 0 terminated	
t+1-u	suffix for each color name (maximum of 32 bytes) 7 bit ASCII, 0 terminated	
u+1-v	first color root name (maximum of 32 bytes) 7 bit ASCII, 0 terminated	
v+1-w	first name's color coordinates. Color space of data	
w+1-x	second color root name (maximum of 32 bytes) 7 bit ASCII, 0 terminated	
x+1-y	second name's color coordinates. Color space of data	
y+1-z	the remaining count-2 name structures as described in the first two name structures (assuming count > 2)	

TABLE 47.

6.5.9 namedColor2Type

The namedColor2Type is a count value and array of structures that provide color coordinates for 7 bit ASCII color names. For each named color, a PCS and optional device representation of the color are given. Both representations are 16 bit values. The device representation corresponds to the header's "color space of data" field. This representation should be consistent with the "number of device components" field in the namedColor2Type. If this field is 0, device coordinates are not provided. The PCS representation corresponds to the header's PCS field. The PCS representation is always provided. Color names are fixed-length, 32 byte fields including null termination. In order to maintain maximum portability, it is strongly recommended that special characters of the 7 bit ASCII set not be used.

The namedColorType should no longer be used.

Byte Offset	Content	Encoded as...
0-3	‘ncl2‘(6E636C32h) type descriptor	
4-7	reserved, must be set to 0	
8-11	vendor specific flag (lower 16 bits reserved for Consortium use)	
12-15	count of named colors	uInt32Number
16-19	number of device coordinates for each named color	uInt32Number
20-51	prefix for each color name (32 byte field including null termination) 7 bit ASCII	
52-83	suffix for each color name (32 byte field including null termination) 7 bit ASCII	
84-115	first color root name (32 byte field including null termination) 7 bit ASCII	
116-121	first named color's PCS coordinates. Each coordinate is a uInt16Number. The encoding is the same as the encodings for the PCS color spaces as described in Annex A: “Color Spaces”. Only 16 bit L*a*b* and XYZ are allowed. The number of coordinates is consistent with the header's PCS.	
122-y	first named color's device coordinates. Each coordinate is a uInt16Number where 0000h represents the minimum value for the device coordinate and FFFFh represents the maximum value for the device coordinate. The number of coordinates is given by the "number of device coordinates" field. If the "number of device coordinates" field is 0, this field is not given.	
y+1-z	if count > 1 the remaining count-1 colors are described in a manner consistent with the first named color.	

TABLE 48.

6.5.10 profileSequenceDescType

This type is an array of structures, each of which contains information from the header fields and tags from the original profiles which were combined to create the final profile. The order of the structures is the order in which the profiles were combined and includes a structure for the final profile. This provides a description of the profile sequence from source to destination, typically used with the DeviceLink profile.

Byte Offset	Content
0-3	‘pseq’(70736571h) type descriptor
4-7	reserved, must be set to 0
8-11	count value specifying number of description structures in the array
12-m	‘count’ profile description structures

TABLE 49.

Each profile description structure has the format:

Byte Offset	Content
0-3	Device manufacturer signature (from corresponding profile’s header)
4-7	Device model signature (from corresponding profile’s header)
8-15	Device attributes (from corresponding profile’s header)
16-19	Device technology information such as CRT, Dye Sublimation, etc. (corresponding profile’s technology signature)
20-m	displayable description of device manufacturer (corresponding profile’s deviceMfgDescTag)
m+1- n	displayable description of device model (corresponding profile’s deviceModelDescTag)

TABLE 50.

If the deviceMfgDescTag and/or deviceModelDescTag is not present in a component profile, then a “placeholder” tag should be inserted. This tag should have a 1 in the ASCII count field and a terminating null in the ASCII invariant profile description and zeros in the UniCode and ScriptCode count and code fields.

Also note that the entire tag, including the tag type, should be stored.

If the technologyTag is not present, bytes 16-19 should be 00000000h.

6.5.11 s15Fixed16ArrayType

This type represents an array of generic 4 byte/32 bit fixed point quantity. The number of values is determined from the size of the tag.

Byte Offset	Content
0-3	'sf32'(73663332h) type descriptor
4-7	reserved, must be set to 0
8-n	an array of s15Fixed16Number values

TABLE 51.

6.5.12 screeningType

The screeningType describes various screening parameters including screen frequency, screening angle, and spot shape.

Byte Offset	Content	Encoded as...
0-3	'scrn'(7363726Eh) type descriptor	
4-7	reserved, must be set to 0	
8-11	screening flag	see below
12-15	number of channels	
16-19	channel #1 frequency	s15Fixed16Number
20-23	channel #1 screen angle (degrees)	s15Fixed16Number
24-27	channel #1 spot shape	see below
28-n	frequency, screen angle (degrees) and spot shape for additional channels	see below

TABLE 52.

Screening flag encoding is such that:

Attribute	Bit Position
Use Printer Default Screens (true is 1)	0
Frequency units in Lines/Inch (value 1) or Lines/cm (value 0)	1

TABLE 53.

Spot function encoding is such that:

Spot Function Value	Encoded Value
unknown	0
printer default	1
round	2
diamond	3
ellipse	4
line	5
square	6
cross	7

TABLE 54.

6.5.13 signatureType

The signatureType contains a four byte sequence used for signatures. Typically this type is used for tags that need to be registered and can be displayed on many development systems as a sequence of four characters. Sequences of less than four characters are padded at the end with spaces, 20h.

Byte Offset	Content
0-3	‘sig ‘(73696720h) type descriptor
4-7	reserved, must be set to 0
8-11	four byte signature

TABLE 55.

6.5.14 textDescriptionType

The textDescriptionType is a complex structure that contains three types of text description structures: 7 bit ASCII, Unicode and ScriptCode. Since no single standard method for specifying localizable character sets exists across the major platform vendors, including all three provides access for the major operating systems. The 7 bit ASCII description is to be an invariant, nonlocalizable name for consistent reference. It is preferred that both the Unicode and ScriptCode structures be properly localized.

The localized Macintosh profile description contains 67 bytes of data, of which at most ‘count’ bytes contain a ScriptCode string, including a null terminator. The ‘count’ cannot be greater than 67.

The count field for each types are defined as follows:

ASCII: The count is the length of the string in bytes including the null terminator.

Unicode: The count is the number of characters including a Unicode null where a character is always two bytes.

ScriptCode: The count is the length of the string in bytes including the terminating null.

If both Unicode and ScriptCode structures cannot be localized then the following guidelines should be used:

If Unicode is not native on the platform, then the Unicode language code and Unicode count should be filled in as 0, with no data placed in the Unicode localizable profile description area.

If Scriptcode is not native on the platform, then the ScriptCode code and ScriptCode count should be filled in as 0. The 67 byte localizable Macintosh profile description should be filled with 0's.

Byte Offset	**Content**	**Encoded as...**
0-3	'desc'(64657363h) type descriptor	
4-7	reserved, must be set to 0	
8-11	7 bit ASCII invariant description count, including terminating null (description length)	uInt32Number
12 - n-1	7 bit ASCII invariant description	
n - n+3	Unicode language code	uInt32Number
n+4 - n+7	Unicode localizable description count (description length)	uInt32Number
n+8 - m-1	Unicode localizable description	
m - m+1	ScriptCode code	uInt16Number
m+2	Localizable Macintosh description count (description length)	uInt8Number
m+3- m+69	Localizable Macintosh description	

TABLE 56.

An example of a textDescriptionType which has neither Unicode nor

ScriptCode is:

Byte Offset	Content
0-3	‘desc‘(64657363h)
4-7	uInt32Number = 0 (reserved)
8-11	uInt32Number = 15 (ASCII count)
12-26	ASCII string “An ICC Profile” with a NULL byte at the end
27-30	uInt32Number = 0 (Unicode code)
31-34	uInt32Number = 0 (Unicode count)
35-36	uInt16Number = 0 (ScriptCode code)
37	uInt8Number = 0 (ScriptCode count)
38-104	uInt8Number[67] filled with 0’s

TABLE 57.

Note: It has been found that textDescriptionType can contain misaligned data (see clause 4.1 for the definition of “aligned”). Because the Unicode language code and Unicode count immediately follow the ASCII description, their alignment is not correct if the ASCII count is not a multiple of four. The ScriptCode code is misaligned when the ASCII count is odd. Profile reading and writing software must be written carefully in order to handle these alignment problems.

6.5.15 textType

The textType is a simple text structure that contains a 7 bit ASCII text string. The length of the string is obtained by subtracting 8 from the element size portion of the tag itself. This string must be terminated with a 00h byte.

Byte Offset	Content
0-3	‘text‘(74657874h) type descriptor
4-7	reserved, must be set to 0
8-n	a string of (element size - 8) ASCII characters

TABLE 58.

6.5.16 u16Fixed16ArrayType

This type represents an array of generic 4 byte/32 bit quantity. The number of values is determined from the size of the tag.

Byte Offset	Content
0-3	‘uf32‘(75663332h) type descriptor
4-7	reserved, must be set to 0
8-n	an array of u16Fixed16Number values

TABLE 59.

6.5.17 ucrbgType

This type contains curves representing the under color removal and black generation and a text string which is a general description of the method used for the UCR and BG.

Byte Offset	Content	Encoded as...
0-3	‘bfd ’(62666420h) type descriptor	
4-7	reserved, must be set to 0	
8-11	count value specifying number of entries in the UCR curve	uInt32Number
12-m	actual UCR curve values starting with the zeroth entry and ending with the entry count-1. If the count is 1, the value is a percent.	uInt16Number[]
m+1 - m+4	count value specifying number of entries in the BG curve	uInt32Number
m+5 - n	actual BG curve values starting with the zeroth entry and ending with the entry count-1. If the count is 1, the value is a percent.	uInt16Number[]
n+1 - p	a string of ASCII characters, with a null terminator.	

TABLE 60.

Note: It has been found that ucrbgType can contain misaligned data (see clause 4.1 for the definition of “aligned”). Because the BG count immediately follows the UCR curve values, its alignment is not correct if the UCR count is odd. Profile reading and writing software must be written carefully in order to handle this alignment problem.

6.5.18 uInt16ArrayType

This type represents an array of generic 2 byte/16 bit quantity. The number of values is determined from the size of the tag.

Byte Offset	Content
0-3	‘ui16‘(75693136h) type descriptor
4-7	reserved, must be set to 0
8-n	an array of unsigned 16 bit integers

TABLE 61.

6.5.19 uInt32ArrayType

This type represents an array of generic 4 byte/32 bit quantity. The number of values is determined from the size of the tag.

Byte Offset	Content
0-3	‘ui32‘(75693332h) type descriptor
4-7	reserved, must be set to 0
8-n	an array of unsigned 32 bit integers

TABLE 62.

6.5.20 uInt64ArrayType

This type represents an array of generic 8 byte/64 bit quantity. The number of values is determined from the size of the tag.

Byte Offset	Content
0-3	‘ui64‘(75693634h) type descriptor
4-7	reserved, must be set to 0
8-n	an array of unsigned 64 bit integers

TABLE 63.

6.5.21 uInt8ArrayType

This type represents an array of generic 1 byte/8 bit quantity. The number of values is determined from the size of the tag.

Byte Offset	Content
0-3	‘ui08‘(75693038h) type descriptor
4-7	reserved, must be set to 0
8-n	an array of unsigned 8 bit integers

TABLE 64.

6.5.22 viewingConditionsType

This type represents a set of viewing condition parameters including: absolute illuminant white point tristimulus values and absolute surround tristimulus values.

Byte Offset	Content	Encoded as...
0-3	‘view‘(76696577h) type descriptor	
4-7	reserved, must be set to 0	
8-19	absolute XYZ value for illuminant in cd/m^2	XYZNumber
20-31	absolute XYZ value for surround in cd/m^2	XYZNumber
32-35	illuminant type	as described in measurement-Type

TABLE 65.

6.5.23 XYZType

The XYZType contains an array of three encoded values for the XYZ tristimulus values. The number of sets of values is determined from the size of the tag. The byte stream is given below. Tristimulus values must be non-negative. The signed encoding allows for implementation optimizations by minimizing the number of fixed formats.

Byte Offset	Content	Encoded as...
0-3	'XYZ '(58595A20h) type descriptor	
4-7	reserved, must be set to 0	
8-n	an array of XYZ numbers	XYZNumber

TABLE 66.

Annex A: Color Spaces

The International Color Profile Format supports a variety of both device-dependent and device-independent color spaces divided into three basic families: 1) CIEXYZ based, 2) RGB based, and 3) CMY based.

The CIE color spaces are defined in CIE publication 15.2 on Colorimetry. A subset of the CIEXYZ based spaces are also defined as connection spaces. The device dependent spaces below are only representative and other device dependent color spaces may be used without needing to update the profile format specification or the software that uses it.

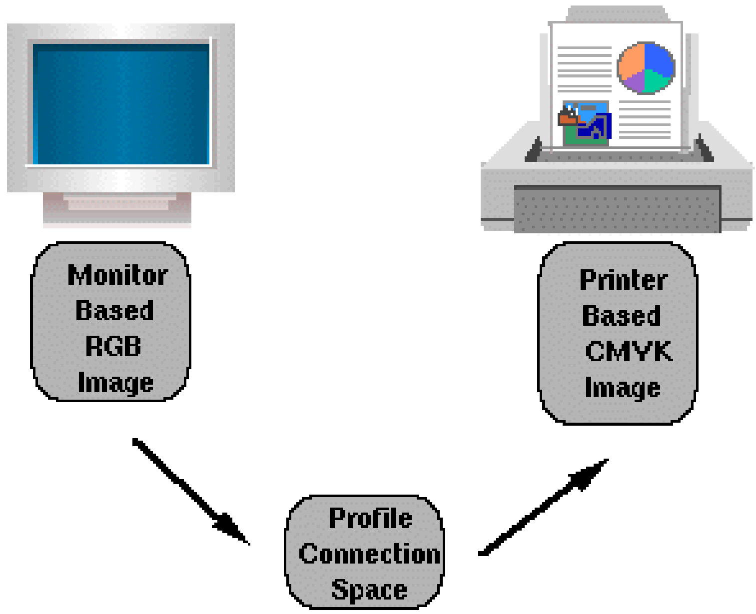
Base Space	**Description**	**Derivative Space**
CIEXYZ	base CIE device-independent color space	CIELAB
GRAY	monochrome device-dependent color space	
RGB	base additive device-dependent color space	HLS, HSV
CMY	base subtractive device-dependent color space	CMYK

TABLE 67.

A.1 Profile Connection Spaces

A key component of these profiles is a well-defined profile connection space. This space is the interface which provides an unambiguous connection between the input and output profiles as illustrated in the diagram below. The profile connection space is based on the CIE 1931 standard observer. This experimentally derived standard observer provides a very good representation of the human visual system color matching capabilities. Unlike device dependent color spaces, if two colors have the same CIE colorimetry they will match if viewed under the same conditions. Because the imagery is typically produced for a wide variety of viewing environments, it is necessary to go beyond simple application of the CIE system.

FIGURE 3.



The profile connection space is defined as the CIE colorimetry which will produce the desired color appearance if rendered on a reference imaging media and viewed in a reference viewing environment. This reference corresponds to an ideal reflection print viewed in an ANSI standard viewing booth.

The default measurement parameters for the profile connection space and all other color spaces defined in this specification are based on the ANSI CGATS.5-1993 standard, "Graphic technology - Spectral measurement and colorimetric computation for graphic arts images." Essentially this defines a standard illuminant of D50, the 1931 CIE standard observer, and 0/45 or 45/0 reflectance measurement geometry. The reference viewing condition is ANSI PH2.30-1989, which is a D50 graphic arts viewing environment.

One of the first steps in profile building involves measuring the colorimetry of a set of colors from some imaging media or display. If the imaging media or viewing environment differ from the reference, it will be necessary to adapt the measured colorimetry to that appropriate for the profile connection space. These adaptations account for such differences as white point chromaticity and luminance relative to an ideal reflector, maximum density, viewing surround, viewing illuminant, and flare. Currently, it is the responsibility of the profile builder to do this adaptation.

However, the possibility of allowing a variable illuminant in the PCS is under active consideration by the International Color Consortium. For this reason, a PCS illuminant field is in the profile header, but must be set to the CIE Illuminant D50 [X=0.9642, Y=1.0000, Z=0.8249].

The PCS is based on relative colorimetry. This is in comparison to absolute colorimetry. In absolute colorimetry colors are represented with respect to the illuminant, for example D50. In relative colorimetry, colors are represented with respect to a combination of the illuminant and the media's white, e.g. unprinted paper. The translation from relative colorimetry XYZ data, XYZ_r to absolute colorimetric data, XYZ_a, is given by

$$X_a = \left(\frac{X_{mw}}{X_i}\right) \cdot X_r \qquad (A1)$$

$$Y_a = \left(\frac{Y_{mw}}{Y_i}\right) \cdot Y_r \qquad (A2)$$

$$Z_a = \left(\frac{Z_{mw}}{Z_i}\right) \cdot Z_r \qquad (A3)$$

where XYZ_{mw} represents the media's white and XYZ_i represents the illuminant white.

The actual media and actual viewing conditions will typically differ from the reference conditions. The profile specification defines tags which provide information about the actual white point and black point of a given media or display. These tags may be used by a CMM to provide functionality beyond that of the default. For example, an advanced CMM could use the tags to adjust colorimetry based on the Dmin of a specific media. A tag is also provided to describe the viewing environment. This information is useful in choosing a profile appropriate for the intended viewing method.

There are many ways of encoding CIE colorimetry. This specification provides three methods in order to satisfy conflicting requirements for accuracy and storage space. These encodings, an 8 bit/component CIELAB encoding, a 16 bit/component CIELAB encoding, and a 16 bit/component CIEXYZ encoding are described below. The CIEXYZ space represents a linear transformation of the derived matching responses and the CIELAB space represents a transformation of the CIEXYZ space into one that is nearly perceptually uniform. This uniformness allows color errors to be equally weighted throughout its domain. While supporting multiple CIE encodings increases the complexity of color management, it provides immense flexibility in addressing different user requirements such as color accuracy and memory footprint.

It is important to understand that the PCS encodings do not represent a quantization of the connection space. The purpose of the encodings is to allow points within the space to be specified. Since the processing models benefit from interpolation between table entries, the interpolated AToB results should be used as the inputs to the BToA transforms. The AToB results should not be rounded to the nearest encoding value.

For the CIEXYZ encoding, each component (X, Y, and Z) is encoded as a fixed unsigned 16-bit quantity which has 15 fractional bits (u1.15).

An example of this encoding is:

0	0000h
1.0	8000h
1 + (32767/32768)	FFFFh

TABLE 68.

The encoding encompasses a large range of values. However, not all encodable values are useable. Since the PCS represents an ideal reflection print, and the media is a perfect diffuser, the largest valid XYZ values are those of the PCS illuminant (specified in the profile header).[1] This encoding was chosen to allow for PCS illuminants that have an X or Z greater than 1.0.

For the CIELAB PCS encodings, the L* values have a different encoding than the a* and b* values. The L* encoding is:

Value (L*)	**8 bit**	**16 bit**
0	00h	0000h
100.0	FFh	FF00h
100 + (25500/65280)	*none*	FFFFh

TABLE 69.

You can convert between the 8-bit and 16-bit encodings by multiplying or dividing by 256.

Although the 16-bit encoding can represent values slightly greater than 100.0, these are not valid PCS L* values and they should not be used.

1. For a D50 illuminant, the largest valid XYZ values are [0.9642, 1.0, 0.8249], or [7B6Bh, 8000h, 6996h] in encoded form. Note that the PCS illuminant values are stored in s15.16 format, so you must translate them to u1.15 format to find the encoded PCS limits.

The a* and b* encoding is:

Value (a* or b*)	8 bit	16 bit
-128.0	00h	0000h
0	80h	8000h
127.0	FFh	FF00h
127 + (255/256)	*none*	FFFFh

TABLE 70.

Note that this is not "two's complement" encoding, but a linear scaling after an offset of 128. This encoding was chosen to prevent discontinuities in CLUTs when going from negative to positive values.

You can convert between the 8-bit and 16-bit encodings by multiplying or dividing by 256.

Note that the 16-bit encoding can represent values slightly greater than 127.0. Since a* and b* have no defined limits, these are valid PCS values.

Because of the way that PCS encodings map to input tables, the BToAn tags must be able to handle invalid PCS values. However, the results of sending invalid values to these tags is up to the creator of the profile.

An important point to be made is that the PCS is not necessarily intended for the storage of images. A separate series of “interchange color spaces” may be defined in a future version of this specification for this purpose. The design choices made for these spaces (colorimetric encoding, reference media, viewing conditions, etc.) might be different than that of the PCS.

Annex B: Embedding Profiles

This annex details the requirements and options for embedding device profiles within PICT, EPS, TIFF, and JFIF documents. All profiles except abstract profiles can be embedded. The complete profile must be embedded with all tags intact and unchanged.

Embedding DeviceLink profiles renders the color data device dependent and significantly reduces portability. This may be useful in some situations, but may also cause problems with accurate color reproduction.

B.1 Embedding ICC Profiles in PICT Files

Apple has defined a new QuickDraw picture comment type for embedded ICC profiles. The picture comment value of 224 is followed by a 4-byte selector that describes the type of data in the comment. Using a selector allows the flexibility to embed more CMM related information in the future. The following selectors are currently defined:

Selector	Description	
0	Beginning of an ICC profile.	Profile data to follow.
1	Continuation of ICC profile data.	Profile data to follow.
2	End of ICC profile data.	No profile data follows.

TABLE 71.

Because the dataSize parameter of the PicComment procedure is a signed 16-bit value, the maximum amount of profile data that can be embedded in a single picture comment is 32763 bytes (32767 - 4 bytes for the selector). You can embed a larger profile by using multiple picture comments of selector type 1. The profile data must be embedded in consecutive order, and the last piece of profile data must be followed by a picture comment of selector type 2.

All embedded ICC profiles, including those that fit within a single picture comment, must be followed by the end-of-profile picture comment (selector 2), as shown in the following examples.

Example 1: Embedding a 20K profile.

```
PicComment kind = 224, dataSize = 20K + 4, selector = 0, profile data = 20K

PicComment kind = 224, dataSize = 4, selector = 2
```

Example 2: Embedding a 50K profile.

```
PicComment kind = 224, dataSize = 32K, selector = 0, profile data = 32K - 4
```

```
PicComment kind = 224, dataSize = 18K + 8, selector = 1, profile data = 18K + 4

PicComment kind = 224, dataSize = 4, selector = 2
```

In ColorSync 1.0, picture comment types `CMBeginProfile` (220) and `CMEndProfile` (221) are used to begin and end a picture comment. The `CMBeginProfile` comment is not supported for ICC profiles; however, the `CMEndProfile` comment can be used to end the current profile and begin using the System Profile for both ColorSync 1.0 and 2.0.

The `CMEnableMatching` (222) and `CMDisableMatching` (223) picture comments are used to begin and end color matching in both ColorSync 1.0 and 2.0

See "Advanced Color Imaging on the Mac OS", Apple Computer 1995, for more information about picture comments.

B.2 Embedding ICC Profiles in EPS Files

There are two places within EPS files that embedding International Color Consortium (ICC) profiles are appropriate. 1) Associated with a screen preview. 2) Associated with the page description. Embedding ICC profiles within a screen preview is necessary so that applications using this screen preview to display a representation of the EPS page description can do so with accurate colors. Embedding ICC profiles within a page description is necessary so that sophisticated applications, such as OPI server software, can perform color conversions along with image replacement. For general information concerning PostScript's Document Structuring Conventions (DSC), the EPS file format, or specific PostScript operators, see the PostScript Language Reference Manual, second edition.

1) There are a variety of different methods of storing a screen preview within an EPS file depending on the intended environment. For cross platform applications with embedded ICC profiles, TIFF screen previews are recommended. The TIFF format has been extended to support the embedding of ICC profiles. ICC profiles can also be embedded in a platform specific manner. For example on the Macintosh, Apple has defined a method for embedding ICC profiles in PICT files, see clause B.1: "Embedding ICC Profiles in PICT Files".

Note that a given page description may use multiple distinct color spaces. In such cases, color conversions must be performed to a single color space to associate with the screen preview.

2) ICC profiles can also be embedded in the page description portion of an EPS file using the `%%BeginICCProfile:` / `%%EndICCProfile` comments. This convention is defined as follows.

```
%%BeginICCProfile:  <profileid> <numberof> [<type> [<bytesorlines>]]
<profileid> ::= <text> (Profile ID)
<numberof> ::= <int> (Lines or physical bytes)
<type> ::= Hex | ASCII (Type of data)
<bytesorlines> ::= Bytes | Lines  (Read in bytes or lines)
%%EndICCProfile (no keywords)
```

These comments are designed to provide information about embedded ICC profiles. If the type argument is missing, ASCII data is assumed. ASCII refers to an ASCII base-85 representation of the data. If the `bytesorlines` argument is missing, `<numberof>` shall be considered to indicate bytes of data. If `<numberof>` = -1, the number of bytes of data are unknown. In this case, to skip over the profile one must read data until the encountering the `%%EndICCProfile` comment.

`<profileID>` provides the profile's ID in order to synchronize it with PostScript's setcolorspace and findcolorrendering operators and associated operands (see below). Note that `<numberof>` indicates the bytes of physical data, which vary from the bytes of virtual data in some cases. With hex, each byte of virtual data is represented by two ASCII characters (two bytes of physical data). Although the PostScript interpreter ignores white space and percent signs in hex and ASCII data, these count toward the byte count.

Each line of profile data shall begin with a single percent sign followed by a space (%). This makes the entire profile section a PostScript language comment so the file can be sent directly to a printer without modification. The space avoids confusion with the open extension mechanism associated with DSC comments.

ICC profiles can be embedded within EPS files to allow sophisticated applications, such as OPI server software, to extract the profiles, and to perform color processing based on these profiles. In such situations it is desirable to locate the page description's color space and rendering intent, since this color space and rendering intent may need to be modified based on any color processing. The `%%BeginSetColorSpace:` / `%%EndSetColorSpace` and `%%BeginRenderingIntent:` / `%%EndRenderingIntent` comments are used to delimit the color space and rendering intent respectively.

```
%%BeginSetColorSpace: <profileid>
<profileid> ::= <text> (ICC Profile ID)
%%EndSetColorSpace (no keywords)
```

`<profileid>` provides the ICC profile's ID corresponding to this color space. The ICC profile with this profile must have occurred in the PostScript job using the `%%BeginICCProfile:` / `%%EndICCProfile` comment convention prior to this particular `%%BeginSetColorSpace:` comment.

An example usage is shown here for CIE 1931 (XYZ)-space with D65 white point that refers to the ICC profile with `<profileid>` = XYZProfile.

```
%%BeginSetColorSpace: XYZProfile
[/CIEBasedABC <<
/WhitePoint [0.9505 1 1.0890]
/RangeABC [0 0.9505 0 1 0 1.0890]
/RangeLMN [0 0.9505 0 1 0 1.0890]
>>] setcolorspace
%%EndSetColorSpace
```

Note that the setcolorspace command is included within the comments. The PostScript enclosed in these comments shall not perform any other operations other than setting the color space and shall have no side effects.

```
%%BeginRenderingIntent: <profileid>
<profileid> ::= <text>            (ICC Profile ID)
%%EndRenderingIntent (no keywords)
```

`<profileid>` provides the ICC profile's ID corresponding to this rendering intent. The ICC profile with this profile must have occurred in the PostScript job using the `%%BeginICCProfile:` / `%%EndICCProfile` comment convention prior to invocation of this particular `%%BeginRenderingIntent:` comment.

An example usage is shown here for the Perceptual rendering intent that refers to the ICC profile with `<profileid>` = RGBProfile.

```
%%BeginRenderingIntent: RGBProfile
/Perceptual findcolorrendering pop
/ColorRendering findresource setcolorrendering
%%EndRenderingIntent
```

Note that the setcolorrendering command is included within the comments. The PostScript enclosed in these comments shall not perform any other operations other than setting the rendering intent and shall have no side effects.

B.3 Embedding ICC Profiles in TIFF Files

The discussion below assumes some familiarity with TIFF internal structure. It is beyond the scope of this document to detail the TIFF format, and readers are referred to the "TIFF™ Revision 6.0" specification, which is available from Adobe Systems Incorporated.

The International Color Consortium has been assigned a private TIFF tag for purposes of embedding ICC device profiles within TIFF image files. This is not a required TIFF tag, and Baseline TIFF readers are not currently required to read it. It is, however, strongly recommended that this tag be honored.

An ICC device profile is embedded, in its entirety, as a single TIFF field or Image File Directory (IFD) entry in the IFD containing the corresponding image data. An IFD should contain no more than one embedded profile. A TIFF file may contain more than one image, and so, more than one IFD. Each IFD may have its own embedded profile. Note, however, that Baseline TIFF readers are not required to read any IFDs beyond the first one.

The structure of the ICC Profile IFD Entry is as follows.

Byte Offset	Content
0-1	The TIFFTag that identifies the field = 34675(8773.H)
2-3	The field Type = 7 = UNDEFINED (treated as 8-bit bytes).
4-7	The Count of values = the size of the embedded ICC profile in bytes.
8-11	The Value Offset = the file offset, in bytes, to the beginning of the ICC profile.

TABLE 72.

Like all IFD entry values, the embedded profile must begin on a word boundary, so the Value Offset will always be an even number.

A TIFF reader should have no knowledge of the internal structure of an embedded ICC profile and should extract the profile intact.

B.4 Embedding ICC Profiles in JFIF Files

The JPEG standard (ISO DIS 10918-1) supports application specific data segments. These segments may be used for tagging images with ICC profiles. The APP2 marker is used to introduce the tag. Given that there are only 15 supported APP markers, there is a chance of many applications using the same marker. ICC tags are thus identified by beginning the data with a special null terminated byte sequence, "ICC_PROFILE".

The length field of a JPEG marker is only two bytes long; the length of the length field is included in the total. Hence, the values 0 and 1 are not legal lengths. This would limit maximum data length to 65533. The identification sequence would lower this even further. As it is quite possible for an ICC profile to be longer than this, a mechanism must exist to break the profile into chunks and place each chunk in a separate marker. A mechanism to identify each chunk in sequence order would thus be useful.

The identifier sequence is followed by one byte indicating the sequence number of the chunk (counting starts at 1) and one byte indicating the total number of chunks. All chunks in the sequence must indicate the same total number of chunks. The one-byte chunk count limits the size of embeddable profiles to 16,707,345 bytes.

Annex C: C Header File Example

This annex provides a cross-platform conditionally compilable header file for the ICC Profile Format.

Go to the ICC Web Site (www.color.org) for a machine-readable version of this C header file. You can also obtain it from the ICC Technical Secretary.

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

/***************************************************************
 Copyright (c) 1994-1997 SunSoft, 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 SunSoft, Inc.
shall not be used in advertising or otherwise to promote the
sale, use or other dealings in this Software without written
authorization from SunSoft Inc.
****************************************************************/

/*
 * This version of the header file corresponds to the profile
 * specification version 3.4.
 *
 * 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                 0x02100000L     /* 2.1.0, BCD */

/* 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 low 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 */

/*
 * Profile header flags, the low 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 */

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  /* 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' */
    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' */
    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' */
    icMaxEnumTag                        = 0xFFFFFFFFL
} 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                 = 0xFFFFFFFFL
} icTechnologySignature;

/* type signatures */
typedef enum {
    icSigCurveType                      = 0x63757276L,  /* 'curv' */
    icSigDataType                       = 0x64617461L,  /* 'data' */
    icSigDateTimeType                   = 0x6474696DL,  /* 'dtim' */
    icSigLut16Type                      = 0x6d667432L,  /* 'mft2' */
    icSigLut8Type                       = 0x6d667431L,  /* 'mft1' */
    icSigMeasurementType                = 0x6D656173L,  /* 'meas' */
    icSigNamedColorType                 = 0x6E636f6CL,  /* 'ncol'
                                                         * OBSOLETE, use ncl2 */
    icSigProfileSequenceDescType        = 0x70736571L,  /* 'pseq' */
    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' */
    icMaxEnumType                       = 0xFFFFFFFFL
} 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                       = 0xFFFFFFFFL
} 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                      = 0xFFFFFFFFL
} icProfileClassSignature;

/* Platform Signatures */
typedef enum {
    icSigMacintosh                      = 0x4150504CL,  /* 'APPL' */
    icSigMicrosoft                      = 0x4D534654L,  /* 'MSFT' */
    icSigSolaris                        = 0x53554E57L,  /* 'SUNW' */
    icSigSGI                            = 0x53474920L,  /* 'SGI ' */
    icSigTaligent                       = 0x54474E54L,  /* 'TGNT' */
    icMaxEnumPlatform                   = 0xFFFFFFFFL
} icPlatformSignature;

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

/* Measurement Flare, used in the measurmentType tag */
typedef enum {
    icFlare0                            = 0x00000000L,  /* 0% flare */
    icFlare100                          = 0x00000001L,  /* 100% flare */
    icMaxFlare                          = 0xFFFFFFFFL
} icMeasurementFlare;

/* 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                      = 0xFFFFFFFFL
} icMeasurementGeometry;

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

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


/*------------------------------------------------------------------------*/
/*
 * 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 */
    icMeasurementFlare          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.
 *
 * icTextDescription            deviceMfgDesc;  * Manufacturer text
 * icTextDescription            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        desc[icAny];    /* Null terminated string */
} icCrdInfo;

/*------------------------------------------------------------------------*/
/*
 * 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 */

/*------------------------------------------------------------------------*/

/*
 * 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 */
```

Annex D: PostScript Level 2 Tags

The PostScript Level 2 tags are provided in order to control exactly the PostScript Level 2 operations that should occur for a given profile. These tags are only valid for PostScript Level 2 (and conceivably future versions of PostScript) devices, and are not generally supported in PostScript Level 1 devices. In addition, some of the tags may correspond to PostScript operations that are not supported in all PostScript Level 2 devices. Using such tags requires first checking for the available operators. All operators described in the PostScript Language Reference Manual, second edition, are available on all PostScript Level 2 devices. Documentation for extensions to PostScript Level 2 are available through Adobe's Developer Support Organization. In addition, guidelines for PostScript compatibility with this profile format are available. For details of such operator support, compatibility guidelines, the PostScript Level 2 device independent color model, or other PostScript-related issues contact Adobe's Developer Support Organization.

In general, there is a straightforward relationship between the profile's header fields and tags, and these PostScript tags. It is anticipated that the various CMSs that support this profile format will also provide support for these optional PostScript tags. To verify such support contact the CMS vendors directly. In cases where such support is provided, and the desired model of operations is the same for PostScript processing as it is for CMS processing, these tags can be omitted, since all necessary information is in the profile itself. In the case where such CMS support is in question or processing different than that provided by an arbitrary CMS is desired, these tags can be populated to provide exact control over the PostScript processing. For example, if private tags are used in the profile to achieve a non-public type of processing on certain CMSs, such processing can be achieved on a PostScript device by populating the appropriate PostScript tags.

Some of the PostScript tags have a tag type of textType or uInt8Type. This choice is provided in order to match the properties of the communications channel to the data in these tags. Encoding the data in uInt8Type form is recommended to save memory and/or reduce transmission times. Applications and drivers may convert it to ASCII Coded PostScript, Binary Coded PostScript, or Token Binary Coded PostScript or leave it in binary format to match the requirements of the communications channel. Applications and drivers are responsible for this potential conversion from binary data to channel compatible data. The data should be encoded in textType in those cases where the amount of data is relatively small or where the conversion from binary to channel compatible data is not available.

The PostScript contained in these tags is not self evaluating - it simply provides operands. These operands must be followed by operators like setcolorspace, setcolorrendering, and findcolorrendering.

D.1 Synchronizing Profiles and CRDs

ICC profiles (profiles) on the host and PostScript color rendering dictionaries (CRDs) in the printer can contain identical information for color transformations. In order to reduce printer memory requirements and PostScript transmission times it is advantageous to synchronize profiles and CRDs.

CRDs should be created with an optional CreationDate field indicating the date/time of CRD creation or most recent modification. This should correspond to the date/time field of any companion profiles. This profile information is available in the profile's header and the calibrationDateTimeTag. Even if no companion profile is constructed, this field should still be supplied in CRDs. Companion means embodying the same transformation, but in a different format -- profile vs. CRD.

CreationDate is a PostScript string whose format closely follows the format defined by the international standard ASN.1 (Abstract Syntax Notation One, defined in CCITT X.208 or ISO/IEC 8824). This string is of the form:

```
(YYYYMMDDHHmmSSOHH'mm')
```

where

`YYYY` is the year

`MM` is the month (01-12)

`DD` is the day (01-31)

`HH` is the hour (00-23)

`mm` are the minutes (00-59)

`SS` are the seconds (00-59)

`O` is the relation of local time to GMT where + indicates that local time is later than GMT, - indicates that local time is earlier than GMT, and Z indicates that local time is GMT

`HH'` is the absolute value of the offset from GMT in hours

`mm'` is the absolute value of the offset from GMT in minutes

Trailing fields other than the year are optional. The default value for day and month is 1; all other numerical fields default to 0. If no GMT information is specified, the relationship of the specified time to GMT is considered unknown. Whether the time zone is known or not, the rest of the date should be specified in local time.

Profiles should be extended with the optional tag crdInfoTag. This tag contains the PostScript product name to which this profile corresponds and the names of the companion CRDs. See clause 6.5.1 "crdInfoType" for the format.

CreationDate and crdInfoTag can be synchronized in different ways depending on the availability of bidirectional communications between the host and printer, as well as whether the CRD came with the printer or was downloaded from the host in the field.

When bidirectional communications are available one can query the printer and determine the availability of a given CRD and its associated CreationDate. When such communications are not available, the list of printer resident CRDs and their CreationDate entries are available through the printer's PPD and the host profile registry.

PPDs currently contain the names of the CRDs that ship with a printer. The PPD format will be extended to also contain the CreationDate entry for each CRD. For those CRDs downloaded (or deleted) in the field, the registry should be updated accordingly. The existence and form of this registry may differ from platform to platform.

There are 3 cases to consider.

1) CRDs and profiles made together.

If one wants to know whether it is necessary to construct and download a CRD for a given profile one can:

a) Optionally check to see if the profile corresponds to the printer by comparing the PostScript product name field in the crdInfoTag and the printer's product name. The product name for the printer is determined using the PostScript product operator or from the PPD. One may want to perform this step to limit the user's selection of profiles to only those appropriate for the printer.

b) Based on the desired rendering intent, check to see if the printer has a CRD of the appropriate name as found in crdInfoTag.

c) If b) is successful, compare the profile's date/time field to the CRD's CreationDate. If a match, then the profile doesn't need to be downloaded -- the companion CRD already exists and is used for the job. If not successful in a), b) or c) see 2) below.

2) CRDs generated from ICC profiles and then downloaded.

One can download a CRD for just a job in which case future synchronization is not an issue. Or one can make the CRD persistent in the printer. In this case one fills in the CRD CreationDate field, updates the registry, and updates the profile to have the appropriate crdInfoTag tag.

3) CRD in printer for which no profiles exist.

This is mainly a problem for CRDs that exist today without companion profiles. If one wants to make a companion profile and have synchronization with CRDs one can:

a) Use the CRD's CreationDate field if available for the date/time field of the profile.

b) Update the CRD in the printer, and the registry, with the CreationDate corresponding to the new profile's date/time field.

In both cases the profile's crdInfoTag should be filled in appropriately.

This works like 1) with the exception that the CRD updates may be volatile to power cycles of the printer. Such power cycles should result in the registry being updated.

Annex E: Profile Connection Space Explanation

E.1 Introduction

This Appendix is intended to clarify certain issues of interpretation in the ICC Profile Format.

The goal of color management is to provide the capability of maintaining control over color rendering among various devices and media that may be interconnected through a computer system or network. To achieve this goal, the color characteristics of each device are determined and encapsulated in a *device profile*, which is a digital representation of the relation between device coordinates and a device-independent specification of color.

By *device coordinates* we mean the numerical quantities through which a computer system communicates with a color peripheral—such as the digital code values used to drive a monitor or printer, or the digital signals received from a scanner. These quantities are usually labeled *RGB* (or *CMYK*), but the labels identify the channels of the device rather than specific visual colors; the quantities are often encoded as unsigned 8-bit integers for each channel in the typical digital interface.

The *device-independent specification* is best given in a color space based on human visual experience. Thus, a device profile provides a means of translating (or transforming) color image data from device coordinates into a visual color space or vice versa.

Furthermore, if the various profiles available to a color-management system are referenced to the *same* visual color space, the system can translate data from one device's coordinates to another's—while maintaining color consistency—by (conceptually) passing through the intermediary of the visual color space; the latter, then, constitutes a standard interface for color communication, allowing profiles to be connected together in a meaningful sequence. A color space used in this way may be termed a *Profile Connection Space* (PCS). For example, the transformation of a color image from a scanner into monitor coordinates can be described as a transformation into the PCS (via the scanner's device profile) followed by a transformation out of the PCS (via the monitor's device profile). In practice, these successive transformations may be implemented in a variety of ways, and the image may never actually be represented in the PCS on disk or in computer memory. Thus, the PCS is to be regarded as a convenient reference for the definition of profiles—as an intermediate, or virtual, stage of the image processing—, in contrast to an *interchange* or *exchange color space*, which is an encoding for the storage and transmission of color images. The issues regarding the choice or design of a PCS are somewhat different from those related to an interchange space; this Annex is concerned only with PCS issues.

A PCS consists of a coordinate system for color space and an interpretation of the data represented in that coordinate system. In fact, multiple coordinate systems can easily be supported in the same or different color-management systems, as long as they share a common interpretation, since it is usually a well-defined and relatively simple mathematical task to transform from one coordinate system to another. However, if the interpretation of the represented colors is different, there may be no satisfactory way of translating the data from one to another.

The purpose of this paper is to present an unambiguous interpretation for the PCS implicit in the ICC Profile Format. It is especially important in the heterogeneous environments currently found on desktop platforms and networks to establish this interpretation in an open, non-proprietary specification, so that different color-management systems can communicate with each other and exchange profiles within and across platforms and operating systems.

E.2 Colorimetry and Its Interpretation

The issue of interpretation has received little attention in the recent past, because it has been widely believed that the choice of a suitable coordinate system—preferably one founded on *CIE colorimetry*, a system of measurement and quantification of visual color stimuli created and promoted by the *Commission Internationale de l'Éclairage*—would suffice to guarantee device independence. The notion was that colorimetric matching of the renderings on various media was the key to satisfactory color reproduction, and that interpretation was not needed. However, although colorimetry can be an essential element of a successful approach to color management, it is usually necessary to modify the colorimetric specification for renderings on different media.

Different media require different physical color stimuli, in certain cases, because they will be viewed in different environments—e.g., different surround conditions or illuminants; the observers, therefore, will experience different adaptive effects. In order to preserve the same *color appearance* in these different environments, the colorimetry must be corrected to compensate for the adaptation of the human visual system and for physical differences in the viewing environments, such as flare. Although color appearance is still an active research topic, the most common forms of adaptation are understood reasonably well, so that the required corrections in the colorimetry for different viewing conditions can be modeled with sufficient accuracy.

There are other reasons why the colorimetry may be altered for specific media. For instance, hard-copy media—*even those intended for the same viewing conditions*—differ considerably in their dynamic range and color gamut. A well-crafted rendering of an image on a specific medium will take advantage of the capabilities of that medium without creating objectionable artifacts imposed by its limitations. For instance, the tone reproduction of the image should provide sufficient contrast in the midtones without producing blocked-up shadows or washed-out highlights. The detailed shape of the tone curve will depend on the minimum and maximum densities (*D*min and *D*max) attainable in the medium. Clearly, there is considerable art involved in shaping the tone-reproduction and color-reproduction characteristics of different media, and much of this art is based on subjective, aesthetic judgments. As a result, the substrate (paper, transparency material, etc.) and the colorants used in a medium will be exploited to impart a particular "personality" to the reproduction that is characteristic of the medium.

Furthermore, the desired behavior of a color-management system depends strongly on artistic intent. If the output medium is identical to the input medium—say, 35-mm slides—, the desired behavior is typically to create a duplicate of the original. But if the two media are different, it is not so obvious what the default behavior should be. In some cases, the intent may be to retain all or part of the personality of the original; in other cases, it may be more important to remove the personality of the original and replace it with a fresh rendering that has the full personality of the output medium. Sometimes the simulation of a third medium may be important—as when an image is displayed on a monitor to preview a rendering on a dye-diffusion printer, retaining (as well as possible) the personality of an original image scanned from a photographic print! It is essential to the success of color-management systems that a broad range of options be kept open. The interpretation of the PCS merely defines the particular default behavior that will be facilitated by the system without explicit intervention by the application or user. Alternative behaviors are not excluded by this choice; they simply will not be the default and will require more work.

With this context in mind, we present the following interpretation:

The PCS represents desired color appearances.

Here, the term *desired* is used to indicate that the interpretation is oriented towards colors to be produced on an output medium. It also is used to imply that these colors are not restricted by the limitations of any particular output medium. It is helpful here to conceptualize a "reference reproduction medium", with a large gamut and dynamic range, as the target medium for the desired colors. Consequently, it is the responsibility of the output device profiles to clip or compress these colors into the gamut of the actual output media. And, of course, "desired" also implies the expression of artistic intent.

The term *color appearance* is used to imply that adaptive effects are taken into account. Associated with the reference reproduction medium is a "reference viewing environment". More precisely, therefore, the PCS represents the "desired color appearances" in terms of *the CIE colorimetry of the colors to be* *rendered on the reference medium and viewed in the reference environment*. Output profiles for media that are viewed in different environments are responsible for modifying the colorimetry to account for the differences in the observer's state of adaptation (and any substantial differences in flare light present in these environments), so that color appearance is preserved. Similarly, input profiles are responsible for modifying the colorimetry of the input media to account for adaptation and flare; they also have the responsibility to account for the artistic intent implicit in the word "desired".

We define the *reference reproduction medium* as an idealized print, to be viewed in reflection, on a "paper" that is a perfect, non-selective diffuser (i.e., *D*min = 0), with colorants having a large dynamic range and color gamut. We define the *reference viewing environment* to be the standard viewing booth (ANSI PH-2.30); in particular, it is characterized by a "normal" surround—i.e., where the illumination of the image is similar to the illumination of the rest of the environment—, and the adapting illuminant is specified to have the chromaticity of D50 (a particular daylight illuminant).

E.3 Color Measurements

The PCS, so interpreted, represents colors for a hypothetical reference medium; device profiles must relate these colors to those that can be measured on real media. For consistency of results, these measurements must be made in accordance with the principles of CIE colorimetry.

For one particular class of media (namely, those intended for the graphic-arts), the colorimetry should conform to graphic-arts standards for color measurement.[1] Here, the illuminant is specified to be D50, so that no corrections need to be applied for chromatic adaptation. The colorimetry standard is based on a theoretical D50 illuminant, as defined by the CIE in the form of a tabulated spectral distribution. However, the fluorescent D50 simulators found in typical professional viewing booths have rather different spectral distributions, and the color stimuli produced can be noticeably different.[2] Often, better results can be obtained by basing the colorimetry on the actual, rather than the theoretical, illumination source; unfortunately, there is no standardized, practically realizable source.

For other, non-graphic-arts media, the illuminant may be different from D50. In general, for best results, the actual illumination spectrum should be used in the color measurements. And if the chromaticity of the illuminant is different from

1.IT8.7/3, "Graphic technology—Input data for characterization of 4-color process printing", draft standard of Subcommittee 4 (Color) of ANSI Committee IT8 (Digital Data Exchange Standards), 14 December 1992, Paragraph 4.2.

2.D. Walker, "The Effects of Illuminant Spectra on Desktop Color Reproduction", in *Device-Independent Color Imaging*, R. Motta and H. Berberian, ed., *Proc. SPIE*, **1909**, 1993, pp. 236–246.

that of D50, corrections for chromatic adaptation will be needed and will be incorporated into the device-profile transforms. This aspect of the PCS interpretation provides flexibility to the color-management system. For example, it will be possible to transform data from a medium intended for tungsten illumination to a medium intended for cool-white-fluorescent: the input profile handles the adaptation from tungsten to D50, and the output profile handles the adaptation from D50 to cool-white.

Since substantial flare (perhaps 2–3%) may be present in an actual viewing environment,[1] the colorimetry is defined in an ideal, flareless measurement environment; in this way, difficult telescopic color measurements in the viewing environment can be avoided, and simple contact instruments and/or controlled laboratory conditions can be used instead. (Corrections should be applied to the data for any appreciable flare in the actual measurement environment and instruments.)

E.4 Colorimetry Corrections and Adjustments in Output Profiles

The implications of this interpretation should be emphasized: the creator of a profile is obliged to correct and adjust the PCS data for various effects. Since the PCS is interpreted with an output orientation, we will first examine the nature of these corrections and adjustments for output profiles. Then, in the next section, we will discuss the consequences for input profiles.

Let us look at a number of possible output paths:

E.5 Output to reflection print media

Included here are computer-driven printers, off-press proofing systems, offset presses, gravure printing, photographic prints, etc. These are generally intended for “normal” viewing environments; but corrections may be needed—e.g., for chromatic adaptation, if the illuminant’s chromaticity is other than that of D50.

In the simplest scenario, the user desires to reproduce colors colorimetrically (aside from adaptive corrections) so as to attain an appearance match. A distinction can be made between “absolute” and “relative” colorimetry in this context. *Absolute* colorimetry coincides with the CIE system: color stimuli are referenced to a perfectly reflecting diffuser. All reflection print media have a reflectance less than 1.0 and cannot reproduce densities less than their particular *D*min. In a cross-rendering task, the choice of absolute colorimetry leads to a close appearance match over most of the tonal range, but, if the *D*min of the input medium is different from that of the output medium, the areas of the image that are left blank will be different. This circumstance has led to the use of

1.R.W.G. Hunt, *The Reproduction of Colour*, Fourth Edition, Fountain Press, 1987, pp. 52–53.

relative colorimetry, in which the color stimuli are referenced to the paper (or other substrate). This choice leads to a cross-rendering style in which the output image may be lighter or darker overall than the input image, but the blank areas will coincide. Both capabilities must be supported, since there are users in both camps. However, the default chosen for ICC is relative colorimetry.

This can be made more precise: the default “colorimetric” transform will effectively apply a scaling operation in the CIE 1931 *XYZ* color space:

$$X_a = (X_{mw} / X_i)\, X_r \quad \text{(EQ1)}$$

$$Y_a = (Y_{mw} / Y_i)\, Y_r \quad \text{(EQ2)}$$

$$Z_a = (Z_{mw} / Z_i)\, Z_r \quad \text{(EQ3)}$$

where $(XYZ)_r$ are the coordinates of a color in the PCS, $(XYZ)_a$ are the coordinates of the corresponding color to be produced on the output medium, $(XYZ)_i$ are the coordinates of the lightest neutral represented in the PCS (namely, one with the chromaticity of D50 and a luminance of 1.0), and $(XYZ)_{mw}$ are the coordinates of the output paper (or other substrate) *adapted to the PCS illuminant* (D50). Thus, the lightest neutral in the PCS will be rendered as blank paper—regardless of the reflectance or color cast of the paper—; other neutrals and colors will be rescaled proportionately and will be rendered darker than the paper. Output on different reflection print media will then agree with the PCS and with each other in relative colorimetry and, therefore, in relative appearance.

In other cases, the preference may be for absolute colorimetry. This means that, within the limitations of the output medium, the CIE colorimetry of the output image should agree with values represented in the PCS. I.e., $X_a = X$, $Y_a = Y$, and $Z_a = Z$. One way of achieving this result is to apply a separate transformation to the PCS values, outside of the device profile (e.g., in application or system software):

$$X' = (X_i / X_{mw})\, X \quad \text{(EQ4)}$$

$$Y' = (Y_i / Y_{mw})\, Y \quad \text{(EQ5)}$$

$$Z' = (Z_i / Z_{mw})\, Z \quad \text{(EQ6)}$$

The relative values, X' Y' Z', can then be processed through the default colorimetric transform (i.e., they are effectively substituted for $(XYZ)_r$ in Equations 1–3) to achieve the desired result.

This capability depends on the availability to the color-management software of the colorimetry of the paper. The mediaWhitePointTag in the profile can be used for this purpose and should represent the adapted, absolute colorimetry of the lightest neutral that the device and/or medium can render (usually the blank substrate).

In either case, it may happen that the dynamic range and/or color gamut of the output medium is not sufficient to encompass all the colors encoded in the PCS. Some form of clipping will then occur—in the highlights, in the shadows, or in the most saturated colors. While an appearance match may be achieved over much of a color space, there will be a loss of detail in some regions. If this is objectionable, the operator should have an option for selecting a more explicit form of gamut compression to be applied to the colors as part of the output profile. ICC supports two styles of controlled gamut compression—“perceptual” and “saturation”—in addition to the “colorimetric” option, which clips abruptly at the gamut boundary. (An important case requiring explicit gamut compression is that of input from a transparency, where the dynamic range, even of the corrected colors, may exceed that of any reflection print medium.) Note that an explicit compression maps colors from the dynamic range and gamut of the reference medium to the range and gamut of the actual medium, so that only (XYZ)D50—i.e., the lightest PCS neutral—will be rendered as blank paper, just as in the relative-colorimetric case. This time, however, the entire tone scale may be readjusted, to keep the shadows from blocking up and to maintain proper midtones, and some in-gamut colors may be adjusted to make room for out-of-gamut colors.

E.6 Output to transparency media

This category might include overhead transparencies and large-format color-reversal media, as well as slide-production systems. Transparency materials are normally intended to be viewed by projection (using a tungsten lamp) in a dim or darkened room; in some cases, however, they are placed on a back-lit viewer for display, and in others they are used as a graphic-arts input medium, in which case they are examined on a light box or light table with the aid of a loupe. Accordingly, there are several possible viewing conditions for transparencies, requiring somewhat different corrections.

Typical color-reversal films have a much larger dynamic range than reflection media and higher midscale contrast. Their tone-reproduction characteristics have evolved empirically, but it may be plausible to explain them as partially compensating for dark-surround adaptation and the flare conditions typical in a projection room. The state of brightness adaptation in a projection room is also different from that in a reflection environment. To the extent that these explanations are valid, the colorimetry should be corrected for these effects. Furthermore, in some of these environments the visual system is partially adapted to a tungsten source, and chromatic corrections should be applied for the difference between tungsten and D50.

A “colorimetric” rendering, in this case, will actually produce an appearance match to the colors in the PCS, rather than a colorimetric match—i.e., the colors measured on the resulting transparency will differ from those encoded in the PCS, but will appear the same *when the transparency is viewed in its intended environment* as the PCS colors would if rendered on the reference medium and viewed in reflection.

Note that the lightest neutral, (XYZ)D50,will be rendered at or near Dmin of the transparency in the default (relative) colorimetric transform. An absolute-colorimetric rendering can be generated in software, as described above for reflection-print media.

Explicit gamut compression can be provided as an option; it normally would not be needed for images input from photographic media, but it may be useful for input from computer graphics, since some of the highly saturated colors available on a computer color monitor fall outside the gamut of transparency media.

E.7 Negative media

Here the target colors are those of a reflection print to be made from the negative. No adaptive corrections are required, unless the print is intended to be viewed under an illuminant other than D50. Explicit gamut compression is a useful option, and both relative and absolute colorimetric matches can be provided as in the case of direct-print media.

E.8 Monitor display

The viewing conditions of a CRT monitor may require some corrections to the colorimetry, due to the effects of surround and flare. Also, if the monitor's white point is other than D50, chromatic adaptation must be accounted for. When corrections for these effects are applied, the colors in the display should match the appearance of those in the PCS and should provide accurate and useful feedback to the operator.

In most cases, the rendering should be “colorimetric” (possibly including adaptive corrections), in order to achieve this result. (As for reflection print media, this would be “relative” by default, but “absolute” colorimetry is also supported.) In other cases (video production, perhaps), it may be more important to the user to create a pleasing image on the monitor (without having out-of-gamut colors block up, for instance) than to preserve an appearance match to the PCS; for that purpose, explicit gamut compression would be a useful option.

In many scenarios, the monitor display is not the end product, but rather a tool for an operator to use in controlling the processing of images for other renderings. For this purpose, it will be possible to simulate on the monitor the colors that would be obtained on various other output media. The PCS colors are first transformed into the output-device coordinates, using any preferred style of gamut compression. Then they are transformed back to the PCS by using the (colorimetric) inverse output transform. (These two steps can be replaced by an equivalent "preview" transform.) Finally they are transformed (colorimetrically) into monitor coordinates for previewing. The result of compression to the output gamut should then be visible in the displayed image.

E.9 Colorimetry Corrections and Adjustments in Input Profiles

The purpose of an input profile is to transform an image into the PCS—i.e., to specify the colors that are desired in the output. Since there are many possible intentions that a user might have for these colors, we cannot impose many restrictions on the nature of the transforms involved. Bearing in mind the capabilities of the output profiles, as just outlined, we can suggest the possibilities available to various classes of input profiles.

E.10 Scanned reflection prints

Here the intended viewing environment may be identical to the reference, but, if not, adaptive corrections should be applied to the colorimetry. In the simplest case, the profile may consist of a transformation from scanner signals to the colorimetry of the medium. In this case, the personality of the input medium has been preserved. If the output rendering is also "colorimetric", the result will be an appearance match to the original. Indeed, if the output medium is the same as the input medium, the result should be a close facsimile or duplicate of the original.

By default, the rendering is based on relative colorimetry, as discussed above. Therefore, it should be remembered, when creating an input profile, that the (XYZ)D50 point of the PCS will be mapped to the Dmin of the output medium. This implies that the Dmin of the input medium must be mapped to the (XYZ)D50 point of the PCS, in order to facilitate the duplication of an original and a relative-colorimetry match when cross-rendering.

In order to enable the alternative of absolute colorimetry, the "white point" field in the header of the input profile should be used to specify the colorimetry of the paper. This allows the absolute colorimetry of the original to be computed from relative colorimetry represented in the PCS, by analogy to Equations 1–3 above. These absolute color stimuli can then be converted to relative colorimetry for output by using the "white point" field of the output profile in Equations 4–6.

There are other possibilities, however. The input profile could be designed to remove some or all of the personality of the input medium, so that the PCS encoding makes use of more of the gamut and dynamic range of the reference medium. In these cases, it will probably be best to choose some form of explicit gamut compression in the output profile. The result may differ in appearance considerably from the original and will constitute a fresh rendering tuned to the capabilities and limitations of the output medium.

In any case, a calibrated color monitor, if available, can be used to display an accurate preview of the result.

E.11 Scanned transparencies

Since transparencies are intended for viewing in a variety of environments, different kinds of adaptive corrections may be applied to the colorimetry of the input medium to obtain colors in the PCS. For instance, the device profile might transform scanner signals into the colorimetry of a reference print that would have the same appearance in the reference environment as the transparency produces in a projection environment. (Note that there may be no actual reflection print medium that has sufficient dynamic range to reproduce all of these color appearances). In this scenario, the personality of the color-reversal film or other transparency material is retained, even though the colorimetry has been modified for the PCS; still, this may be loosely termed a “colorimetric” transform, since the only corrections are for flare and adaptation.

As in the case of input prints, there are other possibilities: some or all of the personality of the input medium can be removed, according to artistic intent, yielding different results, which also depend on the style of gamut compression selected for output.

Normally, the *D*min of the input medium should be mapped to (*XYZ*)D50 in the PCS. The absolute, adapted *XYZ* of the *D*min color is recorded in the “medium white point” tag.

E.12 Scanned negatives

Photographic negatives, of course, are not intended for direct viewing. Therefore, the colorimetry that is relevant here might be that of a hypothetical reflection print made from the negative and intended for viewing in the reference environment. No adaptive corrections should be applied. The personality of the result is that of the negative-positive system as a whole. Again, other possibilities exist, depending on artistic intent.

E.13 Computer graphics

Such imagery is usually synthesized in the *RGB* space of a display monitor that provides visual feedback to the operator. Thus, adaptive corrections may need to be applied to the colorimetry of the monitor to define the colorimetry of a reference print having the same appearance.

The personality here is that of the synthetic image on the monitor screen.

E.14 Scene capture

This pathway refers to video cameras, electronic still cameras, and other technologies (such as Photo CD™) that provide a capability of approximately determining the colorimetry of objects in a real-world scene. In most cases, the tone scale must be adjusted to provide enough contrast for viewing the reference medium in the reference environment; the colorfulness of the image should also be enhanced somewhat for that environment. The personality of the result, of course, depends on the nature of these adjustments.

E.15 Colorimetric input

In some cases, input colors are specified that are intended to be processed colorimetrically, without any tone shaping or chromatic enhancement. This might be the case, for instance, when a scene-capture device is used to record the colorimetry of real-world objects for scientific reasons, rather than for creating a pleasing reproduction. It may also be the case when particular spot colors are specified in colorimetric terms. In these cases, the specified colorimetric values are left intact in the transformation to the PCS; no adaptive corrections or adjustments are applied. The PCS values should be represented in relative colorimetry, and the “white point” tag specifies the reference point for the scaling. In some cases this reference point will have a luminance of 1.0, and there will be no difference between relative and absolute colorimetry. In other cases the reference point will have the colorimetry of (say) the paper stock used in a spot-color sample book or of a particular light neutral in a scene. In most of these cases, the preferred output rendering will also be “colorimetric”. By default, as before, this will entail relative colorimetry; absolute colorimetry can be achieved, outside of the default transforms, by taking account of the “white point” tags of the input and output profiles and converting appropriately.

An image of this kind can be said to have no personality.

As can be inferred from some of these examples, the user may have a choice of input profiles having different intents, as well as a choice among output transforms having different intents. The end result depends on both of these choices, which, for the most predictable color reproduction, should be made in coordination. To aid in this coordination, there are profile tags that specify the rendering intent and that distinguish between input transforms that are colorimetric (aside from possible corrections for flare and adaptation) and those that have applied adjustments to the colorimetry.

E.16 Techniques for Colorimetry Corrections

As we have seen, if the viewing conditions of the medium are different from the reference (e.g., projected slides or video viewed in dim or dark surround), corrections to the colorimetry of the reproduction should be applied.[1] These should be designed to correct for differences in the flare light present in these environments, as well as the effects of non-normal surround, brightness adaptation to the absolute radiant flux of the illumination, and any other effects that are found to be significant. And if the medium was intended to be viewed under an illuminant of different chromaticity than that of D50, the profile should incorporate corrections for chromatic adaptation; these can simply be based on a linear scaling in *XYZ* (which happens automatically in the CIELAB system); alternatively, it can be based on the linear Von Kries transformation[2] (or, if preferred, a more sophisticated, nonlinear model of color appearance, such as that of Hunt[3] or Nayatani[4]).

If the creators of device profiles universally apply these corrections to their colorimetric data, the PCS will have a universal, unambiguous interpretation, and images rendered “colorimetrically” will evoke (as nearly as possible) the same appearance, regardless of the medium and viewing environment of the reproduction. In this way, the same image can be rendered on photographic transparency material, various reflective print media, CRT’s, etc., and will, by and large, appear similar to the viewer. This goal cannot be achieved simply by matching the colorimetry of the reproductions. Various forms of explicit gamut compression and input effects can be made available for situations where other goals are important; the recommended PCS interpretation does not limit these possibilities in any way: it merely facilitates the default behavior of the color-management system.

1.Hunt, *op. cit.*, pp. 56–61.

2.R.W.G. Hunt, *Measuring Color*, Ellis Horwood, pp. 70–71.

3.*Ibid.*, pp. 146–173.

4.Y. Nayatani, K. Takahama, and H. Sobagaki, “Prediction of color appearance under various adapting conditions”, *Color Res. Appl.*, **11**, 62 (1986).

Annex F: Summary of Spec Changes

This annex lists the major changes made to the specification from Version 3.3 (November 1, 1996). Cosmetic changes are not listed.

1. The Definitions clause has been updated. (See clause 4.)
2. A new clause has been added for symbols and abbreviations used in the Specification. Abbreviations that were previously under Definitions have been moved to the new clause. (See clause 5.2.)
3. The dataType and textType descriptions now spell out how the data size is calculated. (See clauses 6.5.3 and 6.5.15.)
4. The method for embedding profiles in JFIF images has been added. (See Annex B.4.) *ICC #199701*
5. The C Header File Example has been updated. The conditional compilation has been altered to provide a default definition of the data types in all circumstances. The typedef for `icCrdInfoType` has been altered, and comments have been added to explain its use. (See Annex C:.)