International Color Consortium®

Specification ICC.2:2019

(Profile version 5.0.0 - iccMAX)

Image technology colour management — Extensions to architecture, profile format and data structure

[REVISION of ICC.2:2018]

Copyright notice

Licenses and trademarks

For additional information on the ICC

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

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 130, *Graphic technology*, in cooperation with the International Color Consortium (ICC).

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

0 General

This document defines specifications that provide a platform for defining extended (iccMAX) colour management profiles and systems for various colour workflow domains. It can be thought of as an extension to ISO 15076-1, defined by the International Color Consortium® (ICC). ISO 15076-1 specifies a profile format that is intended to provide a cross-platform profile format for the creation and interpretation of colour data. Central to ISO 15076-1 is the encoding of colour transforms between device colour encodings and profile connection spaces (PCSs) based upon D50 colorimetry with the CIE 1931 Standard 2-degree observer. For many workflows ISO 15076-1 has proven adequate for defining successful colour management systems. For other workflows ISO 15076-1 has been found to be limited in the flexibility of encoding colour transforms as well as defining means of profile colour connection that incorporate physical attributes of colour in addition to mere colour appearance.

The intent of this document is to provide a platform on which domain-specific specifications can be defined that make use of these extensions to the existing cross-platform profile format of ISO 15076-1. Thus, there is greater flexibility for defining colour transforms and PCSs to meet needs that cannot easily be met with ISO 15076-1. As such, it is not envisioned that all colour management systems that use this document will implement all the features or capabilities specified by this document. Specific requirements related to what is necessary to be implemented and supported relative to this document can be found in workflow domain specifications. Additionally, for some domain-specific workflows it is envisioned that there will be the need for simultaneous support for and interaction between ISO 15076-1 and profiles defined by this document.

It is assumed that the reader of this document has a good understanding of ISO 15076-1 as well as a good understanding of colour science and imaging, such as familiarity with CIE, ISO and IEC colour standards, general knowledge of device measurement and characterization, and familiarity with at least one operating system level colour management system.

The following subclauses introduce a few of the more significant differences from ISO 15076-1.

0.1 Extended profile connection spaces

0.1.1 ISO 15076-1 PCS encoding

In ISO 15076-1 PCS transform results are encoded relative to D50 with a 2-degree observer. If and when ISO 15076-1-based profiles are used in conjunction with this document, the PCS encoding specified in ISO 15076-1 are used with necessary conversions as needed.

0.1.2 Extended PCS encoding

PCS encoding is extended to allow PCS transform results to be relative to arbitrary illuminants and observers. Profile connection conditions (PCC) provided by either a profile or directly to the colour management module (CMM) can be applied to convert between different illuminants and observers. Additionally, a profile can define use of a spectrally-based PCS independent of the colorimetric-based PCS usage, with separate transform data between device encoding and the colorimetry and spectral PCS encodings.

0.2 Extended transform encoding

0.2.1 ISO 15076-1 transform encoding

ISO 15076-1 defines transforms using integer encoding in AToBx and BToAx tags. Floating point transform encoding can additionally be specified in optional DToBx and BToDx tags using multi-processing element tags.

Integer-based LUT tags have specific requirements for transform data and order.

The multi-processing element tag type allows a sequence of transform elements to be applied in order to transform between device encoding and PCS encoding. The processing elements consist of matrices, one-dimensional curve sets and *n*-dimensional lookup tables.

0.2.2 iccMAX extended transform encoding

Spectrally-based PCS transforms are encoded using DToBx/BToDx tags when a spectral PCS is used. Colorimetric-based PCS transforms are encoded in matrix/TRC based profiles or AToBx/BToAx tags. Additionally, AToBx/BToAx tag transforms can be encoded using the multi-processing element tag type.

The multi-processing element tag type is extended to provide greater flexibility as well as encoding brevity in defining transforms. Extended elements include a stack-based programmable transform calculator, single-segment curves, N-D lookup tables with integer encoding, colour appearance model (CAM) conversions, sparse matrix processing and tint arrays.

Multi-processing element-based tags are used to define PCC within a profile. The CMM applies these tags as needed to perform PCS conversions.

0.2.3 Late-binding processing elements

The multi-processing element tag type has been extended to allow for processing elements that provide late-binding of the observer and/or illuminant from the PCC utilized by the profile. Either spectral information inside select processing elements is converted to colorimetric data shortly before processing of colour transforms is to be performed, or spectral to colorimetric transforms are established for processing of colour transformations. This late-binding of spectral to colorimetric processing is based on the PCC utilized by the multi-processing element. The media-white point and illuminant colorimetry used for absolute/relative PCS processing is also adjusted based upon the combined profile/PCC relationships when late-binding processing elements are used.

0.3 Colour encoding space profiles

0.3.1 General

In ISO 15076-1, profiles define transforms that go from device to PCS. However, in some workflows the essential requirement is a method of defining what the data are rather than providing a transform that converts the data into a representation of colour.

0.3.2 Colour space encoding

This document establishes a ColorEncodingSpace profile class to define profiles that can be used when the content owner wishes to identify the colour encoding of digital colour content and does not wish to provide a colour transformation to be used in converting or adapting the digital colour content from the identified current colour space encoding to any other colour space encoding.

0.4 Multiplex connection space profiles

0.4.1 General

Generally, the data encoding sides of profile transforms are not used to connect profiles using ISO 15076-1. Connection of data encoding channels is only meaningful when the number, order and encoding of the data encoding channels are identical. However, in some workflows, flexibility in the number and order of the channels is desirable with a meaningful way of identifying the encoding of the channels.

0.4.2 Multiplex connection space encoding

This document defines an additional profile connection mechanism that allows multiplex connection space (MCS) channels to be connected. MCS connection provides a means of defining flexible connection between “device like” channels of profiles that are identified by name. Order and existence of channels is flexible with the ability for a profile to specify subset requirements on the MCS channels in the connected profile and default values specified for missing channels. The input profile class has been extended to have an optional tag that connects to an MCS. Additionally, MultiplexLink and MultiplexVisualization profile classes have been defined for MCS processing.

0.5 Bidirectional reflection distribution function (BRDF) and directional emission profiles

0.5.1 General

ISO 15076-1 assumes 0:45 measurement geometry for reflection prints and diffuse radiance of displays. However, in many conditions colour appearance can change due to changes in lighting or viewing angle. Such goniochromatic effects cannot be encoded or communicated using ISO 15076-1.

0.5.2 Bidirectional reflection distribution function encoding

This document provides the ability to encode bidirectional reflection distribution function (BRDF) information, as well as example surface information, that 3D rendering systems can use to emulate goniochromatic effects. In this case the BRDF information is provided directly to the 3D rendering system without extensive colour management system involvement. Additionally, BRDF information can be used to define and communicate goniochromatic properties of named colours.

0.5.3 Directional emission function encoding

This document provides the ability to encode directional emission information which can be used to define and communicate goniochromatic properties of colours by viewing angle and relative position on a display.

0.6 Rendering intents

In ISO 15076-1 four rendering intents are defined: perceptual, media-relative colorimetry, ICC-absolute colorimetry and saturation. For the purposes of supporting spectrally-based PCSs, the media-relative colorimetry and ICC-absolute colorimetry intents are referred to in this document as media-relative and ICC-absolute intents which apply to both colorimetric as well as spectral conditions.

Image technology colour management — Extensions to architecture, profile format and data structure

1 Scope

This document is based on ISO 15076-1, and describes an expanded profile specification and profile connections that permit greater flexibility and functionality than ISO 15076-1. All definitions and requirements in ISO 15076-1 are therefore in force unless otherwise specified by this document. This document defines minimum structural and operational requirements for writing and reading ICC profiles. Additional workflow requirements and restrictions are defined in domain-specific interoperability conformance specification (ICS) documents approved and registered by the ICC.

In this document, some ISO 15076-1 types have been removed, and others have been added. A colour management module (CMM) compatible with profiles conforming to this document will have backwards compatibility with profiles conforming to ISO 15076-1.

Where the name of a type in this document is the same as a type in ISO 15076-1, the type definition is based on the ISO 15076-1 definition. The exception is the definition of the MPE type, which has been expanded.

Where the extensions described in this document are not required in a particular workflow, ISO 15076-1 is used as the basis for colour management profiles and architectures.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 15076-1, *Image technology colour management — Architecture, profile format and data structure — Part 1: Based on ICC.1:2010*

ISO 17972-1, *Graphic technology — Colour data exchange format — Part 1: Relationship to CxF3 (CxF/X)*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1.1
profile connection conditions
PCC
information used to define illuminant, observer for PCS along with transforms to convert to and from custom colorimetry and standard D50 colorimetry for the standard 2° observer

3.1.2
profile connection space
PCS
colour space used to connect the source and destination profiles

Note 1 to entry: See ISO 15076-1:2010, Annex D for a full description.

3.2 Abbreviated terms

ANSI	American National Standards Institute
BRDF	bidirectional reflectance distribution function
CAM	colour appearance model
CIE	*Commission Internationale de l'eclairage* (International Commission on Illumination)
CLUT	colour lookup table (multi-dimensional)
CMM	colour management module
CMY	cyan, magenta, yellow
CMYK	cyan, magenta, yellow, key (black)
CRD	colour rendering dictionary
CRT	cathode-ray tube
EPS	encapsulated postscript
ICC	International Color Consortium
ICS	interoperability conformance specification
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
LCD	liquid crystal display
LUT	lookup table
MCS	multiplex connection space
PCC	profile connection conditions
PCS	profile connection space
RGB	red, green, blue
TIFF	tagged image file format
TRC	tone reproduction curve

4 Extended basic types

4.1 General

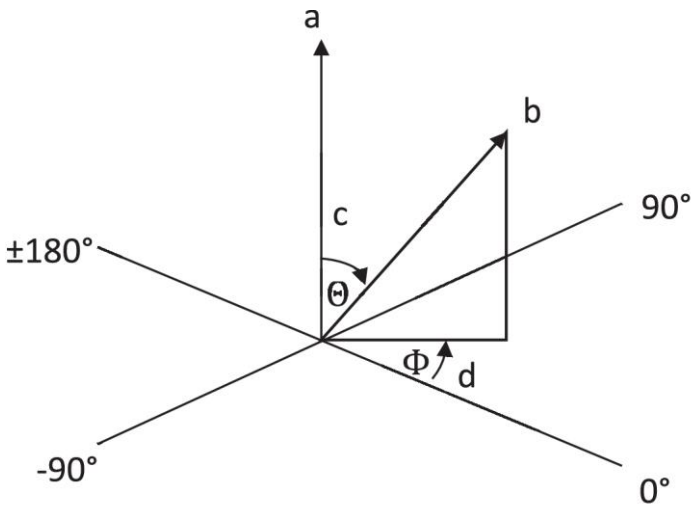
This document assumes the inclusion of all basic types listed in ISO 15076-1, with the exception of 7-bit ASCII. Only extended types in this document are listed below.

4.2 Extended basic type listing

4.2.1 azimuthNumber

An azimuthNumber corresponds to an azimuth angle for BRDF and directional transformations. It shall be encoded as a floating point number that is provided as input to a multiProcessElementType in any of the BRDF function tags (brdfAToB0Tag, brdfAToB1Tag, brdfAToB2Tag, brdfAToB3Tag, brdfBToA0Tag, brdfBToA1Tag, brdfBToA2Tag, brdfBToA3Tag, brdfBToD0Tag, brdfBToD1Tag, brdfBToD2Tag, brdfBToD3Tag, brdfDToB0Tag, brdfDToB1Tag, brdfDToB2Tag, brdfDToB3Tag, directionalAToB0Tag, directionalAToB1Tag, directionalAToB2Tag, directionalAToB3Tag, directionalBToA0Tag, directionalBToA1Tag, directionalBToA2Tag, directionalBToA3Tag, directionalBToD0Tag, directionalBToD1Tag, directionalBToD2Tag, directionalBToD3Tag, directionalDToB0Tag, directionalDToB1Tag, directionalDToB2Tag, directionalDToB3Tag).

The azimuthNumber encoding range shall be from 0,0 to 1,0, with 0,0 representing −180,0 degrees and 1,0 representing +180,0 degrees. Figure 1 shows the azimuth angle in relation to the normal and zenith angles.



Key

a surface normal
b lighting/viewer
c zenith angle
d azimuth angle

Figure 1 — Normal, zenith and azimuth angles

4.2.2 float16Number

A float16Number shall be a half-precision 16-bit floating-point number as specified in IEEE 754, excluding infinities and “not a number” (NaN) values.

NOTE 1 A 16-bit IEEE 754 floating-point number has a 5-bit exponent and a 10-bit mantissa.

NOTE 2 Although infinities and NaN values are not stored in the ICC profile, such values can occur as a result of CMM computations.

4.2.3 float64Number

A float64Number shall be a double-precision 64-bit floating-point number as specified in IEEE 754, excluding infinities, and "not a number" (NaN) values.

NOTE 1 A 64-bit IEEE 754 floating-point number has an 11-bit exponent and a 52-bit mantissa.

NOTE 2 Although infinities and NaN values are not stored in the ICC profile, such values can occur as a result of CMM computations.

4.2.4 horizontalNumber

A horizontalNumber corresponds to the horizontal relative position of a viewing field for directional transformations. It shall be encoded as a floating point number that is provided as input to a multiProcessElementType in any of the directional function tags (directionalAToB0Tag, directionalAToB1Tag, directionalAToB2Tag, directionalAToB3Tag, directionalBToA0Tag, directionalBToA1Tag, directionalBToA2Tag, directionalBToA3Tag, directionalBToD0Tag, directionalBToD1Tag, directionalBToD2Tag, directionalBToD3Tag, directionalDToB0Tag, directionalDToB1Tag, directionalDToB2Tag, directionalDToB3Tag).

The horizontalNumber encoding range shall be from −1,0 to 1,0 with −1,0 representing the leftmost position, 0,0 representing the center and 1,0 representing the rightmost position.

4.2.5 Sparse matrix encodings

4.2.5.1 General

Sparse matrices shall be encoded using compressed row order, which facilitates efficient multiplication of column vectors as well as the interpolation between sparse matrices. A sparse matrix shall be encoded as a variable structure with internal padding within a fixed size data block. The use of a fixed data block size allows for the efficient indexing of arrays of sparse matrices.

In addition to encoding the number of rows, number of columns and number of matrix data entries, the compressed row order encoding shall include three sub-arrays: a padded array of matrix entry data values, a padded array of matrix entry column identifiers, and an array of offsets to successive rows stored in the matrix data and column index arrays.

Successive offset values in the row start offset array shall be greater than or equal to preceding values. The number of matrix data entries associated with a row can therefore be found by subtracting the offset of the row by the offset of the succeeding row.

Successive matrix entry column index values associated with any single row shall be monotonically increasing.

Information about operations with sparse matrices can be found in Annex E.

Multiple sparse matrix encodings are permitted, but shall differ in the encoding of the matrix entry data values as follows:

— The sparseMatrixUInt8 encoding shall use uInt8Numbers to encode matrix data values (Table 1). The internal representation of the values 0 to 255 shall represent matrix values 0,0 to 1,0.

— The sparseMatrixUInt16 encoding shall use uInt16Numbers to encode matrix data values (Table 2). The internal representation of the values 0 to 65 535 shall represent matrix values 0,0 to 1,0.

— The sparseMatrixFloat16 encoding shall use float16Numbers to encode matrix values (Table 3).

— The sparseMatrixFloat32 encoding shall use float32Numbers to encode matrix values (Table 4).

Table 1 — sparseMatrixUInt8 encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…1	2	Rows (R)	uInt16Number
2…3	2	Columns (C)	uInt16Number
4…3+R*2	R*2	Row start offset array	uInt16Number[R]
4+R*2…5+R*2	2	Number of matrix entries (N)	uInt16Number
6+R*2…5+R*2 + N*2	N*2	Matrix entry column index array	uInt16Number[N]
6+R*2+N*2…O-1		Index padding, shall be 0	
O…O+N-1	N	Matrix entry data array	uInt8Number[N]
O+N…end		Data padding, shall be 0	

Table 2 — sparseMatrixUInt16 encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…1	2	Rows (R)	uInt16Number
2…3	2	Columns(C)	uInt16Number
4…3+R*2	R*2	Row start offset array	uInt16Number[R]
4+R*2…5+R*2	2	Number matrix entries (N)	uInt16Number
6+R*2…5+R*2 + N*2	N*2	Matrix entry column indices	uInt16Number[N]
6+R*2+N*2…O-1		Index padding, shall be 0	
O…O+N*2-1	N*2	Matrix entry data values	uInt16Number[N]
O+N*2…end		Data padding, shall be 0	

Table 3 — sparseMatrixFloat16 encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…1	2	Rows (R)	uInt16Number
2…3	2	Columns (C)	uInt16Number
4…3+R*2	R*2	Row start offset array	uInt16Number[R]
4+R*2…5+R*2	2	Number matrix entries (N)	uInt16Number
6+R*2…5+R*2 + N*2	N*2	Matrix entry column indices	uInt16Number[N]
6+R*2+N*2…O-1		Index padding, shall be 0	
O…O+N*2-1	N*2	Matrix entry data values	float16Number[N]
O+N*2…end		Data padding, shall be 0	

Table 4 — sparseMatrixFloat32 encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…1	2	Rows (R)	uInt16Number
2…3	2	Columns (C)	uInt16Number
4…3+R*2	R*2	Row start offset array	uInt16Number[R]
4+R*2…5+R*2	2	Number matrix entries (N)	uInt16Number

Table 4 *(continued)*

Byte position	Field length (bytes)	Content	Encoded as...
6+R*2…5+R*2 + N*2	N*2	Matrix entry column indices	uInt16Number[N]
6+R*2+N*2…O-1		Index padding, shall be 0	
O…O+N*4-1	N*4	Matrix entry data values	float32Number[N]
O+N*4…end		Data padding, shall be 0	

4.2.5.2 Compact padding

When sparse matrices are encoded in a profile they shall be compacted so that the index and data padding result in the matrix entry data values and the end of the sparse matrix being aligned on a 4 byte boundary.

NOTE Compact padding can result in variability in the size of individual sparse matrices in a sparse matrix array or sparse matrix LUT.

4.2.5.3 Fixed block size padding

When a colour space uses sparse matrix encoding it is useful for the internal encoding to use a fixed block size determined by the number of samples associated with the colour space.

NOTE One method of internally encoding Sparse Matrices within a CMM adjusts the index and data padding to allow the number of matrix entries to vary without the size of the encoded data block size changing. The block size therefore determines a fixed upper limit to the number of entries that can be encoded.

When fixed block size padding is used, the maximum number of matrix entries (M) that can be encoded for each of the sparse matrix encodings is determined by the fixed data block size (B) used to store the sparse matrix, the number of rows (R) and the byte size (S) of a matrix entry data value as shown in Formula (1):

$$M = \text{floor}\ ((B - 8 - 2*R - (S - 1)]/(S + 2)) \quad (1)$$

When fixed block size padding is used for each of the sparse matrix encodings in Table 1 to Table 4 the offset of the matrix entry data array (O) is determined by the number of rows (R) and the maximum number of matrix entries (M), as well as the byte size (S) of a matrix entry data value as shown in Formula (2):

$$O = \text{floor}\ ((8 + 2*R + 2*M + (\text{S} - 1))/\text{S})\ *\text{S} \quad (2)$$

4.2.6 sparseMatrixEncodingType

When encoding sparse matrices the exact data encoding type used shall be specified using a sparseMatrixEncodingType parameter. Where used, values for a sparseMatrixEncodingType parameter shall be encoded as defined in Table 5.

Table 5 — sparseMatrixEncodingType selection of sparse matrix encoding in SparseMatrixLut

sparseMatrixEncodingType	Sparse matrix encoding
1	sparsematrixUInt8
2	sparseMatrixUInt16
3	sparseMatrixFloat16
4	sparseMatrixFloat32

4.2.7 spectralRange

The spectralRange data type shall be used to specify spectral ranges. This data type shall be made up of two float16Number values and a uInt16Number value that define the starting wavelength, ending wavelength and total number of steps in the range. The encoding of a spectralRange data type is shown in Table 6.

Table 6 — spectralRange encoding

Byte position	Field length (bytes)	Field contents	Encoded as...
0 to 1	2	Start wavelength	float16Number
2 to 3	2	End wavelength	float16Number
4 to 5	2	Steps in wavelength range	uInt16Number

4.2.8 tintArray

A tintArray defines the relationship between a single input value (tint) and multiple output values. A tintArray is used by both the multi processing element tintArrayElement (11.2.14) as well as the namedColorStructure (12.2.5). Any of the numeric array types (e.g. uInt8ArrayType) are permitted in a tintArray. The relationship between N tints (each made up of M samples per tint) in a tintArray is depicted in Table 7.

Table 7 — tintArray sample index assignments

Tint index	Sample 1	Sample 2	...	Sample M-1	Sample M
0	0	1	...	M-2	M-1
1	M	M+1	...	2M-2	2M-1
...	...	...	...	...	...
N-2	(N-2)M	(N-2)M+1	...	(N-1)M-2	(N-1)M-1
N-1	(N-1)M	(N-1)M+1	...	NM-2	NM-1

A tint array shall have N × M entries in the array.

NOTE Assignment of tint values ranging from 0,0 to 1,0 to tint indices can vary and is specific to the use case in which a tint array is used.

4.2.9 valueEncodingType

When encoding values in sampled curves and colour lookup tables (CLUTs) the exact data encoding type used shall be specified using a valueEncodingType parameter. Values for a valueEncodingType parameter shall be as defined in Table 8. Encoded values for the float32Number and float16Number types shall represent the actual encoding number. Encoded values for the uInt16Number and uInt8Number types shall represent an encoding between the range of 0,0 to 1,0.

Table 8 — valueEncodingType values

valueType	value Encoding
0	float32Number
1	float16Number
2	uInt16Number
3	uInt8Number

Values for a valueEncodingType may be encoded as either a uInt16Number or a uInt32Number.

4.2.10 verticalNumber

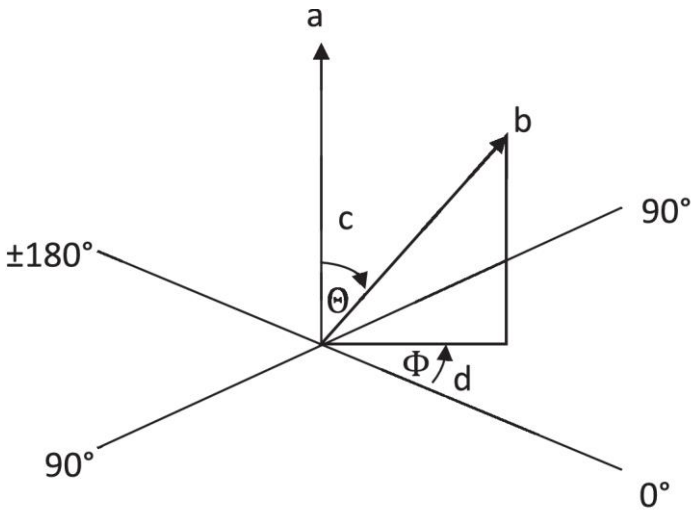
A verticalNumber corresponds to the vertical relative position of a viewing field for directional transformations. For example, the viewing field of a display represents the physical limits of the display. A verticalNumber shall be encoded as a floating point number that is provided as input to a multiProcessElementType in any of the directional function tags (directionalAToB0Tag, directionalAToB1Tag, directionalAToB2Tag, directionalAToB3Tag, directionalBToA0Tag, directionalBToA1Tag, directionalBToA2Tag, directionalBToA3Tag, directionalBToD0Tag, directionalBToD1Tag, directionalBToD2Tag, directionalBToD3Tag, directionalDToB0Tag, directionalDToB1Tag, directionalDToB2Tag, directionalDToB3Tag).

The verticalNumber encoding range shall be from −1,0 to 1,0 with −1,0 representing the topmost position, 0,0 representing the center and 1,0 representing the bottommost position.

4.2.11 zenithNumber

A zenithNumber can be used to specify to a zenith angle θ for BRDF and directional transformations or to define geometry of measurement. It shall be encoded as a floating point number that is provided as input to a multiProcessElementType in any of the BRDF function tags (brdfAToB0Tag, brdfAToB1Tag, brdfAToB2Tag, brdfAToB3Tag, brdfBToA0Tag, brdfBToA1Tag, brdfBToA2Tag, brdfBToA3Tag, brdfBToD0Tag, brdfBToD1Tag, brdfBToD2Tag, brdfBToD3Tag, brdfDToB0Tag, brdfDToB1Tag, brdfDToB2Tag, brdfDToB3Tag, directionalAToB0Tag, directionalAToB1Tag, directionalAToB2Tag, directionalAToB3Tag, directionalBToA0Tag, directionalBToA1Tag, directionalBToA2Tag, directionalBToA3Tag, directionalBToD0Tag, directionalBToD1Tag, directionalBToD2Tag, directionalBToD3Tag, directionaDToB0Tag, directionalDToB1Tag, directionalDToB2Tag, directionalDToB3Tag).

The zenithNumber encoding range shall be from 0,0 to 1,0, with 0,0 representing 0,0 degrees and 1,0 representing 90,0 degrees. Figure 2 shows the zenith angle in relation to the normal and azimuth angles.



Key

a surface normal
b lighting/viewer
c zenith angle
d azimuth angle

Figure 2 — Normal, zenith and azimuth angles

5 Conformance

Any colour management system, application, utility or device driver that claims full conformance with this document shall have the ability to read the profiles as they are defined in this document, including all specified tags and types. Any profile-generating software and/or hardware that claims full conformance with this document shall have the ability to create profiles as they are defined in this document, including all specified tags and types.

In addition, software or hardware may claim partial conformance with this document by reading and/or creating profiles that contain a subset of the tags and types listed in a separate domain-specific ICS approved and registered by the ICC.

Software conforming to this document shall use the ICC profiles generated in accordance with this document, or with an approved ICS.

Unambiguous operation of profiles in workflow scenarios identified in an ICS is provided by the use of CMM processing control options. For more information about workflow scenarios and CMM processing options see Annex K.

A colour management system, profile-generating software and/or hardware application, utility or device driver that claims full or partial conformance with this document shall also conform with all definitions and requirements in ISO 15076-1 unless otherwise specified by this document, and shall register all signatures for CMM type, device manufacturer, device model, profile tags and profile tag types with the ICC to ensure that all profile data are uniquely defined. Domain-specific specification documents (ICSs) shall also be registered. The registration authority for all of these is the ICC Technical Secretary.

NOTE See the ICC website (http://www.color.org) for contact information.

6 Expanded PCSs, rendering intents and device encoding

6.1 General considerations

This document describes extensions to profiles and their connections that permit a greater flexibility and functionality than ISO 15076-1. Where such extensions are not needed in a particular workflow, users should continue to use ISO 15076-1 as the basis for colour management profiles and architectures.

6.2 Extensions to device colour encoding

The number of channels associated with a colour space is determined from the colour space signature. Extensions that allow for processing elements to utilize up to 65 535 channels require uniquely defined colour space signatures associated with up to 65 535 channels. This is accomplished by extending the signature definition to use the binary representation of the actual number of channels within the 32-bit signature. The 16 most significant bits (corresponding to textual digits) shall match an extended colour space type (signature identifier), and the 16 least significant bits shall define a binary representation of the number of channels (signature channels). This results in a colour space signature containing 32 bits that cannot be represented as four text characters.

A six-character text string shall be used for cases when a textual representation of these extended colour space signatures is desired with the first two characters corresponding to the signature identifier and the last four digits corresponding to a hexadecimal representation of the signature channels. (Thus: “nc0014” corresponds to the 32-bit hexadecimal extended colour space signature encoding (6e630014h) used in the profile which represents N-channel data with twenty device channels.) Having a six-character text representation of an extended colour space signature is for convenience purposes only for describing the signature value (as colour signatures are only encoded as binary 32-bit values within profiles). Extended N channel device data colour space signature encoding is provided in Table 9.

Table 9 — Extended data colour space signatures

Spectral colour space type	Signature identifier	Signature channels	Combined hex encoding	Signature representation
N channel device data	'nc' (6e63h)	1 … 65 535 (0001h … FFFFh)	6e630001h … 6e63FFFFh	"nc0001" … "ncFFFF"
None (PCS defined by PCS header field)	0	0	00000000h	0

6.3 Extensions to PCSs

6.3.1 General

ISO 15076-1 defines PCSs in terms of D50 colorimetry with the CIE 1931 standard 2-degree observer. The resulting transform data are therefore encoded relative to this illuminant and observer. Use of the chromatic adaptation tag in ISO 15076-1 provides a means of converting encoded PCS transform results for the media white point to and from actual illuminant/observer colorimetry. A CMM that is in conformance with this document shall always provide transforms to and from D50/2-degree colorimetry that can be used when connecting profiles. Such transforms shall make use of the profile connection condition tags defined in this document rather than the chromatic adaptation tag defined in ISO 15076-1. This document extends profile connection to allow direct encoding of colour data in transforms with arbitrary illuminants and observers. This results in the CMM performing conversions between PCS observer and illuminant colorimetry as needed using profile connection conditions (PCC). The PCC defines information about the observer and illuminant as well as transforms between custom observer/illuminant conditions and the standard D50/2-degree observer colorimetry. Where PCC are used to connect profiles, the CMM shall either use the PCC from within a profile or a PCC provided by the CMM. Profiles based on colorimetry using an illuminant other than CIE D50 or a colorimetric observer other than the CIE 1931 standard observer (known as the 2-degree observer) shall encode profile connection condition tags (see 6.3.2).

Additional PCSs can be specified that are based upon spectral representation of colour rather than colorimetry. A spectral PCS use is defined and encoded separately from colorimetric PCS use within a profile. Therefore both colorimetric and spectral PCS transforms can simultaneously exist in a profile. Profile connection condition tags shall also be encoded within a profile whenever a spectral PCS is used within a profile.

Additional tags can be provided in a profile that can define surface characteristics such as gloss, with relationships between lighting and viewing angle defined. These tags provide an additional connection interface separate from the PCS. These BRDF transforms provide information for three-dimensional rendering systems that are capable of rendering light interactions based upon the surface characteristics. The measurement angle tag also allows the profile generator to identify the measurement geometry used to generate the PCS data in the profile.

Lastly, a multiplex-based connection method is introduced that allows profiles to be connected using named multiplex channels. This can be considered as an extension of device-channel-based connection with flexibility and channel matching rules.

6.3.2 Profile connection conditions

The standard PCS shall be defined to use the 2-degree observer under D50 illumination. Whenever either a non-standard PCS or a spectral PCS is used, three tags shall be encoded in a profile and shall be used to define default PCC for a profile. In cases of spectral PCS use or late-binding colorimetric processing elements, a CMM may optionally be provided or use external PCC, thus overriding the defaults provided by the profile.

The spectralViewingConditions tag shall define the spectral power distribution of the illuminant, colour matching functions (CMFs) of the observer and the lighting levels of the surround. If the

spectralViewingConditions are different from the standard PCS viewing conditions then two tags are used to convert between custom and standard connection conditions (see 9.2.105).

The customToStandardPCC tag shall define a transform that converts from the custom viewing condition colorimetry to standard viewing condition colorimetry (see 9.2.56).

The standardToCustomPCC tag shall define a transform that converts from standard viewing condition colorimetry to the custom viewing condition (see 9.2.107).

Apart from the spectral tables, all other tables, both forward and inverse, shall convert colours between the colour space of the PCS and coding values of the colour reproduction device, taking into account the reference and actual viewing conditions.

6.3.3 Spectral PCSs

6.3.3.1 General

This document allows for a spectrally-based PCS to be defined for DToBx/BToDx tags using a spectralPCS signature field in the profile header. The use of DToBx/BToDx tags for colorimetric processing has been deprecated in this document. Additional header fields are also added to allow specification of spectral information used by the profile.

A distinction is made between self-emitting colours and non self-emitting colours, here referred to as luminous colours and object colours. Luminous colours are characterized by their emission spectra, whereas for object colours reflectance or transmission spectra are used. These three types of spectra are referred to as object characterization spectra.

Reflectance spectra are specified in relation to the perfect reflector whereas transmission spectra are related to a perfect transmitter. Therefore, both types of spectra can be seen as relative data. For emission spectra, Y tristimulus values correspond to luminance values, and hence these are regarded as absolute data.

Different types of spectral data can be defined. In normal circumstances, only reflectance, transmission or emission spectra are used but in other circumstances additional data shall be provided according to the processing to be carried out. To represent bi-spectral data, a Donaldson matrix is used and is specified by bi-spectral data fields in the header.

6.3.3.2 Encoding spectral data

To define the use of a spectrally-based PCS, one of the spectral colour space signatures in Table 10 shall be used to encode the colour space implied by the spectralPCS field of the profile header. These colour space signatures define both the colour space type and the number of channels associated with the colour space. Therefore, the number of signature channels associated with the spectralPCS colour space signature shall match the number of channels indicated by the steps field(s) of the corresponding spectralRange structures in the profile header.

Table 10— Spectral colour space signatures

Spectral colour space type	Signature identifier	Signature channels	Combined hex encoding	Signature representation
None (PCS defined by PCS header field)	0	0	00000000h	0
Reflectance spectra with N channels	’rs’ (7273h)	1 … 65 535 (0001h … FFFFh)	72730001h … 7273FFFFh	“rs0001” … “rsFFFF”
Transmission spectra with N channels	’ts’ (7473h)	1 … 65 535 (0001h … FFFFh)	74730001h … 7473FFFFh	“ts0001” … “tsFFFF”

Table 10 *(continued)*

Radiant (emission) spectra with N channels	'es' (6573h)	1 … 65 535 (0001h … FFFFh)	65730001h … 6573FFFFh	"es0001" … "esFFFF"
Bi-spectral reflectance spectra with N total channels	'bs' (6273h)	1 to 65 535 (0001h … FFFFh)	62730001h … 6273FFFFh	"bs0001" … "bsFFFF"
Bi-spectral reflectance using sparse matrix with N equivalent output channels	'sm' (736d)	1 to 65 535 (0001h … FFFFh)	736D0001h … 736DFFFFh	"sm0001" … "smFFFF"

NOTE Spectral colour space signatures use the same 32-bit binary encoding mechanism as N colour device data signatures (see 6.2.1), with each having a six-character signature representation.

Spectra are normally represented according to their canonical basis, i.e. the spectrum is sampled uniformly along the wavelength axis. The wavelength range is represented by a start wavelength (S), end wavelength (E) and number of steps (n). The wavelength interval between steps (I) is then given by Formula (3).

$$I = (E - S)/(n - 1) \tag{3}$$

6.3.3.3 Spectral consistency of tags in profiles

Apart from measurement files embedded in profiles, spectral data in all other tags are assumed to be sampled at uniform intervals, with a given start, end wavelength and number of steps. The dimensions and range of the spectra in the different tags have to be defined consistently. For normal spectra, it means that the spectral dimension of object characterization spectra have to be the same. If the object characterization spectra are defined by the Donaldson matrix, the Donaldson matrix has to be an nXm matrix with m defined internally in the corresponding colour table. For multiple spectra, suppose k, the object characterization spectra are a column vector with length kXn in the colour tables, but after processing also n dimensional spectra are obtained.

The spectral type of the object characterization spectra is defined in the profile header.

Tags containing measurement files are seen as separate entities and hence the previous conditions do not have to be fulfilled.

6.3.3.4 Spectral fluorescence connection spaces

The characterization of the interaction of light with a diffuse surface can be accomplished using a Donaldson matrix. The multiplication of such a matrix by a vector representing the illumination results in a vector representing the light reflected off the surface. Columns of a Donaldson matrix correspond to incident wavelengths of light and rows of a Donaldson matrix correspond to reflected wavelengths of light. Diagonal entries (where incident and reflected wavelengths are the same) correspond to spectral reflectance. Off diagonal entries (below the diagonal) represent the contribution of a change in the reflected light's wavelength (typically due to fluorescence). Fluorescence occurs when light is absorbed and then re-emitted at a longer wavelength. Using Donaldson matrices to represent colours in an ICC profile allows for a more complete description of colour to be encoded than using only spectral reflectance or simple colorimetry. Examples of various spectral calculations can be found in Annex A.

However, directly encoding a Donaldson matrix in a profile can be inefficient. For example, if 41 wavelengths are used to represent an illuminant (300 nm to 700 nm with a 10 nm interval) and 31 wavelengths are used to represent the reflected light (400 nm to 700 nm with a 10 nm interval) then a corresponding Donaldson matrix encodes 31 × 41 = 1 271 entries. However, most of the off diagonal

entries are zero with the only entries with non-zero data for fluorescent terms. By using a sparse matrix encoding (which only encodes non-zero entries of a matrix), large profile files can be avoided while simultaneously reducing processing overhead because fewer computations need to be performed. This results in a compression of information and requires that interpolation and application of the matrices be carried out correctly.

When using Donaldson matrices to represent colour values in a colour LUT, intermediate matrices shall be determined (using interpolation) to establish the intermediate "colour". However, since a Donaldson matrix is associated with a single combination of device code values, interpolation between matrices associated with different device code values needs to be performed to estimate a Donaldson matrix for intermediate device code values.

The use of a single sparse matrix LUT encoding can be used in two contexts. The first is in a tag containing a single dimensional array of sparse matrices representing different tint values of a single colour (used by NamedColors). The second is a multi-dimensional table of sparse matrices in the context of a multi-process element which is useful for characterizing a device using Donaldson matrices for each possible input combination.

Normally colour lookup tables (CLUTs) define multiple output samples for each input coordinate in lookup table. A sparse matrix expands the meaning of the output colour samples being passed around in a CMM. When sparse matrices are implied by a colour space the array of colour samples should instead be interpreted as using a sparse matrix encoding. The number of samples defined in a sparse matrix colour space establishes the upper limit to the number of matrix entries that can be encoded. A compressed row order encoding of sparse matrices is utilized. This encoding format allows for efficient interpolation of matrices as well as efficient multiplication of vectors by sparse matrices.

6.3.4 BRDF connection

None of the transforms defined by ISO 15076-1 support BRDF as an input to a transform. It is possible for a CMM to transform BRDF values to connection space values that are needed.

Extended information about using BRDF with tags encoded in profiles defined by this document can be found in Annex G.

6.3.5 Directional viewing connection

None of the transforms defined by ISO 15076-1 support directional or positional information as input to a transform. It is possible for a CMM to transform both directional angles and relative positional information to determine connection space values that are needed in order to compute device values. Reverse transforms are also possible to determine device values that achieve connection space values.

Extended information about using directional tags encoded in profiles defined by this document can be found in Annex H.

6.4 Multiplex connection spaces

6.4.1 General

This document allows for a connection space defined by multiplex channel identification for AToM0/MToA0/MToB0/MToS0 tags using a multiplex connection space (MCS) signature field in the profile header. If this field is zero, then multiplex connection is not defined.

MCS connection is performed by passing values for multiplex channels between profiles that have identical multiplex channel type values (defined in the multiplexTypeArrayTag of both profiles). The MCS subset requirements shall be met before profiles can be linked (see 7.2.13). Once these requirements are met the channels with a multiplex channel type in the source profile that are not in the destination profile are ignored, and channels with multiplex channel types in the destination profile that are not in the source profile are processed with the multiplex channel value defined for the channel in the

multiplexDefaultValuesTag or zero if this tag is not present. Figures 3 and 4 show examples of MCS profile connection.

Figure 3 — Workflow connecting a MultiplexIdentification (MID)/input profile, a MultiplexVisualization (MVIS) profile and an output profile

Figure 4 — Workflow connecting a MultiplexIdentification (MID)/input profile with a MultiplexLink (MLNK) profile

Examples of MCS workflows and connection scenarios can be found in Annex I.

6.4.2 MCS signature encoding

To define the use of MCS connection with multiplex channel identification the signatures in Table 11 shall be used to encode the MCS field of the profile header. These colour space signatures define both the colour space type and the number of channels associated with the colour space.

Table 11 —Multiplex colour space signatures

Multiplex colour space type	Signature identifier	Signature channels	Combined hex encoding	Signature representation
None (no MCS is used)	0	0	0	0
Multiplex channel values with N channels	'mc' (6d63h)	1 … 65 535 (0001h … FFFFh)	6d630001h … 6d63FFFFh	"mc0001" … "mcFFFF"

NOTE Multiplex colour space signatures use the same 32-bit binary encoding mechanism as N colour device data signatures (see 6.2.1), with each having a six-character signature representation.

6.5 Colour encoding space profiles

This document provides a means of defining a colour encoding relative to a named reference encoding. Therefore it defines what the data are (not how they are transformed). The reference shall be a registered colour encoding in the ICC three-component colour encoding registry.

Because the transform is not defined by the profile, the CMM is responsible for determining what transform to use.

The intent of ColorEncodingSpace profiles is to allow for profile files that have a minimum data structure that can be embedded in images with clear, concise and non-redundant (canonical) information relative to a "named" reference provided to the CMM for determining the actual transforms to apply.

Minimally, a ColorEncodingSpace profile shall have a header, a tag directory and a referenceNameTag (see 9.2.103) which defines the named reference encoding associated with the colour encoding space.

Various modes of operation are defined for ColorEncodingSpace profiles in 8.7. Brief guidelines for transform determination by the CMM can be found in Annex J.

NOTE It is envisioned that the set of required and optional named colour encoding spaces will be defined by ICSs external to this document.

7 Profile requirements

7.1 General

An ICC profile shall include the following elements, in the order shown, as a single file:

a) a 128-byte profile header as defined in 7.2;

b) a profile tag table as defined in 7.3;

c) profile tagged element data as defined in 7.4.

This is illustrated in Figure 5.

The required tags for each profile type are tabulated in Clause 8. The definition of all publicly available tags and their signatures is contained in Clause 9 along with the permitted tag types for each tag. Tag types are defined in Clause 10. Extended ICC profiles may support tags defined as using either the multiProcessElementsType or the tagStructType. Multiple processing elements are defined in Clause 11. Tag structure types are defined in Clause 12. Tag array types are defined in Clause 13.

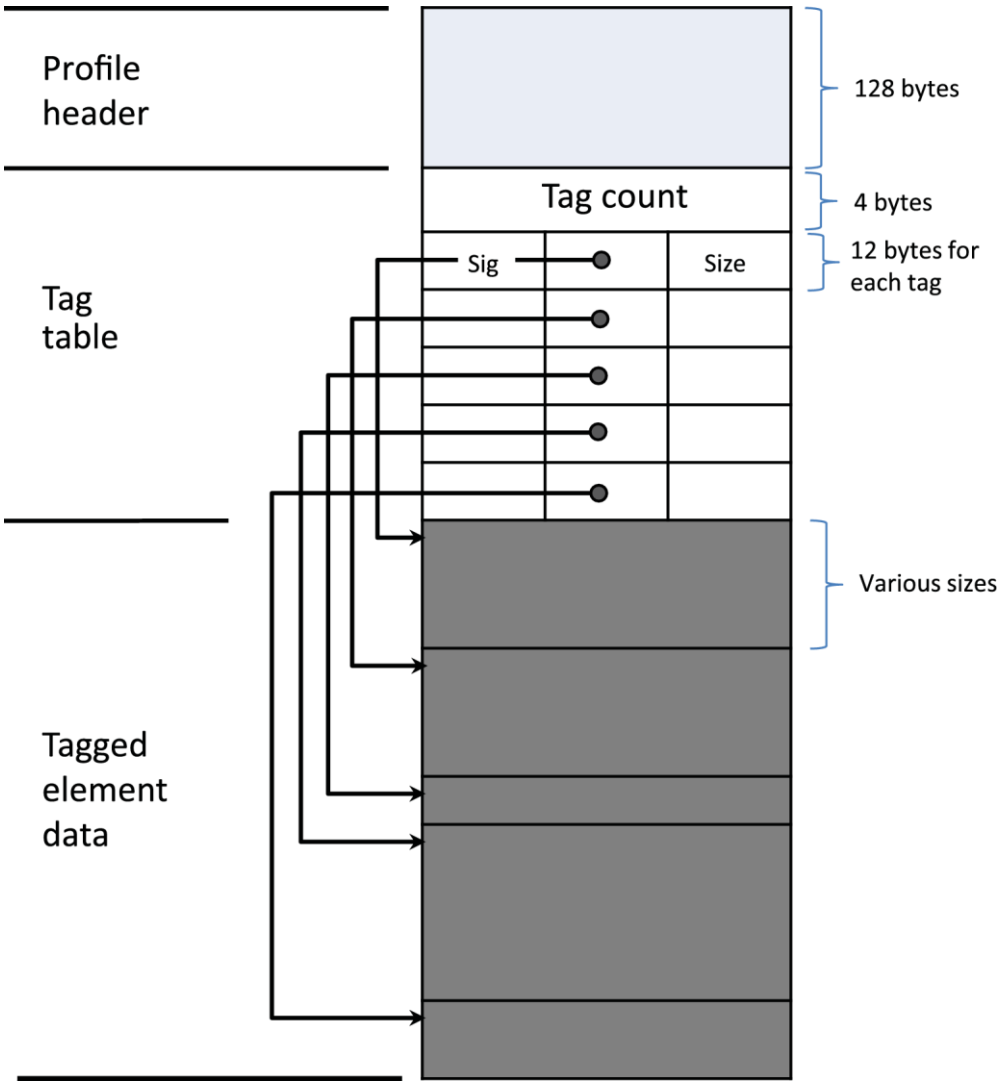


Figure 5 — Profile structure

Within the profile structure:

a) all profile data shall be encoded as big-endian;

b) the first set of tagged element data shall immediately follow the tag table;

c) all tagged element data, including the last, shall be padded by no more than three following pad bytes to reach a 4-byte boundary;

d) all pad bytes shall be NULL (ISO 646, character 0/0).

NOTE 1 This implies that the length of the file is a multiple of four.

NOTE 2 The above restrictions result in two key benefits. First, the likelihood of two profiles which contain the same tag data, yet have different checksum values, is reduced. Second, all profiles are reduced to a minimum size.

7.2 Profile header

7.2.1 General

ISO 15076-1 defines a single header specification for defining ICC profiles. ISO 20677 extends the use of the ICC header in two ways. For spectral PCS support additional entries have been defined. For ColorEncodingSpace profile device class profiles a minimal subset of header field entries is defined.

7.2.2 Extended profile header field definitions

The encoding of the profile header with spectral PCS support shall be as shown in Table 12.

Table 12 — Profile header fields

Byte position	Field length (bytes)	Field contents	Encoded as...
0 to 3	4	Profile size	uInt32Number; See 7.2.4
4 to 7	4	Preferred CMM type	4-byte signature; see 7.2.5
8 to 11	4	Profile version and sub-version number	uInt32Number; see 7.2.6
12 to 15	4	Profile/device class	4-byte signature; see 7.2.7
16 to 19	4	Colour space of data (possibly a derived space)	4-byte signature; see 7.2.8
20 to 23	4	PCS	4-byte signature; see 7.2.9
24 to 35	12	Date and time this profile was first created	dateTimeNumber; see 7.2.10
36 to 39	4	‘acsp’ (61637370h) profile file signature	4-byte signature; see 7.2.11
40 to 43	4	Primary platform signature	4-byte signature; see 7.2.12
44 to 47	4	Profile flags to indicate various options for the CMM such as distributed processing and caching options	uInt32Number; see 7.2.13
48 to 51	4	Device manufacturer of the device for which this profile is created	4-byte signature; see 7.2.14
52 to 55	4	Device model of the device for which this profile is created	4-byte signature; see 7.2.15
56 to 63	8	Device attributes unique to the particular device setup such as media type	uInt64Number see 7.2.16
64 to 67	4	Rendering Intent	uInt32Number; see 7.2.17
68 to 79	12	The nCIEXYZ values of the PCS illuminant, computed with the PCS observer	XYZNumber; see 7.2.18
80 to 83	4	Profile creator signature	4-byte signature; see 7.2.19
84 to 99	16	Profile ID	uInt64Number[2]; see 7.2.20
100 to 103	4	Spectral PCS	4-byte signature; see 7.2.21
104 to 109	6	Spectral PCS wavelength range	spectralRange; See 7.2.22

110 to 115	6	Bi-spectral PCS wavelength range	spectralRange; see 7.2.23
116 to 119	4	MCS signature	uInt32Number; see 7.2.24
120 to 123	4	Profile/device sub-class	4-byte signature; see 7.2.25
124 to 127	4	Reserved bytes shall be zero (00h)	uInt32Number; see 7.2.26

A spectral-only profile shall be defined by setting the PCS header field to zero, setting the spectral PCS field to the desired spectral colour space, and only providing spectral DToBx and BToDx tables in the profile.

7.2.3 ColourEncodingSpace class profile header field definitions

A ColourEncodingSpace class profile shall use only the ICC header fields defined in Table 13 with the other fields either set according to ISO 15076-1 or optionally zero filled.

Table 13 — ColourEncodingSpace profile header fields

Byte position	Field length bytes	Field contents	Encoded as…
0 to 3	4	Profile size	uInt32Number
4 to 7	4	Reserved bytes shall be zero (00h)	
8 to 11	4	Profile version number	5,0 or higher
12 to 15	4	'cenc' (63656E63h) profile device class	
16 to 19	4	Colour space of data (possibly a derived space)	‘RGB ’ (52474220h) or ‘YCC ‘(59434320h)
20 to 35	16	Reserved bytes shall be zero (00h)	
36 to 39	4	‘acsp’ (61637370h) profile file signature	See 7.2.11
40 to 127	88	Reserved bytes shall be zero (00h)	

7.2.4 Profile size field (bytes 0 to 3)

The value in the profile size field shall be the exact size obtained by combining the profile header, the tag table and the tagged element data, including any pad bytes. It shall be encoded as a uInt32Number.

7.2.5 Preferred CMM type field (bytes 4 to 7)

This field may be used to identify the preferred CMM to be used. If used, it shall match a CMM type signature registered in the ICC CMM registry. If no preferred CMM is identified, this field shall be zero (00000000h).

7.2.6 Profile version and sub-version field (bytes 8 to 11)

The profile version with which the profile conforms shall be encoded as binary-coded decimal in the profile version field. The first byte (byte 8) shall identify the major version and byte 9 shall identify the minor version and bug fix version in each 4-bit half of the byte. Bytes 10 and 11 shall be used to identify the profile sub-class version where byte 10 shall be used to identify the sub-class major version and byte 11 shall be used to identify the sub-class minor version. When a sub-class is not associated with a profile (when the Pprofile/device sub-class field is zero) then bytes 10 and 11 shall be zero. The major and minor versions are set by the International Color Consortium (ICC). Profile sub-class versions shall be

established by profile sub-class specification documents. The profile version and sub-version number consistent with this specification is "5.0.0.0" (encoded as 05000000h).

NOTE A major version number change occurs only when changes made to this document require that both CMMs and profile generating software be upgraded in order to correctly produce or use profiles conforming to the revised specification. A minor version number change occurs when profiles conforming to the revised specification can be processed by existing CMMs. For example, adding a new required tag would necessitate a major revision to the specification, whereas adding an optional tag would only require a minor revision.

7.2.7 Profile/device class field (bytes 12 to15)

This field shall contain one of the profile class signatures shown in Table 14.

There are three basic classes of device profiles: Input, Display and Output profiles. In addition to the three basic device profile classes, eight additional colour processing profiles are defined. These profiles provide a standard implementation for use by the CMM in general colour processing, or for the convenience of CMMs which may use these types to store calculated transforms. These eight additional profile classes are DeviceLink, ColorSpace, ColorEncodingSpace, Abstract, NamedColor, MultiplexIdentification, MultiplexLink and MultiplexVisualization.

Table 14 — Profile classes

Profile class	Signature	Hexidecimal encoding
Input Device profile	‘scnr’	73636E72h
Display Device profile	‘mntr’	6D6E7472h
Output Device profile	‘prtr’	70727472h
DeviceLink profile	‘link’	6C696E6Bh
ColorSpace profile	‘spac’	73706163h
Abstract profile	‘abst’	61627374h
NamedColor profile	‘nmcl’	6E6D636Ch
ColorEncodingSpace profile	‘cenc‘	63656E63h
MultiplexIdentification profile	‘mid ’	6D696420h
MultiplexLink profile	‘mlnk’	6d6c6e6bh
MultiplexVisualization profile	‘mvis’	6d766973h

7.2.8 Data colour space field (Bytes 16 to 20)

This field shall contain the signature of the data colour space expected on the A side of the profile transforms.

The names and signatures of the permitted data colour spaces shall be as shown in Table 15.

Table 15 — Data colour space signatures

Colour space type	Signature	Hexidecimal encoding
nCIEXYZ or PCSXYZ[a]	‘XYZ ’	58595A20h
CIELAB or PCSLAB[b]	‘Lab ’	4C616220h
CIELUV	‘Luv ’	4C757620h
YCbCr	‘YCbr’	59436272h
CIEYxy	‘Yxy ’	59787920h

LMS	'LMS '	4C4D5320h
RGB	'RGB '	52474220h
Gray	'GRAY'	47524159h
HSV	'HSV '	48535620h
HLS	'HLS '	484C5320h
CMYK	'CMYK'	434D594Bh
CMY	'CMY '	434D5920h
2 colour	'2CLR'	32434C52h
3 colour (other than those listed above)	'3CLR'	33434C52h
4 colour (other than CMYK)	'4CLR'	34434C52h
5 colour	'5CLR'	35434C52h
6 colour	'6CLR'	36434C52h
7 colour	'7CLR'	37434C52h
8 colour	'8CLR'	38434C52h
9 colour	'9CLR'	39434C52h
10 colour	'ACLR'	41434C52h
11 colour	'BCLR'	42434C52h
12 colour	'CCLR'	43434C52h
13 colour	'DCLR'	44434C52h
14 colour	'ECLR'	45434C52h
15 colour	'FCLR'	46434C52h
N channel device data	*Represented as* "nc0001" – "ncFFFF"	6e630001h – 6e63FFFFh
None	0	00000000h

[a] The signature 'XYZ ' refers to nCIEXYZ or PCSXYZ, depending upon the context.
[b] The signature 'Lab' refers to CIELAB or PCSLAB, depending upon the context.

NOTE Extended "N channel device data" signatures use a 32-bit binary encoding (see 6.2.1) with a six-character signature representation.

Channel encoding order shall be associated with the order that channel names are identified in the signature (for example given signature 'RGB' the channel order shall be channel 1 – R, channel 2 – G, channel 3 – B) with the following exceptions: for the signature 'GRAY' there is only 1 channel; for the signature 'YCbr' the channel order shall be channel 1 – Y, channel 2 – Cb, channel 3 – Cr; for xCLR and N channel data the order shall be the same as the incoming device channel order.

For abstract profiles the data colour space signature shall one of the signatures in Table 15. If set to zero the spectral PCS signature and spectral range fields shall be used to define the A side of the transform.

For MultiplexLink and MultiplexVisualization profiles the data colour space signature shall be zero.

7.2.9 PCS field (Bytes 20 to 23)

For all profile classes (see Table 14), other than a DeviceLink or MultiplexLink profile, the PCS encoding shall be one of the signatures as defined in Table 16. When the profile/device class is a DeviceLink profile or MultiplexLink, the value of the PCS shall be the appropriate data colour space from Table 15. The field represents the colour space on the B-side of the transform.

The PCS for AToBx/BToAx tags shall always be defined by the PCS field.

To define the use of a colorimetric-based PCS one of the non-spectral colour space signatures in Table 16 shall be used to encode the colour space implied by the PCS field of the profile header. These colour space signatures define both the colour space type and the number of channels associated with the colour space.

Table 16 — Non-spectral PCS colour space signatures

Colour space type	Signature	Hexidecimal encoding
nCIEXYZ or PCSXYZ[a]	'XYZ '	58595A20h
CIELAB or PCSLAB[b]	'Lab '	4C616220h
None (spectral PCS defined by spectral PCS header field)	0	00000000h
[a] The signature 'XYZ ' refers to nCIEXYZ or PCSXYZ, depending upon the context. [b] The signature 'Lab' refers to CIELAB or PCSLAB, depending upon the context.		

Channel encoding order shall be associated with the order that channel names are identified in the signature.

7.2.10 Date and time field (bytes 24 to 35)

This header field shall contain the date and time that the profile was first created, encoded as a dateTimeNumber.

7.2.11 Profile file signature field (bytes 36 to 39)

The profile file signature field shall contain the value 'acsp' (61637379h) as a profile file signature.

7.2.12 Primary platform field (bytes 40 to 43)

This field may be used to identify the primary platform/operating system framework for which the profile was created. The primary platforms that have been identified and the signatures that shall be used are shown in Table 17. If no primary platform is identified, this field shall be zero (00000000h).

Table 17 — Primary platforms

Primary platform	Signature	Hexidecimal encoding
Apple Computer, Inc.	'APPL'	4150504Ch
Microsoft Corporation	'MSFT'	4D534654h
Silicon Graphics, Inc.	'SGI '	53474920h
Sun Microsystems, Inc.	'SUNW'	53554E57h

7.2.13 Profile flags field (bytes 44 to 47)

The profile flags field shall contain flags to indicate various hints for the CMM such as distributed processing and caching options. The least-significant 16 bits are reserved for the ICC. Flags in bit positions 0, 1 and 2 shall be used as indicated in Table 18.

Table 18 — Profile flags

Bit position	Field length (bits)	Field contents
0	1	Embedded profile (0 if not embedded, 1 if embedded in file)
1	1	Profile cannot be used independently of the embedded colour data (set to 1 if true, 0 if false)
2	1	MCS channels in this profile shall be a subset of the MCS channels in the profile it is connected to (set to 1 if true, 0 if false)

Bit 2 shall define MCS subset requirements for connecting profiles. When bit 2 is set the CMM shall be instructed to fail the linking of the profile containing this bit (containing profile) to another profile (second profile) using an MCS based connection if the second profile does not contain all the channels in its MCS (defined in second profile's multiplexTypeArrayTag, see 9.2.85) that the containing profile has in its MCS (defined in the containing profile's multiplexTypeArrayTag, see 9.2.85).

7.2.14 Device manufacturer field (bytes 48 to 51)

This field may be used to identify a device manufacturer. If used, the signature shall match the signature contained in the appropriate section of the ICC signature registry at http://www.color.org (see Clause 5). If not used, this field shall be zero (00000000h).

7.2.15 Device model field (bytes 52 to 55)

This field may be used to identify a device model. If used, the signature shall match the signature contained in the appropriate section of the ICC signature registry at http://www.color.org (see Clause 5). If not used, this field shall be zero (00000000h).

7.2.16 Device attributes field (bytes 56 to 63)

The device attributes field shall contain flags used to identify attributes unique to the particular device setup for which the profile is applicable. The least-significant 32 bits of this 64-bit value are defined by the ICC. Bit usage shall be used as shown in Table 19.

Table 19 — Device attributes

Bit position	Field length (bits)	Attribute
0	1	Reflective (0) or transparency (1)
1	1	Glossy (0) or matte (1)
2	1	Media polarity, positive (0) or negative (1)
3	1	Colour media (0), black and white media (1)
4	1	Paper/paperboard (0), non-paper-based (1)
5	1	Non-textured (0), textured (1)
6	1	Isotropic (0), non-isotropic (1)
7	1	Non self-luminous (0) or self-luminous (1)
8 to 31	24	Reserved (set to binary zero)
32 to 63	32	Use not defined by ICC (vendor specific)

NOTE Notice that bit 0 to bit 6 describe the media, not the device. For example, a profile for a colour scanner that has been loaded with black and white film has bit 3 set on, regardless of the value in the data colour space field (see 7.2.8). If the media is not inherently "colour" or "black and white" (such as the paper in an inkjet printer), the reproduction takes on the property of the device. Thus, an inkjet printer loaded with a colour ink cartridge can be thought to have "colour" media.

7.2.17 Rendering intent field (bytes 64 to 67)

The rendering intent field shall specify the rendering intent that should be used (or, in the case of a DeviceLink profile, was used) when this profile is (was) combined with another profile. In a sequence of more than two profiles, it applies to the combination of this profile and the next profile in the sequence and not to the entire sequence. Typically, the user or application selects the rendering intent dynamically at runtime or embedding time. Therefore, this flag may not have any meaning until the profile is used in some context, for example in a DeviceLink or an embedded source profile.

The field is a uInt32Number in which the least-significant 16 bits shall be used to encode the rendering intent. The most significant 16 bits shall be zero (0000h).

The defined rendering intents are perceptual, media-relative colorimetric, saturation and ICC-absolute colorimetric. These shall be identified using the values shown in Table 20.

Table 20 — Rendering intents

Rendering intent	Value
Perceptual	0
Media-relative colorimetric	1
Saturation	2
ICC-absolute colorimetric	3

7.2.18 PCS illuminant field (bytes 68 to 79)

The PCS illuminant field shall contain the nCIEXYZ values of the PCS illuminant. If the PCS illuminant is D50, the values shall be $X = 0{,}964\,2$, $Y = 1{,}0$ and $Z = 0{,}824\,9$ encoded as an XYZNumber. If the PCS illuminant is not D50, the values shall correspond to the colorimetry of the illuminant as computed using the illuminant and observer values specified in the spectralViewingConditions tag, as described in 9.2.105.

See Annex A for further details.

NOTE These values are the nCIEXYZ values of CIE illuminant D50.

The precise value of the PCS illuminant depends on the precision and method of computation. CIE Publication 15[11] gives a different value for Z, which corresponds to an nCIEXYZ value of 0,825 1. Such close approximations should be considered as D50.

7.2.19 Profile creator field (bytes 80 to 83)

This field may be used to identify the creator of the profile. If used, the signature should match the signature contained in the device manufacturer section of the ICC signature registry at http://www.color.org. If not used, this field shall be zero (00000000h).

7.2.20 Profile ID field (bytes 84 to 99)

This field, if not zero (00h), shall hold the profile ID. The profile ID shall be calculated using the MD5 fingerprinting method as defined in Internet RFC 1321[12]. The entire profile, whose length is given by the size field in the header, with the profile flags field (bytes 44 to 47, see 7.2.13), rendering intent field (bytes 64 to 67, see 7.2.14), and profile ID field (bytes 84 to 99) in the profile header temporarily set to zeros (00h), shall be used to calculate the ID. A profile ID field value of zero (00h) shall indicate that a profile ID has not been calculated.

It is recommended that profile creators compute and record a profile ID.

7.2.21 Spectral PCS field (bytes 100 to 103)

This field, when non-zero, defines the meaning of spectrally-based PCS data in a profile.

If DToBx/BToDx or brdfDToBx/brdfBToDx or directionalDToBx/directionalBToDx tags are present and this field is non-zero, then the use of a spectrally-based PCS shall be defined for DToBx/BToDx or brdfDToBx/brdfBToDx or directionalDToBx/directionalBToDx tags. If this field is zero then the use of DToBx/BToDx or brdfDToBx/brdfBToDx or directionalDToBx/directionalBToDx is not defined.

Spectral data shall be assumed to be sampled at equal intervals, with a given start, end wavelength and number of steps. Unless otherwise specified, the type, dimensions and range of the spectra in the different tags shall be defined by the spectral PCS field in addition to the spectral PCS range and bi-spectral PCS range fields.

For normal spectra (i.e. spectra containing no fluorescent emission component), this implies that the spectral dimension of object characterization spectra shall be the same. If the object characterization spectra are defined by the Donaldson matrix, the Donaldson matrix shall be an nXm matrix with m defined internally in the corresponding colour table.

A distinction is made between self-emitting colours and reflective colours, here referred to as luminous colours and object colours. Luminous colours are characterized by their emission spectra, whereas for object colours reflectance or transmission spectra are used. These three types of spectra are referred to as object characterization spectra.

Reflectance spectra are specified in relation to the perfect reflector whereas transmission spectra are related to a perfect transmitter. Therefore, both types of spectra can be seen as relative data. For emission spectra, luminance values are used, hence these are regarded as absolute data.

To define the use of a spectrally-based PCS, one of the spectral colour space signatures in Table 21 shall be used to encode the colour space implied by the spectralPCS field of the profile header. These colour space signatures define both the colour space type and the number of channels associated with the colour space. Therefore, the number of channels implied by the spectralPCS colour space signature shall match the number of channels indicated by the steps field(s) of the corresponding spectralRange structures (7.2.22 and 7.2.23) in the profile header.

Table 21 — BToDx/DToBx or brdfBToDx/brdfDToBx or directionalBToDx/directionalDToBx spectral colour space signatures

Spectral colour space type	Signature identifier	Signature channels	Combined hexidecimal encoding	Signature representation
None (PCS defined by PCS header field)	0	0	00000000h	0
Reflectance spectra with N channels	'rs' (7273h)	1 … 65 535 (0001h … FFFFh)	72730001h … 7273FFFFh	"rs0001" … "rsFFFF"
Transmission spectra with N channels	'ts' (7473h)	1 … 65 535 (0001h to FFFFh)	74730001h … 7473FFFFh	"ts0001" … "tsFFFF"

Radiant (emission) spectra with N channels	'es' (6573h)	1 … 65 535 (0001h … FFFFh)	65730001h … 6573FFFFh	"es0001" … "esFFFF"
Bi-spectral reflectance spectra with N total channels	'bs' (6273h)	1 to 65 535 (0001h … FFFFh)	62730001h … 6273FFFFh	"bs0001" … "bsFFFF"
Bi-spectral reflectance using sparse matrix with N equivalent output channels	'sm' (736d)	1 to 65 535 (0001h … FFFFh)	736D0001h … 736DFFFFh	"sm0001" … "smFFFF"

NOTE Spectral colour space signatures use the same 32-bit binary encoding mechanism as N colour device data signatures (see 6.2.1), with each having a six-character signature representation.

Different types of spectral data can be defined. In most circumstances, only reflectance, transmission or emission spectra are used, but in other circumstances additional data shall be provided according to the processing to be carried out. The range of normal spectra shall be indicated by the spectral PCS range field in the header (7.2.22). To represent bi-spectral data, a form of Donaldson matrix is used and the incident wavelengths corresponding to the columns of the matrix shall be specified by the bi-spectral PCS range field in the header (7.2.23).

7.2.22 Spectral PCS range field (bytes 104 to 109)

This field shall specify the spectral range used for a spectrally-based PCS when the spectralPCS signature field in the profile header is non-zero. If the spectralPCS field is zero then this field shall be zero.

Spectra are normally represented according to their canonical basis, i.e. the spectrum is sampled at equal intervals along the wavelength axis. The wavelength range is represented by a start wavelength (S), end wavelength (E) and number of steps (n). The wavelength interval between steps is given by Formula (4).

$$I = (E - S)/(n - 1) \tag{4}$$

Unless otherwise specified, spectral data in all tags shall be uniformly sampled, with a given start, end wavelength and number of steps as defined by this field. The dimensions and range of the spectra in the different tags shall be defined consistently. For normal spectra, this means that the spectral dimension of object characterization spectra shall be the same.

7.2.23 Bi-Spectral PCS range field (bytes 110 to 115)

This field shall specify the spectral range of the incident light used for a spectrally-based PCS when the spectralPCS signature field in the profile header indicates the use of Bi-spectral reflectance. Otherwise this field shall be zero.

Bi-spectral reflectance characterizes of the interaction of light with a diffuse surface using a Donaldson matrix. The multiplication of such a matrix by a vector representing the illumination results in a vector representing the light reflected off the surface. Columns of a Donaldson matrix correspond to incident wavelengths of light and rows of a Donaldson matrix correspond to reflected wavelengths of light.

Diagonal entries (where incident and reflected wavelengths are the same) correspond to spectral reflectance. Off diagonal entries (below the diagonal) represent the contribution of a change in the reflected light's wavelength (typically due to fluorescence). Fluorescence occurs when light is absorbed and then re-emitted at a longer wavelength. Using Donaldson matrices to represent colours in an

ICC profile allows for a more complete description of colour to be encoded than using only spectral reflectance or simple colorimetry.

7.2.24 MCS field (bytes 116 to 119)

The MCS for AToM0/MToA0/MToB0/MToS0 tags shall always be defined by the MCS field. The field represents the colour space on the M-side of the transform. When this field is non-zero the multiplex channel identification shall be encoded by a multiplexTypeArrayTag (see 9.2.85).

For the MultiplexIdentification and MultiplexVisualization profile classes (see Table 14), the MCS encoding shall be one of the signatures as defined in Table 22.

NOTE Multiplex colour space signatures use the same 32-bit binary encoding mechanism as N colour device data signatures (see 6.2.1), with each having a six-character signature representation.

Table 22 —AToM0/MToA0/MToB0/MToS0 MCS signatures

Multiplex colour space type	Signature identifier	Signature channels	Combined hexidecimal encoding	Signature representation
Multiplex channel values with N channels	'mc' (6d63h)	1 … 65 535 (0001h … FFFFh)	6d630001h … 6d63FFFFh	"mc0001" … "mcFFFF"

For the input profile class (Table 14) the MCS encoding shall be one of the signatures as defined in Table 23.

Table 23 —AToM0/MToA0/MToB0/MToS0 MCS signatures

Multiplex colour space type	Signature identifier	Signature channels	Combined hexidecimal encoding	Signature representation
None (no MCS is used)	0	0	0	0
Multiplex channel values with N channels	'mc' (6d63h)	1 … 65 535 (0001h … FFFFh)	6d630001h … 6d63FFFFh	"mc0001" … "mcFFFF"

For all other profile classes (Table 14) the MCS encoding shall be zero.

7.2.25 Profile/device sub-class (bytes 124 to 127)

This field allows for a profile/device subclass signature associated with the profile class. This field's purpose is to provide a connection with ICS documents that provide specifications for specific colour management workflows. If this field is zero then no profile/device subclass shall be associated with the profile type. When this field is set then the profile sub-version field shall also identify the version associated with the profile/device sub-class that can be referenced with an ICS document.

7.2.26 Reserved field (bytes 124 to 127)

This field of the profile header is reserved for future ICC definition and shall be zero.

7.3 Tag table

7.3.1 Overview

The tag table acts as a table of contents for the tags and an index into the tag data element in the profiles. It shall consist of a 4-byte entry that contains a count of the number of tags in the table followed by a

series of 12-byte entries with one entry for each tag. The tag table therefore contains 4 + 12*n* bytes where *n* is the number of tags contained in the profile. The entries for the tags within the table are not required to be in any particular order nor are they required to match the sequence of tag data element within the profile.

Each 12-byte tag entry following the tag count shall consist of a 4-byte tag signature, a 4-byte offset to define the beginning of the tag data element and a 4-byte entry identifying the length of the tag data element in bytes**.** Table 24 illustrates the structure for this tag table. 7.3.2 to 7.3.5 specify the position and content of the entries composing the tag table.

Table 24 — Tag table structure

Byte offset	Field length (bytes)	Content	Encoded as...
0 to 3	4	Tag count (*n*)	
4 to 7	4	Tag signature	
8 to 11	4	Offset to beginning of tag data element	uInt32Number
12 to 15	4	Size of tag data element	uInt32Number
16 to (12*n*+3)	12(*n*−1)	Signature, offset and size respectively of subsequent *n*−1 tags	
Key *n:* Number of tags contained in the profile			

NOTE The byte offset shown in Table 24 is with respect to the 128-byte header. Thus the tag table starts at byte position 128.

7.3.2 Tag count (byte position 0 to 3)

Byte positions 0 to 3 shall specify the number of tags contained in the tag table, encoded as a uInt32Number.

7.3.3 Tag signature (byte position 4 to 7 and repeating)

Byte positions 4 to 7 (and repeating at 12-byte intervals) shall specify the signature of a tag listed in Clause 9, or of a private tag. Signatures of private tags shall be registered with the ICC as defined in Clause 5.

7.3.4 Offset to beginning of tag data element (byte position 8 to 11 and repeating)

Byte positions 8 to 11 (and repeating at 12-byte intervals) shall specify the address of the beginning of the tag data element, with respect to the beginning of the profile data stream (which has an address of zero), encoded as a uInt32Number.

NOTE For profiles that are not embedded, the number specified is the same as the file offset.

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

7.3.5 Tag data element size (byte position 12 to 15 and repeating)

The tag data element size shall be the number of bytes in the tag data element encoded as a uInt32Number. The value of the tag data element size shall be the number of actual data bytes and shall not include any padding at the end of the tag data element.

7.4 Tag data

The first set of tag data elements shall immediately follow the tag table and all tag data elements, including the last tag data element, shall be padded by no more than three following pad bytes to reach a 4-byte boundary.

The size of individual tag data elements and the accumulated size of all tag data elements shall only be restricted by the limits imposed by the 32-bit tag data offset value and the 32-bit tag data element size value.

8 Required tags

8.1 General

8.2 to 8.10 identify the tags that are required, in addition to the header defined in 7.2, for each profile type.

NOTE Profiles can include additional tags beyond those listed as required.

The intent of requiring certain tags with each type of profile is to provide a common base level of functionality. If a custom CMM is not present, then the required tags have enough information to allow the default CMM to perform the requested colour transformations. The particular models are identified for each profile type and described in detail in Annex A. While the data provided by the required tags might not provide the level of quality obtainable with optional tags and private data, the data provided is adequate for sophisticated device modelling.

8.2 Common requirements

With the exception of ColorEncodingSpace, DeviceLink, MultiplexIdentification and MultiplexLink profiles, all profiles shall contain the following tags:

— profileDescriptionTag (see 9.2.101);

— copyrightTag (see 9.2.55);

— mediaWhitePointTag (see 9.2.88) if the PCS field in header is non-zero;

— spectralWhitePointTag (see 9.2.106) if the spectralPCS field in profile header is non-zero.

NOTE 1 A ColorEncodingSpace profile is not required to have either a profileDescriptionTag, copyrightTag, mediaWhitePointTag or spectralWhitePointTag.

NOTE 2 A DeviceLink, MultiplexIdentification or MultiplexLink profile is not required to have a mediaWhitePointTag or spectralWhitePointTag.

8.3 Input profiles

Input profiles are generally used with devices such as scanners and digital cameras.

In addition to the tags listed in 8.2 an input profile shall contain one or more of the following: AToB0Tag (see 9.2.1), AToB1Tag (see 9.2.2), AToB2Tag (see 9.2.3), AToB3Tag (see 9.2.4), DToB0Tag (see 9.2.76), DToB1Tag (see 9.2.77), DToB2Tag (see 9.2.78), DToB3Tag (see 9.2.79).

The colorantInfoTag (9.2.53) should be used for colour spaces with either an ‘xCLR’ signature or a signature represented by “ncXXXX”. It enables the names and optionally colorimetric and/or spectral values of the colorants to be specified for these colour spaces (Table 15), as these names are not otherwise implicit in the choice of the colour space.

The BToA0Tag (see 9.2.30), BToA1Tag (see 9.2.39), BToA2Tag (see 9.2.40), BToA3Tag (see 9.2.41), BToD0Tag (see 9.2.42), BToD1Tag (see 9.2.43), BToD2Tag (see 9.2.44) and BToD3Tag (see 9.2.45) may

also be included in an N-component LUT-based input profile. If these are present, their usage shall be as defined in Table 25 (see 9.2.1).

The gamutBoundaryDescriptor0Tag (see 9.2.80), gamutBoundaryDescriptor1Tag (see 9.2.81), gamutBoundaryDescriptor2Tag (see 9.2.82) and/or gamutBoundaryDescriptor3Tag (see 9.2.83) may be included.

MCS connection may be included in addition to PCS based tags. When the MCS header field is non-zero the input class shall also include an AToM0Tag (see 9.2.5) and a multiplexTypeArrayTag (see 9.2.85).

8.4 Display profiles

This class of profiles represents display devices such as monitors and projectors.

In addition to the tags listed in 8.2 a display profile shall contain the following tags:

— one or more of the following: AToB0Tag (see 9.2.1), AToB1Tag (see 9.2.2), AToB2Tag (see 9.2.3), AToB3Tag (see 9.2.4), DToB0Tag (see 9.2.76), DToB1Tag (see 9.2.77), DToB2Tag (see 9.2.78), DToB3Tag (see 9.2.79);

— one or more of the following: BToA0Tag (see 9.2.38), BToA1Tag (see 9.2.39), BToA2Tag (see 9.2.40), BToA3Tag (see 9.2.41), BToD0Tag (see 9.2.42), BToD1Tag (see 9.2.43), BToD2Tag (see 9.2.44), BToD3Tag (see 9.2.45).

The colorantInfoTag (9.2.52) is a recommended tag for colour spaces with either an ‘xCLR’ signature or a signature represented by “ncXXXX”. It enables the names and optionally colorimetric and/or spectral values of the colorants to be specified for these colour spaces (Table 15), as these names are not otherwise implicit in the choice of the colour space.

The gamutBoundaryDescriptor0Tag (see 9.2.80), gamutBoundaryDescriptor1Tag (see 9.2.81), gamutBoundaryDescriptor2Tag (see 9.2.82), and/or gamutBoundaryDescriptor3Tag (see 9.2.83) may be included.

8.5 Output profiles

Output profiles are used to support devices such as printers and film recorders. The types of profiles available for use as output profiles are N-component LUT-based and Monochrome.

In addition to the tags listed in 8.2 an output profile shall contain the following tags:

— one or more of the following: AToB0Tag (see 9.2.1), AToB1Tag (see 9.2.2), AToB2Tag (see 9.2.3), AToB3Tag (see 9.2.4), DToB0Tag (see 9.2.76), DToB1Tag (see 9.2.76), DToB2Tag (see 9.2.76), DToB3Tag (see 9.2.76);

— one or more of the following: BToA0Tag (see 9.2.38), BToA1Tag (see 9.2.39), BToA2Tag (see 9.2.40), BToA3Tag (see 9.2.41), BToD0Tag (see 9.2.42), BToD1Tag (see 9.2.43), BToD2Tag (see 9.2.44), BToD3Tag (see 9.2.45).

The colorantInfoTag (9.2.52) is a recommended tag for colour spaces with either an ‘xCLR’ signature or a signature represented by “ncXXXX”. It enables the names and optionally colorimetric and/or spectral values of the colorants to be specified for these colour spaces (Table 15), as these names are not otherwise implicit in the choice of the colour space.

The gamutBoundaryDescriptor0Tag (see 9.2.80), gamutBoundaryDescriptor1Tag (see 9.2.81), gamutBoundaryDescriptor2Tag (see 9.2.82) and/or gamutBoundaryDescriptor3Tag (see 9.2.83) may be included.

8.6 DeviceLink profile

A device link profile shall contain the following tags:

— profileDescriptionTag (see 9.2.101);

— copyrightTag (see 9.2.55);

— one or more of the following: AToB0Tag (see 9.2.1), DToB0Tag (see 9.2.76).

A profileSequenceInformationTag (see 9.2.102) may be included.

This profile contains a pre-evaluated transform that cannot be undone, which represents a one-way link or connection between devices. It does not represent any device model nor can it be embedded into images.

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

The data colour space field (see 7.2.8) in the DeviceLink profile shall be the same as the data colour space field of the first profile in the sequence used to construct the device link. The PCS field (see 7.2.9) shall be the same as the data colour space field of the last profile in the sequence.

If the data colour space field is set to xCLR, where x is hexadecimal 1 to F or has a signature representation of "ncXXXX" where XXXX is hexadecimal 0001 to FFFF, the colorantInfoTag (9.2.52) is a recommended tag to specify the names and optionally colorimetric and/or spectral values of the input colorants (Table 15), as these names are not otherwise implicit in the choice of the colour space. These colorants represent the input values of the profile.

Correspondingly, if the PCS field is set to xCLR where x is hexadecimal 1 to F or has a signature representation of "ncXXXX" where XXXX is hexadecimal 0001 to FFFF, the colorantInfoOutTag (9.2.53) is a recommended tag to specify the names and optionally colorimetric and/or spectral values of the output colorants (Table 15), as these names are not otherwise implicit in the choice of the colour space. These colorants represent the output values of the profile.

NOTE The colorantOrderTag 'clro' specifies the laydown order of the input colorants, and the colorantoOrderOutTag specifies the laydown order of the output colorants.

8.7 ColorEncodingSpace profile

A ColorSpaceEncoding profile [signature 'cenc' (63656e63h)] shall contain the following tag: referenceNameTag (see 9.2.103).

A ColorSpaceEncoding profile may also contain a colorEncodingParamsTag and a colorSpaceNameTag defined by the following criteria.

In the first mode of operation the referenceNameTag solely contains the text "ISO 22028-1" (quotes excluded) and the elements in the colorEncodingParamsTag shall uniquely determine the colour space encoding parameters and the colorSpaceNameTag shall define the name associated with the colour space encoding.

In the second mode of operation the referenceNameTag contains any text besides "ISO 22028-1" (quotes excluded). In this case the colorSpaceNameTag defines the colour space name and a colorEncodingParamsTag may optionally be present. If the colorEncodingParamsTag exists then any elements in the colorEncodingParamsTag shall provide overrides to the assumed default values for the encoding space. Any elements not in the colorEncodingParamsTag shall have assumed default values associated with the colour space encoding.

8.8 ColorSpace profile

In addition to the tags listed in 8.2 a ColorSpace profile shall contain the following tags:

— one or more of the following: AToB0Tag (see 9.2.1), AToB1Tag (see 9.2.2), AToB2Tag (see 9.2.3), AToB3Tag (see 9.2.4), DToB0Tag (see 9.2.76), DToB1Tag (see 9.2.77), DToB2Tag (see 9.2.78) or DToB3Tag (see 9.2.79);

— one or more of the following: BToA0Tag (see 9.2.38), BToA1Tag (see 9.2.39), BToA2Tag (see 9.2.40), BToA3Tag (see 9.2.41), BToD0Tag (see 9.2.42), BToD1Tag (see 9.2.43), BToD2Tag (see 9.2.44) or BToD3Tag (see 9.2.45).

This profile provides the relevant information to perform a transformation between colour encodings and the PCS. This type of profile is based on modelling rather than device measurement or characterization data. ColorSpace profiles may be embedded in images.

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

The gamutBoundaryDescriptor0Tag (see 9.2.80), gamutBoundaryDescriptor1Tag (see 9.2.81), gamutBoundaryDescriptor2Tag (see 9.2.82) and/or gamutBoundaryDescriptor3Tag (see 9.2.83) may be included.

8.9 Abstract profile

In addition to the tags listed in 8.2 an abstract profile shall contain one or more of the following tags: AToB0Tag (see 9.2.1), DToB0Tag (see 9.2.76).

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

8.10 NamedColor profile

In addition to the tags listed in 8.2 a NamedColor profile shall contain the following tag: namedColorTag (see 9.2.99).

NamedColor 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 colour conversions and one or more named colour profiles to handle named colours.

The namedColorTag provides a combination of PCS, spectral PCS and optional device representation for each named colour in a list of named colours. NamedColor profiles can be device specific in that their data are shaped for a particular device. There might be multiple NamedColor profiles to account for different consumables or multiple named colour vendors. The PCS and spectral PCS representations are provided to support general colour management functionality, and are useful for display and emulation of the named colours.

When using a NamedColor profile with the device for which it is intended, the device representation of the colour specifies the exact device coordinates for each named colour, if available. The PCS and spectral PCS representations 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 colour management system performing the transformations.

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

Additional information about NamedColor profiles can be found in Annex D.

8.11 MultiplexIdentification profile

In addition to the tags listed in 8.2, a MultiplexIdentification profile shall contain the following tags:

— AToM0Tag (see 9.2.6);

— multiplexTypeArrayTag (see 9.2.85).

This profile converts device values into independent multiplex channel values.

8.12 MultiplexLink profile

In addition to the tags listed in 8.2, a MultiplexIdentification profile shall contain the following tags:

— MToA0Tag (see 9.2.90);

— multiplexTypeArrayTag (see 9.2.85).

Profiles of this class can optionally provide the following tag: multiplexDefaultValuesTag (see 9.2.84).

This profile converts multiplex channel values to device values. MultiplexLink profiles shall not be embedded in images.

8.13 MultiplexVisualization profile

In addition to the tags listed in 8.2 a MultiplexVisualization profile shall contain the following tags:

— one or more of the following: MToB0Tag (see 9.2.91), MToB1Tag (see 9.2.92), MToB2Tag (see 9.2.93), MToB3Tag (see 9.2.94), MToS0Tag (see 9.2.95), MToS1Tag (see 9.2.96), MToS2Tag (see 9.2.97), or MToS3Tag (see 9.2.98);

— multiplexTypeArrayTag (see 9.2.85).

Profiles of this class can optionally provide the following tag: multiplexDefaultValuesTag (see 9.2.84).

This profile represents a visualization of multiplex channel values and does not represent any device model. Colour transformations using abstract profiles are performed from either MCS to device or MCS to PCS. MultiplexVisualization profiles shall not be embedded in images.

8.14 Precedence order of tag usage

8.14.1 General

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

8.14.2 Input, display, output or colour space profile types

For input, display, output or colour space profile types, the precedence order of the tag usage for PCSXYZ or PCSLab connection for a designated rendering intent shall be:

a) use the BToA0Tag, BToA1Tag, BToA2Tag, BToA3, AToB0Tag, AToB1Tag, AToB2Tag, or AToB3Tag designated for the rendering intent if present;

b) use the BToA0Tag or AToB0Tag if present, when the tags in 1 are not used;

c) use the BToA1Tag or AToB1Tag if present, when the tags in 1 and 2 are not used;

d) use the BToA3Tag or AToB3Tag if present, when the tags in 1 to 3 are not used.

See Table 25.

When spectrally-based PCS connection is used and the spectralPCS header field is non-zero, for input, display, output or colour space profile types, the precedence order of the tag usage for a designated rendering intent shall be:

1) use the BToD0Tag, BToD1Tag, BToD2Tag, BToD3Tag, DToB0Tag, DToB1Tag, DToB2Tag or DToB3Tag designated for the rendering intent if the tag is present;

2) use the BToD0Tag or DToB0Tag if present, when the tags in 1 are not used;

3) use the BToD1Tag or DToB1Tag if present, when the tags in 1 and 2 are not used;

4) use the BToD3Tag or DToB3Tag if present, when the tags in 1 to 3 are not used.

See Table 25.

When MCS connection is used for input profile types, the precedence order of the tag usage for a designated rendering intent shall be: use the AToM0Tag.

8.14.3 Abstract profile types

For abstract profile types when PCSXYZ or PCSLab connection is used, the precedence order of the tag usage shall be: use the AToB0Tag.

For abstract profile types when a spectrally-based PCS is used, the precedence order of the tag usage shall be: use the DToB0Tag.

8.14.4 DeviceLink profile types

For the DeviceLink profile type, the precedence order of the tag usage shall be: use the AToB0Tag.

8.14.5 MultiplexIdentification profile types

For the MultiplexIdentification profile type, the precedence order of the tag usage shall be: use the AToM0Tag.

8.14.6 MultiplexLink profile types

For either the MultiplexIdentification profile type or the input type when a device to multiplex channel transform is desired, the precedence order of the tag usage shall be: use the MToA0Tag.

8.14.7 MultiplexVisualization profile types

For the MultiplexVisualization profile type, when a multiplex channel to colorimetric PCS transform is desired with PCSXYZ or PCSLab connection for a designated rendering intent, the precedence order of the tag usage shall be:

a) use the MToB0Tag, MToB1Tag, MToB2Tag, MToB3Tag designated for the rendering intent if present;

b) use the MToB0Tag if present, when the tags in 1 are not used;

c) use the MToB1Tag if present, when the tags in 1 and 2 are not used;

d) use the MToB3Tag if present, when the tags in 1 to 3 are not used.

For the MultiplexVisualization profile type, when a multiplex channel to spectral PCS transform is desired using a spectrally-based PCS connection for a designated rendering intent and the spectralPCS header field is non-zero, the precedence order of the tag usage shall be:

1) use the MToS0Tag, MToS1Tag, MToS2Tag, MToS3Tag designated for the rendering intent if present;

2) use the MToS0Tag if present, when the tags in 1 are not used;

3) use the MToS1Tag if present, when the tags in 1 and 2 are not used;

4) use the MToS3Tag if present, when the tags in 1 to 3 are not used.

8.14.8 MCS to parameter-based BRDF profile table usage

For the MultiplexVisualization profile type, when a MCS to parameter-based BRDF colorimetric transform is desired with PCSXYZ or PCSLab connection for a designated rendering intent, the precedence order of the tag usage shall be:

a) use the brdfMToB0Tag, brdfMToB1Tag, brdfMToB2Tag or brdfMToB3Tag designated for the rendering intent if present;

b) use the brdfMToB0Tag if present, when the tags in 1 are not used;

c) use the brdfMToB1Tag if present, when the tags in 1 and 2 are not used;

d) use the brdfMToB3Tag if present, when the tags in 1 to 3 are not used.

For the MultiplexVisualization profile type, when a MCS to parameter-based BRDF spectral transform is desired with a spectrally-based PCS connection for a designated rendering intent and the spectralPCS header field is non-zero, the precedence order of the tag shall be:

1) use the brdfMToS0Tag, brdfMToS1Tag, brdfMToS2Tag or brdfMToS3Tag designated for the rendering intent if the tag is present;

2) use the brdfMToS0Tag when the tags in 1 are not used;

3) use the brdfMToS1Tag when the tags in 1 and 2 are not used;

4) use the brdfMToS3Tag if present, when the tags in 1 to 3 are not used.

8.14.9 BRDF profile table usage

For input, display, output or colour space profile types that provide BRDF function tags which are desired to be used, the precedence order of the tag usage for PCSXYZ or PCSLab connection for a designated rendering intent shall be:

a) use the brdfBToA0Tag, brdfBToA1Tag, brdfBToA2Tag, brdfBToA3, brdfAToB0Tag, brdfAToB1Tag, brdfAToB2Tag or brdfAToB3Tag designated for the rendering intent if present;

b) use the brdfBToA0Tag or brdfAToB0Tag if present, when the tags in 1 are not used;

c) use the brdfBToA1Tag or brdfAToB1Tag if present, when the tags in 1 and 2 are not used;

d) use the brdfBToA3Tag or brdfAToB3Tag if present, when the tags in 1 to 3 are not used.

When spectrally-based PCS connection is used and the spectralPCS header field is non-zero, for input, display, output or colour space profile types that provide BRDF tags which are desired to be used, the precedence order of the tag usage for a designated rendering intent shall be:

1) use the brdfBToD0Tag, brdfBToD1Tag, brdfBToD2Tag, brdfBToD3Tag, brdfDToB0Tag, brdfDToB1Tag, brdfDToB2Tag or brdfDToB3Tag designated for the rendering intent if the tag is present;

2) use the brdfBToD0Tag or brdfDToB0Tag when the tags in 1 are not used;

3) use thebrdf BToD1Tag or brdfDToB1Tag when the tags in 1 and 2 are not used;

4) use the brdfBToA3Tag or brdfDToB3Tag if present, when the tags in 1 to 3 are not used.

8.14.10 Parameter-based BRDF profile table usage

For input, display, output or colour space profile types that provide colorimetric parameter-based BRDF tags which are desired to be used, the precedence order of the tag usage for a designated rendering intent shall be:

a) use the brdfColorimetricParameter0Tag, brdfColorimetricParameter1Tag, brdfColorimetricParameter2Tag, brdfColorimetricParameter3Tag, designated for the rendering intent if present;

b) use the brdfColorimetricParameter0Tag if present, when the tags in 1 are not used;

c) use the brdfColorimetricParameter1Tag if present, when the tags in 1 and 2 are not used;

d) use the brdfColorimetricParameter3Tag if present, when the tags in 1 to 3 are not used.

For input, display, output or colour space profile types that provide spectral parameter-based BRDF tags which are desired to be used and the spectralPCS header field is non-zero, the precedence order of the tag usage for a designated rendering intent shall be:

1) use the brdfSpectralParameter0Tag, brdfSpectralParameter1Tag, brdfSpectralParameter2Tag, brdfSpectralParameter3Tag designated for the rendering intent if the tag is present;

2) use the brdfSpectralParameter0Tag when the tags in 1 are not used;

3) use the brdfSpectralParameter1Tag when the tags in 1 and 2 are not used;

4) use the brdfSpectralParameter3Tag if present, when the tags in 1 to 3 are not used.

8.14.11 Directional profile table usage

For input, display, output or colour space profile types that provide directional tags which are desired to be used, the precedence order of the tag usage for PCSXYZ or PCSLab connection for a designated rendering intent shall be:

a) use the directionalBToA0Tag, directionalBToA1Tag, directionalBToA2Tag, directionalBToA3, directionalAToB0Tag, directionalAToB1Tag, directionalAToB2Tag or directionalAToB3Tag designated for the rendering intent if present;

b) use the directionalBToA0Tag or directionalAToB0Tag if present, when the tags in 1 are not used;

c) use the directionalBToA1Tag or directionalAToB1Tag if present, when the tags in 1 and 2 are not used;

d) use the directionalBToA3Tag or directionalAToB3Tag if present, when the tags in 1 to 3 are not used.

When spectrally-based PCS connection is used and the spectralPCS header field is non-zero, for input, display, output or colour space profile types that provide BRDF tags which are desired to be used, the precedence order of the tag usage for a designated rendering intent shall be:

1) use the directionalBToD0Tag, directionalBToD1Tag, directionalBToD2Tag, directionalBToD3Tag, directionalDToB0Tag, directionalDToB1Tag, directionalDToB2Tag or directionalDToB3Tag designated for the rendering intent if the tag is present;

2) use the directionalBToD0Tag or directionalDToB0Tag when the tags in 1 are not used;

3) use thedirectional BToD1Tag or directionalDToB1Tag when the tags in 1 and 2 are not used;

4) use the directionalBToD3Tag or directionalDToB3Tag if present, when the tags in 1 to 3 are not used.

9 Tag definitions

9.1 General

The public tags defined by extended ICC profiles conforming to this document are listed in 9.2 in alphabetical order. All tags, including private tags, have as their first four bytes a tag signature to identify to profile readers what kind of data are contained within a tag. Each entry in 9.2 contains the tag signatures that shall be used for that tag, the permitted tag types for each tag (see Clause 10) and a brief description of the purpose of each tag.

These individual tags are used to create all possible profiles. The tag signature indicates only the type of data and does not imply anything about the use or purpose for which the data are intended. Clause 8 specifies the tags that shall be included for each type of profile. Any other tag in 9.2 may be used as an optional tag as long as they are not specifically excluded in the definition of a profile class.

The interpretation of some tags is context dependent. This dependency is described in Table 25 which provides a summary of the rendering intent associated with each of the main profile classes and models. The term "undefined" means that the use of the tag in that situation is not specified by the ICC. The ICC recommends that such tags not be included in profiles. If the tag is present, its use is implementation dependent. In general, the BToAxTags represent the inverse operation of the AToBxTags, and DToAxTags represent the inverse of AToDxTags.

9.2 Specific tag listing

9.2.1 AToB0Tag

Tag signature: ‘A2B0’ (41324230h).

Permitted tag types: lutAToBType or multiProcessElementsType.

This tag defines a colour transform from Device, Colour Encoding or colorimetric PCS, to colorimetric PCS, or a colour transform from Device 1 to Device 2, using lookup table tag element structures or a multiProcessElementsType transform. For most profile classes it defines the transform to achieve colorimetric-based perceptual rendering (see Table 25). The processing mechanisms are described in lutAToBType or multiProcessElementsType (see 10.2.12 and 10.2.16).

Table 25 — Profile classes and defined AToBx rendering intents

Profile class	AToB0Tag	AToB1Tag	AToB2Tag	AToB3Tag
Input	Device to colorimetric PCS: perceptual	Device to colorimetric PCS: media relative	Device to colorimetric PCS: saturation	Device to colorimetric PCS: absolute
Display	Device to colorimetric PCS: perceptual	Device to colorimetric PCS: media relative	Device to colorimetric PCS: saturation	Device to colorimetric PCS: absolute
Output	Device to colorimetric PCS: perceptual	Device to colorimetric PCS: media relative	Device to colorimetric PCS: saturation	Device to colorimetric PCS: absolute
ColorSpace	Colour encoding to colorimetric PCS: perceptual	Colour encoding to colorimetric PCS: media relative	Colour encoding to colorimetric PCS: saturation	Colour encoding to colorimetric PCS: absolute
Abstract	Colorimetric PCS to colorimetric PCS	Undefined	Undefined	Undefined
DeviceLink	Device 1 to Device2	Undefined	Undefined	Undefined

Table 25 *(continued)*

Profile class	AToB0Tag	AToB1Tag	AToB2Tag	AToB3Tag
NamedColor	Undefined	Undefined	Undefined	Undefined
ColorEncodingSpace	Undefined	Undefined	Undefined	Undefined
MultiplexIdentification	Undefined	Undefined	Undefined	Undefined
MultiplexLink	Undefined	Undefined	Undefined	Undefined
MultiplexVisualization	Undefined	Undefined	Undefined	Undefined

9.2.2 AToB1Tag

Tag signature: 'A2B1' (41324231h).

Permitted tag types: lutAToBType or multiProcessElementsType.

This tag describes the colour transform from Device or Colour Encoding to colorimetric-based PCS using lookup table tag element structures. For most profile classes, it defines the transform to achieve colorimetric rendering (see Table 25). The processing mechanisms are described in lutAToBType or multiProcessElementsType (see 10.2.12 and 10.2.16).

If this tag is not present then relative colorimetric processing shall be performed by using the absolute colorimetric AToB3Tag and then adjusting the colorimetric PCS values by the media white point.

9.2.3 AToB2Tag

Tag signature: 'A2B2' (41324232h).

Permitted tag types: lutAToBType or multiProcessElementsType.

This tag describes the colour transform from Device or Colour Encoding to colorimetric-based PCS using lookup table tag element structures. For most profile classes, it defines the transform to achieve saturation rendering (see Table 25). The processing mechanisms are described in lutAToBType or multiProcessElementsType (see 10.13 and 10.17).

9.2.4 AToB3Tag

Tag signature: 'A2B3' (41324233h).

Permitted tag types: lutAToBType or multiProcessElementsType.

This tag describes the colour transform from Device or Colour Encoding to colorimetric-based PCS using lookup table tag element structures. For most profile classes, it defines the transform to achieve absolute colorimetric rendering (see Table 25). The processing mechanisms are described in lutAToBType or multiProcessElementsType (see 10.2.12 and 10.2.16).

If this tag is not present then absolute colorimetric processing shall be performed by using the relative colorimetric AToB1Tag and then adjusting the colorimetric PCS values by the media white point.

9.2.5 AToM0Tag

Tag signature: 'A2M0' (41324d30h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see Clause 11) tag that converts from device channel values to multiplex channel values.

The number of data channels provided to the transform shall match the number of channels defined by the deviceColor field in the Profile header.

The number of data channels resulting from the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex channel identifications (see 9.2.85) being passed to the appropriate MCS transform in the connecting profile.

Channels in an AToM0Tag that have no match in the connecting profile MCS shall be ignored.

9.2.6 brdfColorimetricParameter0Tag

Tag signature: 'bcp0' (62637030h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for perceptual rendering. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the colorSpace signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.7 brdfColorimetricParameter1Tag

Tag signature: 'bcp1' (62637031h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for media-relative colorimetric intent rendering. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the colorSpace signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.8 brdfColorimetricParameter2Tag

Tag signature: 'bcp2' (62637032h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for colorimetric saturation intent rendering. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the colorSpace signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the

number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.9 brdfColorimetricParameter3Tag

Tag signature: 'bcp3' (62637033h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for absolute intent rendering. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the colorSpace signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.10 brdfSpectralParameter0Tag

Tag signature: 'bsp0' (62737030h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for perceptual rendering. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the colorSpace signature in the profile header.

The number of output channels of the subtag transform depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the spectral PCS signature in the profile header.

9.2.11 brdfSpectralParameter1Tag

Tag signature: 'bsp1' (62737031h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for media-relative intent rendering. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the colorSpace signature in the profile header.

The number of output channels of the subtag transform depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the spectral PCS signature in the profile header.

9.2.12 brdfSpectralParameter2Tag

Tag signature: ‘bsp2’ (62737032h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for saturation intent rendering. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the colorSpace signature in the profile header.

The number of output channels of the subtag transform depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the spectral PCS signature in the profile header.

9.2.13 brdfSpectralParameter3Tag

Tag signature: ‘bsp3’ (62737033h).

Permitted tag types: tagStructType of type brdfTransformStructure.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the colorSpace signature in the profile header.

The number of output channels of the subtag transform depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the spectral PCS signature in the profile header.

9.2.14 brdfAToB0Tag

Tag signature: ‘bAB0’ (62414230 h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve perceptual intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 26.

Table 26 — BRDF device channel encoding

Input channel index	Channel identification	Encoding type
0	Viewing azimuth angle Φr	azimuthNumber
1	Viewing zenith angle θr	zenithNumber
2	Lighting azimuth angle Φi	azimuthNumber
3	Lighting zenith angle θi	zenithNumber

Table 26 *(continued)*

4	Device channel 0	
…	…	
4+N	Device channel N-1	

The output channels are defined by the encoding implied by the PCS field in the profile header.

9.2.15 brdfAToB1Tag

Tag signature: 'bAB1' (62414231h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve media-relative colorimetric intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 26.

The output channels are defined by the encoding implied by the PCS field in the profile header.

If this tag is not present then relative BRDF-based colorimetric processing shall be performed by using the absolute colorimetric brdfAToB3Tag and then adjusting the colorimetric PCS values by the media white point.

9.2.16 brdfAToB2Tag

Tag signature: 'bAB2' (62414232h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 26.

The output channels are defined by the encoding implied by the PCS field in the profile header.

9.2.17 brdfAToB3Tag

Tag signature: 'bAB3' (62414233h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve absolute intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

The number of input channels to this multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 26.

The output channels are defined by the encoding implied by the PCS field in the profile header.

If this tag is not present then relative BRDF-based colorimetric processing shall be performed by using the relative colorimetric brdfAToB1Tag and then adjusting the colorimetric PCS values by the media white point.

9.2.18 brdfBToA0Tag

Tag signature: 'bBA0' (62424130 h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve perceptual intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and colorimetric-based PCS specified by the PCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the PCS field in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 27.

Table 27 — BRDF colorimetric encoding

Input channel index	Channel identification	Encoding type
0	Viewing azimuth angle Φr	azimuthNumber
1	Viewing zenith angle θr	zenithNumber
2	Lighting azimuth angle Φi	azimuthNumber
3	Lighting zenith angle θi	zenithNumber
4	PCS channel 0	
…	…	
4+N	PCS channel N-1	

The number of output channels shall be the number of device channels defined by the colorSpace signature in the profile header.

9.2.19 brdfBToA1Tag

Tag signature: 'bBA1' (62424131h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve media-relative colorimetric intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and colorimetric-based PCS specified by the PCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the PCS field in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 27.

The output channels shall be the number of device channels defined by the colorSpace signature in the profile header.

If this tag is not present then relative BRDF-based colorimetric processing shall be performed by first adjusting the colorimetric PCS values by the media white point and then using the absolute colorimetric brdfBToA3Tag.

9.2.20 brdfBToA2Tag

Tag signature: 'bBA2' (62424132h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and colorimetric-based PCS specified by the PCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the PCS field in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 27.

The output channels shall be the number of device channels defined by the colorSpace signature in the profile header.

9.2.21 brdfBToA3Tag

Tag signature: 'bBA3' (62424133h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve absolute intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and colorimetric-based PCS specified by the PCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the PCS field in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 27.

The output channels shall be the number of device channels defined by the colorSpace signature in the profile header.

If this tag is not present then relative BRDF-based colorimetric processing shall be performed by first adjusting the colorimetric PCS values by the media white point and then using the relative colorimetric brdfAToB1Tag.

9.2.22 brdfBToD0Tag

Tag signature: 'bBD0' (62424430 h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve perceptual intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and the spectral-based PCS specified by the spectralPCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the spectralPCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the spectralPCS field in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 28.

Table 28 — BRDF spectral encoding

Input channel index	Channel identification	Encoding type
0	Viewing azimuth angle Φr	azimuthNumber
1	Viewing zenith angle θr	zenithNumber
2	Lighting azimuth angle Φi	azimuthNumber
3	Lighting zenith angle θi	zenithNumber
4	Spectral PCS channel 0	
…	…	
4+N	Spectral PCS channel N-1	

The output channels shall be the number of channels implied by the colorSpace signature in the profile header.

9.2.23 brdfBToD1Tag

Tag signature: 'bBD1' (62424431h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve relative intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and the spectral-based PCS specified by the spectralPCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the spectralPCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the spectralPCS field in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 28.

The output channels shall be the number of channels implied by the colorSpace signature in the profile header.

If this tag is not present then relative BRDF-based spectral processing shall be performed by first adjusting the spectral PCS values by the spectral media white point and then using the absolute brdfDToB3Tag.

9.2.24 brdfBToD2Tag

Tag signature: 'bBD2' (62424432h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and the spectral-based PCS specified by the spectralPCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the spectralPCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the spectralPCS field in the profile header. The order

and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 28.

The output channels shall be the number of channels implied by the colorSpace signature in the profile header.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

9.2.25 brdfBToD3Tag

Tag signature: 'bBD3' (62424433h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and the spectral-based PCS specified by the spectralPCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the spectralPCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the spectralPCS field in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 28.

The output channels shall be the number of channels implied by the colorSpace signature in the profile header.

If this tag is not present then relative BRDF-based spectral processing shall be performed by using the relative brdfDToB1Tag and then adjusting the spectral PCS values by the spectral media white point.

9.2.26 brdfDToB0Tag

Tag signature: 'bDB0' (62444230 h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve perceptual intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile.

This use of this tag is not defined when the spectralPCS tag is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 26.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

9.2.27 brdfDToB1Tag

Tag signature: 'bDB1' (62444231h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve relative intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile.

This use of this tag is not defined when the spectralPCS tag is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 26.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

If this tag is not present then relative BRDF-based spectral processing shall be performed by using the absolute brdfDToB3Tag and then adjusting the spectral PCS values by the spectral media white point.

9.2.28 brdfDToB2Tag

Tag signature: 'bDB2' (62444232h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile.

This use of this tag is not defined when the spectralPCS tag is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 26.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

9.2.29 brdfDToB3Tag

Tag signature: 'bDB3' (62444233h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile.

This use of this tag is not defined when the spectralPCS tag is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 26.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

If this tag is not present then relative BRDF-based spectral processing shall be performed by using the relative brdfDToB1Tag and then adjusting the spectral PCS values by the spectral media white point.

9.2.30 brdfMToB0Tag

Tag signature: 'bMB0' (624d4230h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for perceptual intent rendering using multiplex channels as input. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the MCS signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.31 brdfMToB1Tag

Tag signature: 'bMB1' (624d4231h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for media-relative rendering using multiplex channels as input. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the MCS signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.32 brdfMToB2Tag

Tag signature: 'bMB2' (624d4232h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for saturation intent rendering using multiplex channels as input. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the MCS signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.33 brdfMToB3Tag

Tag signature: 'bMB3' (624d4233h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for absolute-colorimetric rendering using multiplex channels as input. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the MCS signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the

number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.34 brdfMToS0Tag

Tag signature: 'bMS0' (624d5330h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for perceptual intent rendering using multiplex channels as input. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile header.

This use of this tag is not defined when the spectralPCS tag is set to zero.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the MCS signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.35 brdfMToS1Tag

Tag signature: 'bMS1' (624d5331h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for media-relative rendering using multiplex channels as input. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile header.

This use of this tag is not defined when the spectralPCS tag is set to zero.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the MCS signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.36 brdfMToS2Tag

Tag signature: 'bMS2' (624d5332h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for saturation intent rendering using multiplex channels as input. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile header.

This use of this tag is not defined when the spectralPCS tag is set to zero.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the MCS signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.37 brdfMToS3Tag

Tag signature: 'bMS3' (624d5333h).

Permitted tag types: tagStructType of type brdfTransformStructure.

This tag defines a BRDF model and its parameters for perceptual rendering using multiplex channels as input. Specifically, it describes a BRDF model that transforms viewing angle, lighting angle, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile header.

This use of this tag is not defined when the spectralPCS tag is set to zero.

For the transform subtag of this structure, the number of input channels shall be the same as the number of channels implied by the MCS signature in the profile header.

The number of output channels of the transform subtag depends on the type of BRDF model. For monochrome type models the output of the transform subtag shall be the number of parameters defined by the BRDF model type. For chromatic type models the number of output values shall be the number of parameters defined by the BRDF model type multiplied by the number of channels implied by the PCS signature in the profile header.

9.2.38 BToA0Tag

Tag signature: 'B2A0' (42324130h).

Permitted tag types: lutBToAType or multiProcessElementsType.

This tag defines a colour transform from a colorimetric-based PCS to Device or Colour Encoding using the lookup table tag element structures or multiProcessElementsType transforms. For most profile classes, it defines the transform to achieve perceptual rendering (see Table 29). The processing mechanisms are described in lutBToAType or multiProcessElementsType (see 10.2.13 and 10.2.16).

Table 29 — Profile classes and defined BToAx rendering intents

Profile class	BToA0Tag	BToA1Tag	BToA2Tag	BToA3Tag
Input	Colorimetric PCS to device: perceptual	Colorimetric PCS to device: media relative	Colorimetric PCS to device: saturation	Colorimetric PCS to device: absolute
Display	Colorimetric PCS to device: perceptual	Colorimetric PCS to device: media relative	Colorimetric PCS to device: saturation	Colorimetric PCS to device: absolute
Output	Colorimetric PCS to device: perceptual	Colorimetric PCS to device: media relative	Colorimetric PCS to device: saturation	Colorimetric PCS to device: absolute
ColorSpace	Colorimetric PCS to colour encoding: perceptual	Colorimetric PCS to colour encoding: colorimetric	Colorimetric PCS to colour encoding: saturation	Colorimetric PCS to colour encoding: saturation
Abstract	Undefined	Undefined	Undefined	Undefined
DeviceLink	Undefined	Undefined	Undefined	Undefined
NamedColor	Undefined	Undefined	Undefined	Undefined
ColorEncodingSpace	Undefined	Undefined	Undefined	Undefined

Table 29 *(continued)*

Profile class	BToA0Tag	BToA1Tag	BToA2Tag	BToA3Tag
MultiplexIdentification	Undefined	Undefined	Undefined	Undefined
MultiplexLink	Undefined	Undefined	Undefined	Undefined
MultiplexVisualization	Undefined	Undefined	Undefined	Undefined

9.2.39 BToA1Tag

Tag signature: 'B2A1' (42324131h).

Permitted tag types: lutBToAType or multiProcessElementsType.

This tag defines a colour transform from a colorimetric-based PCS to Device or Colour Encoding using the lookup table tag element structures or multiProcessElementsType transforms. For most profile classes, it defines the transform to achieve relative colorimetric rendering (see Table 29). The processing mechanisms are described in lutBToAType or multiProcessElementsType (see 10.2.13 and 10.2.16).

If this tag is not present then relative colorimetric processing shall be performed by adjusting the colorimetric PCS values by the media white point and then using the absolute colorimetric BToA3Tag.

9.2.40 BToA2Tag

Tag signature: 'B2A2' (42324132h).

Permitted tag types: lutBToAType or multiProcessElementsType.

This tag defines a colour transform from a colorimetric-based PCS to Device or Colour Encoding using the lookup table tag element structures or multiProcessElementsType transforms. For most profile classes, it defines the transform to achieve saturation rendering (see Table 29). The processing mechanisms are described in or lutBToAType or multiProcessElementsType (see 10.2.13 and 10.2.16).

9.2.41 BToA3Tag

Tag signature: 'B2A3' (42324133h).

Permitted tag types: lutBToAType or multiProcessElementsType.

This tag defines a colour transform from a colorimetric-based PCS to Device or Colour Encoding using the lookup table tag element structures or multiProcessElementsType transforms. For most profile classes, it defines the transform to achieve absolute colorimetric rendering (see Table 29). The processing mechanisms are described in lutBToAType or multiProcessElementsType (see 10.2.13 and 10.2.16).

If this tag is not present then absolute colorimetric processing shall be performed by adjusting the colorimetric PCS values by the media white point and then using the relative colorimetric BToA1Tag.

9.2.42 BToD0Tag

Tag signature 'B2D0' (42324430h).

Permitted tag types: multiProcessElementsType.

This tag defines a colour transform from a spectrally-based PCS (determined by the spectralPCS and PCS fields in the header) to Device. The spectralPCS header field shall be non-zero when this tag is present. This tag defines a spectrally-based PCS-to-device transform with the spectral PCS defined by the spectralPCS, spectralRange, and biSpectralRange fields in the profile header. It supports float32Number-encoded input range, output range and transform. As with the BToA0Tag, it defines a transform to achieve a perceptual rendering (see Table 30). The processing mechanism is described in multiProcessElementsType (see 10.2.16).

Table 30 — Profile classes and defined BToDx rendering intents

Profile class	BToD0Tag	BToD1Tag	BToD2Tag	BToD3Tag
Input	Spectral PCS to device: perceptual	Spectral PCS to device: media relative	Spectral PCS to device: saturation	Spectral PCS to device: absolute
Display	Spectral PCS to device: perceptual	Spectral PCS to device: media relative	Spectral PCS to device: saturation	Spectral PCS to device: absolute
Output	Spectral PCS to device: perceptual	Spectral PCS to device: media relative	Spectral PCS to device: saturation	Spectral PCS to device: absolute
ColorSpace	Spectral PCS to colour encoding: perceptual	Spectral PCS to colour encoding: colorimetric	Spectral PCS to colour encoding: saturation	Spectral PCS to colour encoding: saturation
Abstract	Undefined	Undefined	Undefined	Undefined
DeviceLink	Undefined	Undefined	Undefined	Undefined
NamedColor	Undefined	Undefined	Undefined	Undefined
ColorEncodingSpace	Undefined	Undefined	Undefined	Undefined
MultiplexIdentification	Undefined	Undefined	Undefined	Undefined
MultiplexLink	Undefined	Undefined	Undefined	Undefined
MultiplexVisualization	Undefined	Undefined	Undefined	Undefined

9.2.43 BToD1Tag

Tag signature ‘B2D1’ (42324431h).

Permitted tag types: multiProcessElementsType.

This tag defines a colour transform from a spectrally-based PCS (determined by the spectralPCS and PCS fields in the header) to Device. The spectralPCS header field shall be non-zero when this tag is present. This tag defines a spectrally-based PCS to device transform with the spectral PCS defined by the spectralPCS, spectralRange, and biSpectralRange fields in the profile header. It supports float32Number-encoded input range, output range and transform. As with the BToA0Tag, it defines a transform to achieve a media relative rendering (see Table 30). The processing mechanism is described in multiProcessElementsType (see 10.2.16).

If this tag is not present then relative processing shall be performed by adjusting the PCS values by the media white point and then using the absolute rendering BToD3Tag.

9.2.44 BToD2Tag

Tag signature ‘B2D2’ (42324432h).

Permitted tag types: multiProcessElementsType.

This tag defines a colour transform from a spectrally-based PCS (determined by the spectralPCS and PCS fields in the header) to Device. The spectralPCS header field shall be non-zero when this tag is present. This tag defines a spectrally-based PCS to device transform with the spectral PCS defined by the spectralPCS, spectralRange, and biSpectralRange fields in the profile header. It supports float32Number-encoded input range, output range and transform. As with the BToA0Tag, it defines a transform to achieve a saturation rendering (see Table 30). The processing mechanism is described in multiProcessElementsType (see 10.2.16).

9.2.45 BToD3Tag

Tag signature ‘B2D3’ (42324433h).

Permitted tag types: multiProcessElementsType.

This tag defines a colour transform from a spectrally-based PCS (determined by the spectralPCS and PCS fields in the header) to Device. The spectralPCS header field shall be non-zero when this tag is present. This tag defines a spectrally-based PCS to device transform with the spectral PCS defined by the spectralPCS, spectralRange, and biSpectralRange fields in the profile header. It supports float32Number-encoded input range, output range and transform. As with the BToA0Tag, it defines a transform to achieve an absolute rendering (see Table 30). The processing mechanism is described in multiProcessElementsType (see 10.2.16).

If this tag is not present then absolute processing shall be performed by adjusting the PCS values by the media white point and then using the relative rendering BToD1Tag.

9.2.46 calibrationDateTimeTag

Tag signature: ‘calt’ (63616C74h).

Permitted tag type: dateTimeType.

Profile calibration date and time. This allows applications and utilities to verify if this profile matches a vendor’s profile and how recently calibration has been performed.

9.2.47 charTargetTag

Tag signature: ‘targ’ (74617267h).

Permitted tag type: utf8Type or utf8ZipType.

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

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

If the first seven characters are "ICCHDAT", then the remainder of the text shall be a single space followed by the Reference Name of a characterization data set in the Characterization Data Registry maintained by ICC, and terminated with a NULL byte (00h). The Reference Name in the text shall match exactly (including case) the Reference Name in the registry, which may be found on the ICC website (http://www.color.org).

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

9.2.48 colorEncodingParamsTag

Tag Signature: ‘cept’ (63657074h).

Tag Type: tagStructType of type colorEncodingParamsStructure.

The colorEncodingParamsTag is defined using a colorEncodingParamsStructure. Element members in this structure are assumed to be overrides of parameters assumed by the encoding reference name.

9.2.49 colorSpaceNameTag

Tag Signature: 'csnm' (63736e6dh).

Tag Type: utf8Type.

This tag defines the reference name for the three component colour encoding when the profile uniquely defines all the necessary parameters for the encoding. This occurs when the referenceNameTag solely contains the text “ISO 22028-1” (quotes excluded).

If the referenceNameTag does not solely contain the text “ISO 22028-1” then the colorSpaceNameTag shall contain the same text as the referenceNameTag (if the profile is present).

9.2.50 colorantOrderTag

Tag signature: ’clro’ (636C726Fh).

Permitted tag type: colorantOrderType.

This tag specifies the laydown order of colorants associated with the data colour space field in the profile header (see 7.2.8) when the data colour space field is either an xCLR where x is a hexadecimal value from 1 to F, or has a signature representation of “ncXXXX” where X is a hexadecimal value from 1 to FFFF.

9.2.51 colorantOrderOutTag

Tag signature: ’cloo’ (636c6f6fh).

Permitted tag type: colorantOrderType.

This tag specifies the laydown order of colorants associated with the PCS field in the profile header (see 7.2.8) when the PCS field is either an xCLR where x is a hexadecimal value from 1 to F, or has a signature representation of “ncXXXX” where X is a hexadecimal value from 1 to FFFF. This tag is used for DeviceLink profiles only.

9.2.52 colorantInfoTag

Tag signature: 'clin' (636c696eh).

Permitted tag type: tagArrayType with an array type identifier of ‘cinf’ (63696e66h).

This tag identifies the colorants associated with the data colour space field header (see 7.2.8) when the data colour space field is either an xCLR where x is a hexadecimal value from 1 to F, or has a signature representation of “ncXXXX” where X is a hexadecimal value from 1 to FFFF. The colorant information is provided as an array of colorantInfoStructure elements. Each colorantInfoStructure entry provides a name for the colorant and optionally colorimetric or spectral information. See 12.2.2 for a complete description of contents and usage of a colorantInfoStructure.

9.2.53 colorantInfoOutTag

Tag signature: ‘clio’ (636C696fh).

Permitted tag type: tagArrayType with an array type identifier of ‘cinf’ (63696e66h).

This tag identifies the colorants associated with the PCS colour space field header (see 7.2.8) when the PCS colour space field is either an xCLR where x is a hexadecimal value from 1 to F, or has a signature representation of “ncXXXX” where X is a hexadecimal value from 1 to FFFF. The colorant information is provided as an array of colorantInfoStructure elements. Each colorantInfoStructure entry provides a name for the colorant and optionally colorimetric and/or spectral information. See 12.2.2 for a complete description of contents and usage of a colorantInfoStructure.

This tag is used for DeviceLink profiles only.

9.2.54 colorimetricIntentImageStateTag

Tag signature: ‘ciis’ (63696973h).

Permitted tag type: signatureType.

This tag is fully specified by ISO 15076-1 colorimetricIntentImageStateTag.

This tag indicates the image state of PCS colorimetry produced using the colorimetric intent transforms. If present, the colorimetricIntentImageStateTag shall specify one of the ICC-defined image states shown in Table 31 and described herein. Other image state specifications are reserved for future ICC use.

The notable difference between usage in ISO 15076-1 and this document is that an arbitrary observer and white point can now be associated with the colorimetric PCS using PCC (see 6.3.2) resulting in a deprecation of the chromaticAdaptationTag defined in ISO 15076-1. Therefore, colorimetry for each of the states defined by this tag should be directly encoded without the need for chromatic adaptation in the colorimetric colour transforms.

NOTE 1 When the state of the image colorimetry represented in the PCS is different from that of the image data in the file, the colorimetric intent image state includes the word "estimates". This will be the case when transformation of the image file data to colorimetry is not fully deterministic.

EXAMPLE If the spectral sensitivities of a digital camera sensor (or photographic film) are not a linear transform of the CIE XYZ CMFs, there will not be a single "correct" transform to focal plane colorimetry.

Table 31 — colorimetricIntentImageStateTag signatures

Colorimetric intent image state	Signature	Hexidecimal encoding
scene colorimetry estimates	'scoe'	73636F65h
scene appearance estimates	'sape'	73617065h
focal plane colorimetry estimates	'fpce'	66706365h
reflection hardcopy original colorimetry	‘rhoc’	72686F63h
reflection print output colorimetry	‘rpoc’	72706F63h

The tag value 'scoe' (scene colorimetry estimates) shall indicate that colorimetry in the PCS represents estimates of the colorimetry of the scene, as viewed from the capture point. With the media-relative colorimetric intent, the colorimetry is relative to the scene encoding maximum. With the ICC-absolute colorimetric intent, the colorimetry is relative to the scene adopted white. The scene colorimetry can result from a real scene, a synthetically generated scene, an edited scene, or some combination of these, but shall be interpreted as actual scene colorimetry for subsequent processing.

For scene colorimetry estimates, the mediaWhitePointTag is populated with the XYZ tristimulus values of the scene encoding maximum white, normalized to be relative to the scene adopted white (perfect diffuser), and then converted to the corresponding tristimulus values for the PCS white. The scene adopted white Y value is normalized to 1,0; the mediaWhitePointTag Y value is relative to the scene adopted white Y value and can be larger than 1,0.

NOTE 2 The un-normalized adopted white values are stored in the illuminant field in the viewing conditions tag.

The tag value 'sape' (scene appearance estimates) shall indicate that colorimetry in the PCS represents estimates of the appearance of the scene, as viewed from the capture point, fully adapted to the ISO 3664 P2 viewing conditions. With the media relative colorimetric intent, the corresponding colorimetry is relative to the scene encoding maximum white. With the ICC-absolute colorimetric intent, the corresponding colorimetry is relative to the scene adopted white. The scene appearance estimates may result from a real scene, a synthetically generated scene, an edited scene, or some combination of these, but shall be interpreted as scene appearance estimates for an actual scene in subsequent processing. When this image state is specified, the ISO 3664 P2 viewing conditions shall be specified in the spectral viewing conditions tag.

For scene appearance estimates, the mediaWhitePointTag is populated with the XYZ tristimulus values of the scene encoding maximum white, normalized to be relative to the scene adopted white (perfect diffuser), and then converted to the corresponding tristimulus values for the PCS white point defined in

the spectral viewing conditions tag. The scene adopted white Y value is normalized to 1,0; the mediaWhitePointTag Y value is relative to the scene adopted white Y value and can be larger than 1,0.

The tag value 'fpce' (focal plane colorimetry estimates) shall indicate that colorimetry in the PCS represents estimates of the colorimetry of the light present at the focal plane of a camera (digital or film). With the media relative colorimetric intent, the colorimetry is relative to the focal-plane encoding maximum white. With the ICC-absolute colorimetric intent, the colorimetry is relative to the focal plane adopted white. The focal plane colorimetry may result from a real scene, a synthetically generated scene, an edited scene, or some combination of these, but shall be interpreted as focal plane colorimetry for subsequent processing. When this colorimetric intent image state is specified, the actual focal plane viewing conditions, including the adopted white, shall be specified in the spectral viewing conditions tag.

For focal plane colorimetry estimates, the mediaWhitePointTag is populated with the XYZ tristimulus values of the focal plane encoding maximum white, normalized to be relative to the focal plane adopted white (perfect diffuser), and then converted to the corresponding tristimulus values for the PCS white point (if required). The focal plane adopted white Y value is normalized to 1,0; the mediaWhitePointTag Y value is relative to the focal plane adopted white Y value and can be larger than 1,0.

NOTE 3 The effects of any optics in or attached to the camera are included in the focal plane colorimetry estimates; this includes lens flare and filters.

NOTE 4 The un-normalized adopted white values are stored in the illuminant field in the spectral viewing conditions tag.

The tag value 'rhoc' (reflection hardcopy original colorimetry) shall indicate that colorimetry in the PCS represents the colorimetry of a reflection hardcopy original that has been digitally scanned. With the media relative colorimetric intent, the colorimetry is normalized relative to the scan condition encoding maximum white. With the ICC-absolute colorimetric intent, the colorimetry is relative to the perfect reflecting diffuser. When this colorimetric intent image state is specified, the scan illumination conditions, including the adopted white, shall be specified in the spectral viewing conditions.

NOTE 5 The un-normalized adopted white values are stored in the illuminant field in the spectral viewing conditions tag.

The tag value 'rpoc' (reflection print output colorimetry) shall indicate that colorimetry in the PCS represents the colorimetry of reflection print output. With the media relative colorimetric intent, the colorimetry is normalized relative to the print medium white point, measured under the actual print viewing conditions. With the ICC-absolute colorimetric intent, the colorimetry is relative to the perfect reflecting diffuser after chromatic adaptation. When this colorimetric intent image state is specified, the print viewing conditions, including the adopted white, shall be specified in the spectral viewing conditions tag.

NOTE 6 The un-normalized adopted white values are stored in the illuminant field in the spectral viewing conditions tag.

9.2.55 copyrightTag

Tag signature: ‘cprt’ (63707274h).

Permitted tag type: multiLocalizedUnicodeType.

This tag contains the text copyright information for the profile.

9.2.56 customToStandardPccTag

Tag signature: 'c2sp' (63327370h).

Permitted Tag types: multiProcessElementsType.

This tag provides the transform needed to convert from the colorimetry defined by the observer and illuminant defined in the spectralViewingConditionsTag to the colorimetry defined by the CIE 1931

Standard Colorimetric Observer with a D50 illuminant. The multiProcessElementsType structure shall define a sequence of one or more transforms that performs this conversion.

The number of both the input and output channels of the transform shall be three.

9.2.57 cxfTag

Tag signature: 'CxF ' (43784620).

Permitted tag type: utf8Type, utf8ZipType.

This tag contains a Color Exchange Format file. The CxF/X file contains the characterization target and corresponding measurement data. The CxF/X file is an XML document and shall be encoded as specified by ISO 17972-1. The CxF/X specification requires that UTF-8 be used.

The cxfTag shall contain the characterization set and measurement data used to create the profile. The tag may contain any other data that conforms to the CxF/X specification.

9.2.58 deviceMfgDescTag

Tag signature: 'dmnd' (646D6E64h).

Permitted tag type: multiLocalizedUnicodeType.

Structure containing invariant and localizable versions of the device manufacturer for display. The content of this structure is described in 10.2.15.

9.2.59 deviceModelDescTag

Tag signature: 'dmdd' (646D6464h).

Permitted tag type: multiLocalizedUnicodeType.

Structure containing invariant and localizable versions of the device model for display. The content of this structure is described in 10.2.15.

9.2.60 directionalAToB0Tag

Tag signature: 'dAB0' (64414230 h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve perceptual intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 32.

Table 32 — Directional device channel encoding

Input channel index	Channel identification	Encoding type
0	Viewing azimuth angle Φr	azimuthNumber
1	Viewing zenith angle θr	zenithNumber
2	Relative Horizontal Position rx	horizontalType
3	Relative Vertical Position ry	verticalType
4	Device channel 0	
…	…	
4+N	Device channel N−1	

The output channels are defined by the encoding implied by the PCS field in the profile header.

9.2.61 directionalAToB1Tag

Tag signature: ‘dAB1’ (64414231h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve media-relative colorimetric intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 32.

The output channels are defined by the encoding implied by the PCS field in the profile header.

If this tag is not present then relative directional-based colorimetric processing shall be performed by using the absolute colorimetric directionalAToB3Tag and then adjusting the colorimetric PCS values by the media white point.

9.2.62 directionalfAToB2Tag

Tag signature: ‘dAB2’ (64414232h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 32.

The output channels are defined by the encoding implied by the PCS field in the profile header.

9.2.63 directionalAToB3Tag

Tag signature: ‘dAB3’ (64414233h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve absolute intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and Device or Colour Encoding to the colorimetric-based PCS specified by the PCS field in the profile header.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 32.

The output channels are defined by the encoding implied by the PCS field in the profile header.

If this tag is not present then relative directional-based colorimetric processing shall be performed by using the relative colorimetric directionalAToB1Tag and then adjusting the colorimetric PCS values by the media white point.

9.2.64 directionalBToA0Tag

Tag signature: ‘dBA0’ (64424130 h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve perceptual intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and colorimetric-based PCS specified by the PCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the PCS field in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 33.

Table 33 — Directional colorimetric encoding

Input channel index	Channel identification	Encoding type
0	Viewing azimuth angle Φr	azimuthNumber
1	Viewing zenith angle θr	zenithNumber
2	Relative Horizontal Position rx	horizontalNumber
3	Relative Vertical Position ry	verticalNumber
4	PCS channel 0	
…	…	
4+N	PCS channel N−1	

The output channels shall be the number of device channels defined by the colorSpace signature in the profile header.

9.2.65 directionalBToA1Tag

Tag signature: ‘dBA1’ (64424131h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve media-relative colorimetric intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and colorimetric-based PCS specified by the PCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the PCS field in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 33.

The number of output channels shall be the number of device channels defined by the colorSpace signature in the profile header.

If this tag is not present then relative directional-based colorimetric processing shall be performed by first adjusting the colorimetric PCS values by the media white point and then using the absolute colorimetric directionalBToA3Tag.

9.2.66 directionalBToA2Tag

Tag signature: ‘dBA2’ (64424132h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and colorimetric-based PCS specified by the PCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the PCS field in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 33.

The number of output channels shall be the number of device channels defined by the colorSpace signature in the profile header.

9.2.67 directionalBToA3Tag

Tag signature: ‘dBA3’ (64424133h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve absolute intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and colorimetric-based PCS specified by the PCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the PCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the PCS field in the profile header. The order and

encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 33.

The number of output channels shall be the number of device channels defined by the colorSpace signature in the profile header.

If this tag is not present then relative BRDF-based colorimetric processing shall be performed by first adjusting the colorimetric PCS values by the media white point and then using the relative colorimetric brdfAToB1Tag.

9.2.68 directionalBToD0Tag

Tag signature: 'dBD0' (64424430 h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve perceptual intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and the spectral-based PCS specified by the spectralPCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the spectralPCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the spectralPCS field in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 34.

Table 34 — Directional spectral encoding

Input channel index	Channel identification	Encoding type
0	Viewing azimuth angle Φr	azimuthNumber
1	Viewing zenith angle θr	zenithNumber
2	Relative Horizontal Position rx	horizontalNumber
3	Relative Vertical Position ry	verticalNumber
4	Spectral PCS channel 0	
…	…	
4+N	Spectral PCS channel N−1	

The number of output channels shall be the number of channels implied by the colorSpace signature in the profile header.

9.2.69 directionalBToD1Tag

Tag signature: 'dBD1' (64424431h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve relative intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and the spectral-based PCS specified by the spectralPCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the spectralPCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the spectralPCS field in the profile header. The order and

encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 34.

The number of output channels shall be the number of channels implied by the colorSpace signature in the profile header.

If this tag is not present then relative BRDF-based spectral processing shall be performed by first adjusting the spectral PCS values by the spectral media white point, and then using the absolute brdfDToB3Tag.

9.2.70 directionalBToD2Tag

Tag signature: 'bBD2' (64424432h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and the spectral-based PCS specified by the spectralPCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the spectralPCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the spectralPCS field in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 34. The number of output channels shall be the number of channels implied by the colorSpace signature in the profile header.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

9.2.71 directionalBToD3Tag

Tag signature: 'dBD3' (64424433h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and the spectral-based PCS specified by the spectralPCS field in the profile header to Device or Colour Encoding.

This use of this tag is not defined when the spectralPCS field in the profile header is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels defined by the encoding implied by the spectralPCS field in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 34. The number of output channels shall be the number of channels implied by the colorSpace signature in the profile header.

If this tag is not present then relative BRDF-based spectral processing shall be performed by using the relative brdfDToB1Tag and then adjusting the spectral PCS values by the spectral media white point.

9.2.72 directionalDToB0Tag

Tag signature: 'dDB0' (64444230 h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve perceptual intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle

and relative position of the viewing area, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile header.

This use of this tag is not defined when the spectralPCS tag is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 32.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

9.2.73 directionalDToB1Tag

Tag signature: ‘dDB1’ (64444231h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve relative intent rendering in relationship viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile header.

This use of this tag is not defined when the spectralPCS tag is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 32.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

If this tag is not present then relative directional-based spectral processing shall be performed by using the absolute directionalDToB3Tag and then adjusting the spectral PCS values by the spectral media white point.

9.2.74 directionalDToB2Tag

Tag signature: ‘dDB2’ (64444232h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile header.

This use of this tag is not defined when the spectralPCS tag is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 32.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

9.2.75 directionalDToB3Tag

Tag signature: ‘dDB3’ (64444233h).

Permitted tag types: multiProcessElementType.

This tag defines the transform to achieve saturation intent rendering in relation to viewing angles and relative position of a viewing area. Specifically, it describes the colour transform from viewing angle and relative position of the viewing area, and Device or Colour Encoding to the spectral-based PCS specified by the spectralPCS field in the profile header.

This use of this tag is not defined when the spectralPCS tag is set to zero.

The number of input channels to the multiProcessElementsType-based tag shall be four plus the number of channels implied by the colorSpace signature in the profile header. The order and encoding of the directional information and device channels provided to the multiProcessElementType are shown in Table 32.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

If this tag is not present then relative directional-based spectral processing shall be performed by using the relative directionalDToB1Tag and then adjusting the spectral PCS values by the spectral media white point.

9.2.76 DToB0Tag

Tag signature 'D2B0' (44324230h).

Permitted tag types: multiProcessElementsType.

This tag has a different behaviour to the DToB0Tag in ISO 15076-1. It defines a colour transform from device to a spectrally-based PCS (determined by the spectralPCS field in the header). When this tag is present, the spectralPCS header field shall be non-zero. This tag defines a device to spectrally-based PCS transform with the spectral PCS defined by the spectralPCS, spectralRange, and biSpectralRange fields in the profile header. It supports float32Number-encoded input range, output range and transforms. As with the AToB0Tag, it defines a transform to achieve a perceptual rendering. The processing mechanism is described in multiProcessElementsType (see 10.2.16).

9.2.77 DToB1Tag

Tag signature 'D2B1' (44324231h).

Permitted tag types: multiProcessElementsType.

This tag has a different behaviour to the DToB1Tag in ISO 15076-1. It defines a colour transform from device to a spectrally-based PCS (determined by the spectralPCS field in the header). When this tag is present, the spectralPCS header field shall be non-zero. This tag defines a device to spectrally-based PCS transform with the spectral PCS defined by the spectralPCS, spectralRange, and biSpectralRange fields in the profile header. It supports float32Number-encoded input range, output range and transforms. As with the AToB0Tag, it defines a transform to achieve a relative rendering. The processing mechanism is described in multiProcessElementsType (see 10.2.16).

If this tag is not present then relative colorimetric processing shall be performed by using the absolute DToB3Tag and then adjusting the PCS values by the media white point.

9.2.78 DToB2Tag

Tag signature 'D2B2' (44324232h).

Permitted tag types: multiProcessElementsType.

This tag has a different behaviour to the DToB2Tag in ISO 15076-1. It defines a colour transform from device to a spectrally-based PCS (determined by the spectralPCS field in the header). When this tag is present, the spectralPCS header field shall be non-zero. This tag defines a device to spectrally-based PCS transform with the spectral PCS defined by the spectralPCS, spectralRange, and biSpectralRange fields in the profile header. It supports float32Number-encoded input range, output range and transforms. As

with the AToB0Tag, it defines a transform to achieve a saturation rendering. The processing mechanism is described in multiProcessElementsType (see 10.2.16).

9.2.79 DToB3Tag

Tag signature 'D2B3' (44324233h).

Permitted tag types: multiProcessElementsType.

This tag has a different behaviour to the DToB3Tag in ISO 15076-1. It defines a colour transform from device to a spectrally-based PCS (determined by the spectralPCS field in the header). When this tag is present, the spectralPCS header field shall be non-zero. This tag defines a device to spectrally-based PCS transform with the spectral PCS defined by the spectralPCS, spectralRange, and biSpectralRange fields in the profile header. It supports float32Number-encoded input range, output range and transforms. As with the AToB0Tag, it defines a transform to achieve an absolute rendering. The processing mechanism is described in multiProcessElementsType (see 10.2.16).

If this tag is not present then absolute colorimetric processing shall be performed by using the relative DToB1Tag and then adjusting the PCS values by the media white point.

9.2.80 gamutBoundaryDescription0Tag

Tag signature: 'gbd0' (67626430h).

Permitted tag types: gamutBoundaryDescriptionType.

This tag defines the gamut boundary of the reference medium gamut that was used for the creation of the perceptual transform.

9.2.81 gamutBoundaryDescription1Tag

Tag signature: 'gbd1' (67626431h).

Permitted tag types: gamutBoundaryDescriptionType.

This tag defines the gamut boundary for the relative colorimetric transform.

9.2.82 gamutBoundaryDescription2Tag

Tag signature: 'gbd2' (67626432h).

Permitted tag types: gamutBoundaryDescriptionType.

This tag defines the gamut boundary for the saturation intent transform.

9.2.83 gamutBoundaryDescription3Tag

Tag signature: 'gbd3' (67626433h).

Permitted tag types: gamutBoundaryDescriptionType.

This tag defines the gamut boundary for the absolute colorimetric intent transform. The presence of the DToB3 or BToD3 tags may require a gamut boundary description that is different from gamutBoundaryDescription1Tag.

9.2.84 multiplexDefaultValuesTag

Tag signature: 'mdv ' (6d647620h).

Permitted tag types: uInt8NumberArray, uInt16NumberArray, float16NumberArray float32NumberArray.

The multiplexDefaultValuesTag defines a default multiplex channel value for each multiplex channel identified in the multiplexTypeArrayTag. The default values shall be used for processing by the destination profile when the source profile does contain the multiplex channel identifier in its multiplexTypeArrayTag.

The encoding of integer based values shall be interpreted as a logical 0,0 to 1,0 when processed by the MToA0Tag, MToB0Tag, MToB1Tag, MToB2Tag, MToB3Tag, MToS0Tag, MToS1Tag, MToS2Tag, or MToS3Tag. Floating point values shall be directly used by the MToA0Tag, MToB0Tag, MToB1Tag, MToB2Tag, MToB3Tag, MToS0Tag, MToS1Tag, MToS2Tag, or MToS3Tag.

The multiplexDefaultValuesTag is optional, and if not present then a default value of 0,0 shall be used for processing when source multiplex channel data are not available.

The number of array entries in a multiplexDefaultValuesTag shall be the same as the number of multiplex colour channels indicated by the signature used in the MCS profile header field.

9.2.85 multiplexTypeArrayTag

Tag signature: ‘mcta’ (6d637461h).

Permitted tag type: tagArray of utf8Type.

tagArray type signature: ‘mcta’ (6d637461h).

The multiplexTypeArrayTag defines a multiplex channel type name for each channel in the MCS for the purpose of profile connection.

MCS connection between profiles is performed by passing multiplex channel values between channels that have identical multiplex channel type identifications. Channels with a multiplex channel type in the source profile that are not in the destination profile are ignored. Channels with multiplex channel types in the destination profile that are not in the source profile are processed with a multiplex channel value of zero.

NOTE The order of multiplex channel identification of connected profiles does not need to be the same.

Each multiplex channel type name shall be unique within a multiplexTypeArrayTag.

Matching of multiplex channel type names shall be case sensitive.

The number of sub-tag entries in a multiplexTypeArrayTag shall be the same as the number of multiplex colour channels indicated by the signature used in the MCS profile header field.

9.2.86 measurementInfoTag

Tag signature: ‘minf’ (6d696e66h).

Permitted tag type: structType of type measurementInfo.

This tag defines measurement conditions for the colorimetric and/or spectral PCS (defined by PCS or spectralPCS fields of the profile header respectively). If this tag is not present the measurement conditions shall be assumed to have white backing, zero flare, 0°:45° geometry, using M1 illumination (ISO 13655).

NOTE Unless otherwise specified, this tag is informative only with no CMM processing associated with the contents of this tag.

9.2.87 measurementInputInfoTag

Tag signature: ‘miin’ (6d69696eh).

Permitted tag type: structType of type measurementInfo.

This tag defines measurement information for measurements related to values on input side of abstract profiles (defined by device field of the profile header). This tag may therefore only be present if the profile is an abstract profile.

If this tag is not present the measurement conditions shall be assumed to have white backing, zero flare, 0°:45° geometry, using M1 illumination (ISO 13655).

NOTE Unless otherwise specified, this tag is informative only with no CMM processing associated with the contents of this tag.

9.2.88 mediaWhitePointTag

Tag signature: 'wtpt' (77747074h).

Permitted tag type: XYZType.

This tag, which is used for generating the ICC-absolute colorimetric intent, specifies the chromatically adapted nCIEXYZ tristimulus values of the media white point. It is used for generating either the ICC-absolute colorimetric intent using an ICC-relative intent tag when an ICC-absolute colorimetric intent tag is not used or the ICC-relative colorimetric intent using an ICC-absolute intent tag when an ICC-relative colorimetric intent tag is not used. When the measurement data used to create the profile were specified relative to an adopted white with a chromaticity different from that of the PCS adopted white, the media white point nCIEXYZ values shall be adapted to be relative to the PCS adopted white chromaticity using the chromaticAdaptationTag matrix, before recording in the tag.

It is recommended that the PCC (see 6.3.2) be configured so that the PCS uses the measurement data white chromaticity.

9.2.89 metadataTag

Tag signature: 'meta' (6d657461h).

Permitted tag type: dictType.

This tag contains a set of metadata items for the profile.

The names and values in the set shall be taken from the ICC metadata registry, available on the ICC website (http://www.color.org). Display elements should be taken from the metadata registry, as this provides common localizations.

9.2.90 MToA0Tag

Tag signature: 'M2A0' (4d324130h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to device channel values.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed so the output has the same value as the input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a value of zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the deviceColor field in the Profile header.

9.2.91 MToB0Tag

Tag signature: ‘M2B0’ (4d324230h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to colorimetric PCS values for the perceptual rendering intent.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed with the value as input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the PCS field in the Profile header.

9.2.92 MToB1Tag

Tag signature: ‘M2B1’ (4d324231h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to colorimetric PCS values for the media-relative rendering intent.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed with the value as input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the PCS field in the Profile header.

If this tag is not present then relative colorimetric processing shall be performed by using the absolute MToB3Tag and then adjusting the PCS values by the media white point.

9.2.93 MToB2Tag

Tag signature: ‘M2B2’ (4d324232h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to colorimetric PCS values for the saturation rendering intent.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed with the value as input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the PCS field in the Profile header.

9.2.94 MToB3Tag

Tag signature: 'M2B3' (4d324233h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to colorimetric PCS values for the absolute rendering intent.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed with the value as input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the PCS field in the Profile header.

If this tag is not present then absolute colorimetric processing shall be performed by using the relative MToB1Tag and then adjusting the PCS values by the spectral media white point.

9.2.95 MToS0Tag

Tag signature: 'M2S0' (4d325330h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to spectral PCS values for the perceptual rendering intent.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed with the value as input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the spectralPCS field in the Profile header.

9.2.96 MToS1Tag

Tag signature: 'M2S1' (4d325331h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to spectral PCS values for the media-relative rendering intent.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed with the value as input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the spectralPCS field in the Profile header.

If this tag is not present then relative spectral processing shall be performed by using the absolute MToS3Tag and then adjusting the PCS values by the spectral media white point.

9.2.97 MToS2Tag

Tag signature: 'M2S2' (4d325332h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to spectral PCS values for the saturation rendering intent.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed with the value as input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the spectralPCS field in the Profile header.

9.2.98 MToS3Tag

Tag signature: 'M2S3' (4d325333h).

Permitted tag type: multiProcessElementsType.

This tag provides a transformation using a multiProcessElementsType (see 10.2.16) tag that converts from multiplex channel values to spectral PCS values for the absolute rendering intent.

The number of data channels provided to the transform shall match the number of channels associated with the MCS field in the Profile header. MCS connection shall result in multiplex channel values for channels with matching multiplex type identifications (see 9.2.85). Channels that have no multiplex type identification match with the source MCS shall be processed with the value as input from the associated channel in the multiplexDefaultValuesTag (see 9.2.84) or a zero multiplex channel value if this tag is not present.

The number of data channels resulting from the transform shall match the number of channels defined by the spectralPCS field in the Profile header.

If this tag is not present then absolute spectral processing shall be performed by using the relative MToB1Tag and then adjusting the PCS values by the media white point.

9.2.99 namedColorTag

Tag signature: 'nmcl' (6e6d636ch).

Permitted tag type: tagArrayType as a namedColorArray.

Named colour information is provided as a namedColorArray (see 13.2.1) defined as a tagArrayType of tintZeroStructure and namedColorStructure elements. Information related to a named colour can include PCS and as optional device representation for a list of named colours. The first element in the array shall be a tintZeroStructure which corresponds to colour values when a zero tint of any named colour is used. See 12.2.7 for a complete description of contents and usage of a tintZeroStructure. Succeeding elements shall be defined as a namedColorStructure. See 12.2.5 for a complete description of contents and usage of a namedColorStructure.

9.2.100 perceptualRenderingIntentGamutTag

Tag signature: ‘rig0’ (72696730h).

Permitted tag type: signatureType.

There is only one standard reference medium gamut, defined according to ISO 12640-3. When the signature is present, the specified gamut is defined to be the reference medium gamut for the PCS side of both the AToB0 and BToA0 tags, if they are present. If this tag is not present the perceptual rendering intent reference gamut is unspecified.

The standard PCS reference medium gamut signatures that shall be used are listed in Table 35:

Table 35 — Perceptual rendering intent gamut

Name	Signature	Hexidecimal encoding
Perceptual reference medium gamut	‘prmg’	70726D67h

Because the perceptual intent is the typical default rendering intent, the PRMG should be considered for this rendering intent.

NOTE It is possible that the ICC will define other signature values in the future.

9.2.101 profileDescriptionTag

Tag signature: ‘desc’ (64657363h).

Permitted tag type: multiLocalizedUnicodeType.

Structure containing invariant and localizable versions of the profile description for display. The content of this structure is described in 10.2.15. This invariant description has no fixed relationship to the actual profile disk file name.

NOTE It is helpful if an identification of the characterization data that was used in the creation of the profile is included in the profileDescriptionTag (e.g. "based on CGATS TR 001"[3]).

9.2.102 profileSequenceInformationTag

Tag signature: ‘psin’ (7073696eh).

Permitted tag type: tagArrayType with an array type identifier of 'pinf' (70696e66h).

The profileSequenceInformationTag shall contain a profileInfoArray (see 13.2.2) which contains an array of profileInfoStructure structures that each contain information about a profile. The successive elements of the array provide a description of the successive profiles in a sequence from source to destination. The profileSequenceInformation tag is typically used with the DeviceLink profile. See 12.2.6 for a complete description of contents and usage of a profileInfoStructure.

9.2.103 referenceNameTag

Tag Signature: ‘rfnm’ (72666e6dh).

Tag Type: utf8Type.

This text shall contain the Reference name for the three component encoding. This may correspond to the Reference Name field in the 3-component colour encoding registry on the ICC website (http://www.color.org).

When the three component colour encoding profile utilizes a standardized colour space encoding, the elements of the colorEncodingParamsTag can be assumed and any elements existing in the colorEncodingParamsTag shall be considered as overrides of the default values.

If the referenceName tag solely contains the text “ISO 22028-1” (quotes excluded) then the profile shall uniquely define the necessary parameters in the colorEncodingParamsTag. In this case the colorEncodingParamsTag shall be included and all elements shall be fully defined for the colour space. Additionally, the colorSpaceNameTag shall exist and define the assumed reference name for the colour space encoding.

9.2.104 saturationRenderingIntentGamutTag

Tag signature: ‘rig2’ (72696732h).

Permitted tag type: signatureType.

This tag is fully specified by ISO 15076-1.

9.2.105 spectralViewingConditionsTag

Tag signature: ‘svcn’ (7376636eh).

Permitted Tag types: spectralViewingConditionsType.

The reference colorimetric observer and the reference illuminant are defined in this tag. When this tag is present it describes the viewing conditions associated with both the colorimetric and spectral PCS. The content of this structure is described in 10.2.21.

The colorimetric observer type and illuminant type fields of this structure provide information that shall be used for the purposes of matching viewing conditions of profiles and determining the PCS conversion transforms to use for PCS processing. The correlated colour temperature field is also used for the purposes of matching viewing conditions when the illuminant type value is “Black body defined by CCT “ (00000009h) or “Daylight defined by CCT “ (0000000Ah).

9.2.106 spectralWhitePointTag

Tag signature: ‘swpt’ (73777074h).

Permitted tag type: float16ArrayType, float32ArrayType, uInt16ArrayType.

This tag is required when the spectral PCS is non-zero to define the PCS for the white point associated with the data in the profile. The number of entries in the array shall match the number of entries implied by the spectral PCS that is being used.

This tag is used when converting absolute spectral measurement data to relative spectral measurement data or relative spectral measurement data to absolute spectral measurement data.

9.2.107 standardToCustomPccTag

Tag signature: 's2cp' (73326370h).

Permitted Tag types: multiProcessElementsType.

This tag provides the transform needed to convert from the colorimetry defined by the1931 standard colorimetric observer with a D50 illuminant to the colorimetry defined by the observer and illuminant defined in the spectralViewingConditionsTag. The multiProcessElementsType structure shall define a sequence of one or more transforms that performs this conversion.

The number of both the input and output channels of the transform shall be three.

9.2.108 surfaceMapTag

Tag signature: ‘smap’ (736d6170h).

Permitted tag type: embeddedNormalImageType or embeddedHeightImageType.

This tag allows a normal map or height map to be associated with surface characteristics of all colours specified by the encapsulating profile.

9.2.109 technologyTag

Tag signature: ‘tech’ (74656368h).

Permitted tag type: signatureType.

Values for this tag are specified by either ISO 15076-1 or separate profile class specifications.

10 Tag type definitions

10.1 General

All tags, including private tags, shall have as their first four bytes a tag type signature to identify to profile readers what kind of data are 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 to 7) are reserved for future expansion and shall be 0 in this document. The tag signature for all private tags and any tag type signature not defined in Clause 10 shall be registered with the ICC (see Clause 5) in order to prevent signature collisions.

One or more tag types are associated with each tag defined in 9.2. The tag type definitions in 10.2 specify the data structure that shall be used in creating the contents of the tag data element for each tag.

All tag data elements, including those of private tags, shall have a tag type signature in bytes 0 to 3. Bytes 4 to 7 are reserved for future expansion and shall be 0.

Any private tag types used shall be registered with the ICC to prevent tag type signature collisions.

NOTE An effort was made to make sure 1-byte, 2-byte and 4-byte data lies on 1-byte, 2-byte and 4-byte boundaries, respectively. To achieve this, extra spaces indicated with “reserved for padding” are included in some tag type definitions.

Where not otherwise specified, value 0 is defined to imply “unknown value” for all enumerated data structures.

Where not otherwise specified, the least-significant 16 bits of all 32-bit flags in the type descriptions below are reserved for use by the ICC.

In many of the tables shown in Clause 10 the following syntax is used in the encoding column for the various numeric types listed in 4.2: numeric type[*X*] where *X* represents the number of values in that position. Where [...] is used the number of values depends on the number of channels in the tag type or number of entries in a table.

10.2 Specific tag type listing

10.2.1 colorantOrderType

This is an optional tag that specifies the laydown order in which colorants are printed on an n-colorant device. The laydown order may be the same as the channel generation order listed in the colorantTableTag or the channel order of a colour encoding type such as CMYK, in which case this tag is not needed. When this is not the case (for example, ink-towers sometimes use the order KCMY), this tag

may be used to specify the laydown order of the colorants. When used, the byte assignments shall be as given in Table 36.

Table 36 — colorantOrderType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'clro' (636c726fh) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 11	4	Count of colorants (*n*)	uInt32Number
12	1	Number of the colorant to be printed first.	uInt8Number
13 to (11+*n*)	*n*−1	The remaining *n*−1 colorants are described in a manner consistent with the first colorant	uInt8Number

The size of the array is the same as the number of colorants. The first position in the array contains the number of the first colorant to be laid down, the second position contains the number of the second colorant to be laid down, and so on, until all colorants are listed.

When this tag is used, the "count of colorants" shall be in agreement with the data colour space signature of 7.2.8.

10.2.2 curveType

The curveType contains a 4-byte count value and a one-dimensional table of 2-byte values. When used, the byte assignment shall be as given in Table 37.

Table 37 — curveType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'curv' (63757276h) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 11	4	Count value specifying the number of entries (*n*) that follow	uInt32Number
12 to end	2 *n*	Actual curve values starting with the zeroth entry and ending with the entry *n*−1	uInt16Number [...][a]

[a] If *n* = 1 the field length is 1 and the value is encoded as a u8Fixed8Number.

The curveType embodies a one-dimensional function that maps an input value in the domain of the function to an output value in the range of the function. The domain and range values shall be in the range of 0,0 to 1,0.

— when *n* is equal to 0 an identity response is assumed;

— when *n* is equal to 1, then the curve value shall be interpreted as a gamma value, encoded as a u8Fixed8Number. Gamma shall be interpreted as the exponent in the equation $y = x^{\gamma}$ and not as an inverse;

— when *n* is greater than 1 the curve values (which embody a sampled one-dimensional function) shall be defined as follows:

The first entry shall be located at 0,0, the last entry at 1,0, and intermediate entries shall be uniformly spaced using an increment of 1,0 ÷ (*n* − 1). These entries shall be encoded as uInt16Numbers (i.e. the values represented by the entries in the range 0,0 to 1,0 shall be encoded in the range 0 to 65 535). Function values between the entries shall be obtained through linear interpolation.

If the input is PCSXYZ, 1 + (32 767/32 768) shall be mapped to the value 1,0. If the output is PCSXYZ, the value 1,0 shall be mapped to 1 + (32 767/32 768).

10.2.3 dataType

The dataType is a simple data structure that contains either UTF8 or binary data, i.e. utf8Type data or transparent 8-bit bytes. The length of the string is obtained by subtracting 12 from the tag data element size portion of the tag itself as defined in 6.3.5. If this type is used for UTF8 data, it shall be terminated with a 00h byte. When used, the byte assignment shall be as given in Table 38.

NOTE This represents an extension of the dataType in ISO 15076-1 that uses ASCII encoding for text. Since ASCII encoding is a proper subset of UTF8 encoding the use of ASCII encoding has been replaced with UTF8 encoding for text-based data.

Table 38 — dataType encoding

Byte position	Field length (bytes)	Content
0 to 3	4	'data' (64617461h) type signature
4 to 7	4	Reserved, shall be 0
8 to 11	4	Data flag, 00000000h represents UTF8 data, 00000001h represents binary data, other values are reserved for future use
12 to end	(tag data element size) – 12	A string of ((tag data element size) – 12) UTF8 characters or ((tag data element size) – 12) bytes

10.2.4 dateTimeType

This dateTimeType is a 12-byte value representation of the time and date. The actual values are encoded as a dateTimeNumber described in 4.2.1.2. When used, the byte assignment shall be as given in Table 39.

Table 39 — dateTimeType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'dtim' (6474696Dh) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 19	12	Date and time	dateTimeNumber

10.2.5 dictType

The dictTypeStructure contains a dictionary array of name-value pairs with each name being uniquely associated with a single value. Each name and value can optionally be associated with localized text strings for display purposes.

The byte assignment and encoding shall be as given in Table 40 and Table 41.

Table 40 — dictType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'dict' (64696374h) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 11	4	Number of name-value records (M)	uInt32Number
12 to 15	4	The length of each name-value record, in bytes (N). The value shall be 16, 24, or 32	uInt32Number

Table 40 *(continued)*

Byte position	Field length (bytes)	Content	Encoded as…
16 to 15+N	N	The first name-value record	See Table 41
16+N to 15+M*N	N*(M-1)	Additional name-value records as needed	
16+M*N to end	(tag data element size) – (16+M*N)	Storage area of strings of Unicode characters and multiLocalizedType tags	

Table 41 — Name-Value record structure

Byte position	Field length (bytes)	Content	Encoded as…
0 to 7	8	Name string position of UTF16 text array	positionNumber
8 to 15	8	Value string position of UTF-16 text array	positionNumber
16 to 23	8	Display name element position of multiLocalizedType tag element	positionNumber
24 to 31	8	Display value element position of multiLocalizedType tag element	positionNumber

The value in the length of each name-value record filed (N) shall determine how many entries shall be present in each name-value record.

— When the length value is 16, each name-value record shall be 16 bytes long and only the positionNumber values for the name and value items shall be present.

— When the length value is 24, each name-value record shall be 24 bytes long and only the positionNumber values for the name, value and display name items shall be present.

— When the length value is 32, each name-value record shall be 32 bytes long and the positionNumber values for the name, value, display name, and display value items shall be present.

In the general use of dictType, there may be no localized values, so a name-value record length of 16 would be appropriate. In other use cases, localized display values are needed, and 32 would be used. When using localization for value fields and not localizing names, use 32-byte name-value records with the display name positionNumber fields set to zero.

A name string shall be present for each name-value record and name string positionNumber size shall be greater than zero. Other data items referenced by the name-value record are optional according to dictType, although particular dictType tag definitions my impose restrictions.

Both the name string and value string shall be encoded as UTF-16 strings and shall NOT be zero terminated.

Name strings shall contain at least one Unicode character, and the string contents of each name string shall be unique within a dictTypeTag. In general, a zero-length string (NUL) is valid for value strings, and shall be indicated by a non-zero value string positionNumber offset and a value string positionNumber size equal to zero.

NOTE Value string = NUL may be restricted for particular dictType tags.

A positionNumber offset of zero shall indicate that the corresponding data item is not present as it is undefined. When a positionNumber offset is zero, the meaning of the corresponding positionNumber size is undefined and shall be zero. When a localized display name or display value positionNumber is undefined (positionNumber offset equal to zero), no translation is provided for the corresponding name string or value string, and the name string or value string may be displayed. This is equivalent to the behaviour for all name string and value strings when the name-value record length is 16.

Alternatively, a defined display name element positionNumber offset (non-zero) with a display name element positionNumber size equal to zero indicates that the name string is not intended for display. Similarly a defined display value element positionNumberOffset (non-zero) with a display value element size equal to zero indicates that the value string, if provided, is not intended for display. A localized display value may be provided without a localized display name.

Data may be shared between the name-value records of a dictType tag. For example, the offsets for the value strings can be identical, as well as the offsets for display value elements can be identical.

The following pseudocode can be used to determine string validity where pos is value string positionNumber, display name positionNumber, or display value positionNumber:

If pos.offset == 0

Then item is undefined (pos.length can be ignored when pos.offset is zero)

Else

If ((pos.offset >= 20+N*M) && ((pos.length>= minSizeofItemType) || (Length=0)) &&

(pos.offset + pos.length <=end+1)

Then item is defined.

If ((item == value string) &&(pos.length==0)

Then value string is NUL string

Else if ((item== display name element) && (pos.length)==0)

Then name string is not for display use and no display name is provided

Else if ((item==display name element) && pos.length)==0)

Then value string is not for display use and no display value is provided

Else

THROW_ERROR("pos.offset is not zero and pos.offset or pos.length are invalid")

Unless otherwise stated, numbers shall be encoded in the string value as follows:

— A number shall be encoded as zero or more blanks and/or tabs, an optional '+' or '−' sign, a string of decimal digits that contain one decimal point '.', and an optional exponent part. The exponent part shall consist of 'e' or 'E', an optional '+' or '−' sign, and one or two decimal digits. The exponent shall indicate a power of 10.

— Multiple numbers stored in a single string shall be separated by one comma ',' between adjacent numbers.

10.2.6 embeddedHeightImageType

This type provides support for embedding an image that defines a height map. The structure encoding shall be as given in Table 42.

Table 42— embeddedHeightImageType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'ehim' (6568696dh) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 11	4	Seamless indicator	See Table 43
12 to 15	4	Height image encoding format	See Table 44
16 to 19	4	Height in meters of minimum pixel value	float32Number
20 to 23	4	Height in meters of maximum pixel value	float32Number
24 to end	(tag data element size) – 12	Height image data	

The displacement image data can be created so that when the displacement map is tiled across a surface it has no visible seams. The Seamless indicator field indicates if the displacement image is seamless and has the values as given in Table 43.

Table 43 — Displacement image type

Image Type	Value
Not seamless	0
Seamless	1

The image data shall be encoded using the image file format defined by the Displacement Image Format field which can have the values given in Table 44.

Table 44 — Displacement image encoding formats

Image Encoding Format	Value
PNG	0
TIFF	1

The contents of a Displacement Image shall be encoded as greyscale pixels used to identify the height of the displacement. A pixel with a minimum pixel value shall have a displacement equal to the height in meters defined by the height in meters of minimum pixel value field. A pixel with a maximum pixel value shall have a displacement equal to the height in meters defined by the height in meters of maximum pixel value field.

The physical dimensions of the pixels in the image shall be encoded by using the appropriate encoding mechanisms of the image-encoding format. The PNG format uses the pHYs chunk to specify the physical size of the image. The TIFF format uses the perResolutionUnit, XResolution and YResolution tags to specify the physical dimensions.

10.2.7 embeddedNormalImageType

This type provides support for embedding an image that defines a normal map. The structure encoding shall be as given in Table 45.

Table 45 — embeddedNormalImageType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'enim' (656e696dh) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 11	4	Seamless indicator	See Table 46
12 to 15	4	Normal image encoding format	See Table 47
16 to end	(tag data element size) − 12	Normal image data	

The normal image data can be created so that when the normal map is tiled across a surface it has no visible seams. The Seamless indicator field indicates if the normal image is seamless and has the values as given in Table 46.

Table 46 — Normal image type

Image type	Value
Not seamless	0
Seamless	1

The image data shall be encoded using the image file format defined by the Normal Image Format field which can have the values as given in Table 47.

Table 47 — Normal image encoding formats

Image encoding format	Value
PNG	0
TIFF	1

The contents of a Normal Image shall be encoded as RGB pixels used to identify XYZ direction of the normal vector for each point in the image. RGBs are mapped to XYZ directions as follows:

1) red maps from (0 − maximum red value) to X (−1,0 – 1,0);
2) green maps from (0 − maximum green value) to Y (−1,0 – 1,0);
3) blue maps from (0 − maximum blue value) to Z (0,0 – 1,0).

Since normals point towards the observer, negative values of Z are not encoded. The maximum values for the red, green, and blue channels can be found by accessing the appropriate fields of the PNG and TIFF files. The length of the vector specified by the XYZ direction shall be equal to 1,0.

The physical dimensions of the pixels in the image shall be encoded by using the appropriate encoding mechanisms of the image-encoding format. The PNG format uses the pHYs chunk to specify the physical size of the image. The TIFF format uses the perResolutionUnit, XResolution and YResolution tags to specify the physical dimensions.

10.2.8 float16ArrayType

This type represents an array of generic 16-bit encoded half-precision floating point values. The number of values is determined from the size of the tag.

When used, the byte assignment and encoding shall be as given in Table 48.

Table 48 — float16ArrayType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'fl16' (666c3136h) type signature	
4..7	4	Reserved, shall be 0	
8..end	2N	An array of 16-bit half-precision floating point numbers	float16Number[...]

10.2.9 float32ArrayType

This type represents an array of generic 32-bit encoded single-precision floating point numbers values. The number of values is determined from the size of the tag.

When used, the byte assignment and encoding shall be as given in Table 49.

Table 49 — float32ArrayType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'fl32' (666c3332h) type signature	
4..7	4	Reserved, shall be 0	
8..end	4N	An array of 32-bit single-precision floating point numbers	float32Number[...]

10.2.10 float64ArrayType

This type represents an array of generic 64-bit encoded double-precision floating point numbers values. The number of values is determined from the size of the tag.

When used, the byte assignment and encoding shall be as given in Table 50.

Table 50 — float64ArrayType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'fl64' (666c3634h) type signature	
4..7	4	Reserved, shall be 0	
8..end	8N	An array of 64-bit double-precision floating point numbers	float64Number[...]

10.2.11 gamutBoundaryDescriptionType

The GamutBoundaryDescriptionType structure encodes a collection of vertices and faces that describe a gamut boundary. The vertices contain a PCS value and an optional device value. The faces contain a list of vertex IDs. The order of the vertex IDs shall be clockwise when viewed from outside of the gamut. The encoding shall be as shown in Table 51.

Table 51 — Gamut boundary description encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'gbd ' (67626420h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Number of PCS channels (P)	uInt16Number
10..11	2	Number of device channels (Q)	uInt16Number

Table 51 *(continued)*

Byte position	Field length (bytes)	Content	Encoded as...
16..19	4	Number of faces (F)	uInt32Number
20 .. 19+F*12	F*12	Array of vertex IDs for each face	uInt32Number
20+F*12 .. 19+F*12+V*P*4	V*P*4	Array of PCS coordinates for each vertex	float32Number
20+F*12+V*P*4 ... end	V*Q*4	Array of device coordinates for each vertex	float32Number

The number of PCS channels (P) shall be three or greater. The number of output channels (Q) can be zero if device values are not included.

The number of vertices shall be four or greater.

The number of faces shall be four or greater.

The array of vertex IDs is an array that specifies the IDs of each vertex of each face. The array is organized so that the three IDs of the first face are specified first, the three IDs of the second face next, and so on. The ID of the vertex is a number that shall be between 0 and V-1. This ID corresponds to the order of the vertices in the vertex array.

The array of vertex PCS values contains one PCS value for each vertex. The order of the vertices corresponds with the vertex IDS from the face description. The range of the Output Channels is the range of values that can be represented as float32Number.

The optional array of device values contains one device value for each vertex. The order of the vertices corresponds with the vertex IDS from the face description. The range of the Output Channels is the range of values that can be represented as float32Number.

The set of faces should constitute a closed volume.

NOTE Euler's formula can be used to verify that the volume is closed.

Annex B provides details of the encoding and use of a gamutBoundaryDescriptionType.

10.2.12 lutAToBType

10.2.12.1 General

This structure represents a colour transform. The type contains up to five processing elements that are stored in the AToBTag tag in the following order: a set of one-dimensional curves; a 3 × 3 matrix with offset terms; a set of one-dimensional curves; a multi-dimensional lookup table; and a set of one-dimensional output curves. Data are processed using these elements via the following sequence:

("A" curves) ⇒ (multi-dimensional lookup table, CLUT) ⇒ ("M" curves) ⇒ (matrix) ⇒ ("B" curves).

NOTE The processing elements are not in this order in the tag to allow for simplified reading and writing of profiles.

It is possible to use any or all of these processing elements. At least one processing element shall be included. Only the following combinations are permitted:

— B;

— M, Matrix, B;

— A, CLUT, B;

— A, CLUT, M, Matrix, B.

Other combinations may be achieved by setting processing element values to identity transforms. The domain and range of the A and B curves and CLUT are defined to consist of all real numbers between 0,0 and 1,0 inclusive. The first entry is located at 0,0, the last entry at 1,0, and intermediate entries are uniformly spaced using an increment of 1,0/(m−1). For the A and B curves m is the number of entries in the table. For the CLUT M is the number of grid points along each dimension. Since the domain and range of the tables are 0,0 to 1,0 it is necessary to convert all device values and PCSLAB values to this numeric range. It shall be assumed that the maximum value in each case is set to 1,0 and the minimum value to 0,0 and all intermediate values are linearly scaled accordingly.

When using this type, it is necessary to assign each data colour space component to an input and output channel. The channel order shall be the same as that associated with the colour space signature (see 7.2.8 and 7.2.9)

When used, the byte assignment and encoding shall be as given in Table 52.

Table 52 — lutAToBType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'mAB ' (6D414220h) [multi-function A-to-B table] type signature	
4 to 7	4	Reserved, shall be 0	
8	1	Number of Input Channels (i)	uInt8Number
9	1	Number of Output Channels (o)	uInt8Number
10 to 11	2	Reserved for padding, shall be 0	
12 to 15	4	Offset to first "B" curve	uInt32Number
16 to 19	4	Offset to matrix	uInt32Number
20 to 23	4	Offset to first "M" curve	uInt32Number
24 to 27	4	Offset to CLUT	uInt32Number
28 to 31	4	Offset to first "A" curve	uInt32Number
32 to end	Variable	Data	

Each curve and processing element shall start on a 4-byte boundary. To achieve this, each item shall be followed by up to three 00h pad bytes as needed.

Curve data elements may be shared. For example, the offsets for A, B and M curves can be identical.

The offset entries (bytes 12 to 31) point to the various processing elements found in the tag. The offsets indicate the number of bytes from the beginning of the tag to the desired data. If any of the offsets are zero, it is an indication that processing element is not present and the operation is not performed.

This tag type may be used independent of the value of the PCS field specified in the header.

10.2.12.2"A" curves

The number of "A" curves is the same as the number of input channels. The "A" curves may only be used when the CLUT is used. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "A" curve is stored as an embedded curveType or a parametricCurveType (see 10.2.2 or 10.2.17). The length is as indicated by the specification of the respective curve type. Note that the entire tag type, including the tag type signature and reserved bytes, is included for each curve.

10.2.12.3CLUT

The CLUT appears as an n-dimensional array, with each dimension having a number of entries corresponding to the number of grid points.

The CLUT values are arrays of 8-bit or 16-bit unsigned values, normalized to the range of 0 to 255 or 0 to 65 535.

The CLUT is organized as an *i*-dimensional array with a variable number of grid points in each dimension, where *i* is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is an *o*-integer array, where *o* is the number of output channels. The first sequential integer of the entry contains the function value for the first output function, the second sequential integer of the entry contains the function value for the second output function and so on until all of the output functions have been supplied. The size of the CLUT in bytes is (nGrid1 × nGrid2 ×…× nGridN) × number of output channels (*o*) × size of (channel component).

When used, the byte assignment and encoding for the CLUT shall be as given in Table 53.

Table 53 — lutAToBType CLUT encoding

Byte position	Field length (bytes)	Content	Encoded as…
0 to 15	16	Number of grid points in each dimension. Only the first *i* entries are used, where *i* is the number of input channels. Unused entries shall be 00h.	uInt8Number[16]
16	1	Precision of data elements in bytes. Shall be either 01h or 02h.	uInt8Number
17 to 19	3	Reserved for padding, shall be 0	
20 to end	Variable	CLUT data points (arranged as described in the text).	uInt8Number [...] or uInt16Number [...]

If the number of input channels does not equal the number of output channels, the CLUT shall be present.

If the number of grid points in a one-dimensional curve, or in a particular dimension of the CLUT, is two, the data for those points shall be set so that the correct results are obtained when linear interpolation is used to generate intermediate values.

10.2.12.4 "M" curves

When present, the number of "M" curves shall be the same as the number of output channels. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "M" curve is stored as an embedded curveType or a parametricCurveType (see 10.2.2 or 10.2.17). The length is as indicated by the specification of the respective curve type. Note that the entire tag type, including the tag type signature and reserved bytes, is included for each curve. The "M" curves may only be used when the matrix is used.

10.2.12.5 Matrix

The matrix is organized as a 3 × 4 array. The elements appear in order from e_1–e_{12}. The matrix elements are each s15Fixed16Numbers, as shown in Formula (5):

$$\text{array} = [e_{11}, e_{12}, \ldots, e_{1P}, e_{21}, e_{22}, \ldots, e_{2P}, \ldots, e_{Q1}, e_{Q2}, \ldots, e_{QP}, e_1, e_2, \ldots, e_Q] \quad (5)$$

The matrix is used to convert data to a different colour space, according to Formula (6):

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \ldots \\ Y_Q \end{bmatrix} = \begin{bmatrix} e_{11} & e_{12} & \ldots & e_{1P} \\ e_{21} & e_{22} & \ldots & e_{2P} \\ \ldots & \ldots & \ldots & \ldots \\ e_{Q1} & e_{Q2} & & e_{QP} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ \ldots \\ X_P \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \ldots \\ e_Q \end{bmatrix} \quad (6)$$

The range of input values X_1, X_2 and X_3 is 0,0 to 1,0. The resultant values Y_1, Y_2 and Y_3 shall be clipped to the range 0,0 to 1,0 and used as inputs to the "B" curves.

10.2.12.6 "B" curves

The number of "B" curves shall be the same as the number of output channels. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "B" curve is stored as an embedded curveType or a parametricCurveType (see 10.2.2 or 10.2.17). The length is as indicated by the specification of the respective curve type. Note that the entire tag type, including the tag type signature and reserved bytes, are included for each curve.

10.2.13 lutBToAType

10.2.13.1 General

This structure represents a colour transform. The type contains up to five processing elements which are stored in the BToATag in the following order: a set of one-dimensional curves; a 3 × 3 matrix with offset terms; a set of one-dimensional curves; a multi-dimensional lookup table; and a set of one-dimensional curves. Data are processed using these elements via the following sequence:

("B" curves) ⇒ (matrix) ⇒ ("M" curves) ⇒ (multi-dimensional lookup table, CLUT) ⇒ ("A" curves).

It is possible to use any or all of these processing elements. At least one processing element shall be included. Only the following combinations are permitted:

— B;

— B, Matrix, M;

— B, CLUT, A;

— B, Matrix, M, CLUT, A.

Other combinations may be achieved by setting processing element values to identity transforms. The domain and range of the A and B curves and CLUT are defined to consist of all real numbers between 0,0 and 1,0 inclusive. The first entry is located at 0,0, the last entry at 1,0, and intermediate entries are uniformly spaced using an increment of 1,0/(m−1). For the A, M and B curves m is the number of entries in the table. For the CLUT m is the number of grid points along each dimension. Since the domain and range of the tables are 0,0 to 1,0 it is necessary to convert all device values and PCSLAB values to this numeric range. It shall be assumed that the maximum value in each case is set to 1,0 and the minimum value to 0,0 and all intermediate values are linearly scaled accordingly.

When using this type, it is necessary to assign each data colour space component to an input and output channel. The channel order shall be the same as that associated with the colour space signature (see 7.2.8 and 7.2.9)

When used, the byte assignment and encoding shall be as given in Table 54.

Table 54 — lutBToAType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'mBA' (6D424120h) [multi-function BToA table] type signature	
4 to 7	4	Reserved, shall be 0	
8	1	Number of Input Channels (*i*)	uInt8Number
9	1	Number of Output Channels (*o*)	uInt8Number
10-11	2	Reserved for padding, shall be 0	
12 to 15	4	Offset to first "B" curve	uInt32Number
16 to 19	4	Offset to matrix	uInt32Number
20 to 23	4	Offset to first "M" curve	uInt32Number
24 to 27	4	Offset to CLUT	uInt32Number
28 to 31	4	Offset to first "A" curve	uInt32Number
32 to end	Variable	Data	

Each curve and processing element shall start on a 4-byte boundary. To achieve this, each item may be followed by up to three 00h pad bytes as needed.

Curve data elements may be shared. For example, the offsets for A, B and M curves may be identical.

The offset entries (bytes 12 to 31) point to the various processing elements found in the tag. The offsets indicate the number of bytes from the beginning of the tag to the desired data. If any of the offsets are zero, it is an indication that processing element is not present and the operation is not performed.

This tag type shall only be used when the PCS field in the header specifies either PCSXYZ or PCSLAB.

10.2.13.2"B" curves

The number of "B" curves is the same as the number of input channels. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "B" curve is stored as an embedded curveType tag or a parametricCurveType (see 10.2.2 or 10.2.17). The length is as indicated by the specification of the curve type. Note that the entire tag type, including the tag type signature and reserved bytes, is included for each curve.

10.2.13.3Matrix

The matrix is organized as a 3 × 4 array. The elements of the matrix appear in the type in order from e_1 to e_{12}. The matrix elements are each s15Fixed16Numbers, as shown in Formula (7):

$$\text{array} = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}] \tag{7}$$

The matrix is used to convert data to a different colour space, according to Formula (8):

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} + \begin{bmatrix} e_{10} \\ e_{11} \\ e_{12} \end{bmatrix} \tag{8}$$

The range of input values X_1, X_2 and X_3 is 0,0 to 1,0. The resultant values Y_1, Y_2 and Y_3 shall be clipped to the range 0,0 to 1,0 and used as inputs to the "M" curves.

The matrix is permitted only if the number of output channels, or "M" curves, is three.

10.2.13.4"M" curves

When present, the number of "M" curves shall be the same as the number of input channels. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "M" curve is stored as an embedded curveType or a parametricCurveType (see 10.2.2 or 10.2.17). The length is as indicated by the specification of the proper curve type. Note that the entire tag type, including the tag type signature and reserved bytes, are included for each curve. The "M" curves may only be used when the matrix is used.

10.2.13.5CLUT

The CLUT appears as an *n*-dimensional array, with each dimension having a number of entries corresponding to the number of grid points.

The CLUT values are arrays of 8-bit or 16-bit unsigned values, normalized to the range of 0 to 255 or 0 to 65 535.The CLUT is organized as an *i*-dimensional array with a variable number of grid points in each dimension, where *i* is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is an *o*-integer array, where *o* is the number of output channels. The first sequential integer of the entry contains the function value for the first output function, the second sequential integer of the entry contains the function value for the second output function and so on until all of the output functions have been supplied. The size of the CLUT in bytes is (nGrid1 × nGrid2 ×…× nGridN) × number of output channels (*o*) × size of (channel component).

When used, the byte assignment and encoding for the CLUT shall be as given in Table 55.

Table 55 — lutBToAType CLUT encoding

Byte position	Field length (bytes)	Content	Encoded as…
0 to15	16	Number of grid points in each dimension. Only the first *i* entries are used, where *i* is the number of input channels. Unused entries shall be 00h.	uInt8Number[16]
16	1	Precision of data elements in bytes. Shall be either 01h or 02h.	uInt8Number
17 to 19	3	Reserved for padding.	
20 to end	Variable	CLUT data points (arranged as described in the text).	uInt8Number [...] or uInt16Number [...]

If the number of grid points in a one-dimensional curve, or in a particular dimension of the CLUT, is two, the data for those points shall be set so that the correct results are obtained when linear interpolation is used to generate intermediate values.

If the number of input channels does not equal the number of output channels, the CLUT shall be present.

10.2.13.6"A" curves

When present, the number of "A" curves shall be the same as the number of output channels. The "A" curves may only be used when the CLUT is used. The curves are stored sequentially, with 00h bytes used for padding between them if needed. Each "A" curve is stored as an embedded curveType or a parametricCurveType (see 10.2.2 or 10.2.17). The length is as indicated by the specification of the proper curve type. Note that the entire tag type, including the tag type signature and reserved bytes, is included for each curve.

10.2.14 measurementType

This tag structure represents a backwards compatible extension of ISO 15076-1 with the same tag signature. The encoding of this type has been extended from the measurementType in ISO 15076-1. An optional element encodes the measurement condition, and additional standard illuminants have been included.

If the encoded tag structure length is only 36 bytes then the value for the measurement type shall be assumed to be zero.

The measurementType information refers only to the internal profile data and is meant to provide profile makers an alternative to the default measurement specifications. When used, the byte assignment and encoding shall be as given in Table 56.

Table 56 — measurementType structure

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'meas' (6D656173h) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 11	4	Encoded value for standard observer	see Table 57
12 to 23	12	nCIEXYZ tristimulus values for measurement backing	XYZNumber
24 to 27	4	Encoded value for measurement geometry	see Table 58
28 to 31	4	Encoded value for measurement flare	see Table 59
32 to 35	4	Encoded value for standard illuminant	see Table 60
36 to 39 (optional)	4 (optional)	Encoded measurement condition (optional extension)	see Table 61

The encoding for the standard observer field is shown in Table 57.

Table 57 — Standard observer encodings

Standard observer	Hexidecimal encoding
Unknown	00000000h
CIE 1931 standard colorimetric observer	00000001h
CIE 1964 standard colorimetric observer	00000002h

The encoding for the measurement geometry field is shown in Table 58.

Table 58 — Measurement geometry encodings

Geometry	Hexidecimal encoding
Unknown	00000000h
0°:45° or 45°:0°	00000001h
0°:d or d:0°	00000002h

The encoding for the measurement flare value is shown in Table 59, and is equivalent to the basic numeric type u16Fixed16Number in ISO 15076-1:2010, 4.2.1.7.

Table 59 — Measurement flare encodings

Flare	Hexidecimal encoding
0 (0 %)	00000000h
1,0 (or 100 %)	00010000h

The encoding for the standard illuminant field is shown in Table 60. This represents an extension of encodings found in ISO 15076-1.

Table 60 — Standard illuminant encodings

Standard illuminant	Hexidecimal encoding
Custom	00000000h
D50	00000001h
D65	00000002h
D93	00000003h
F2	00000004h
D55	00000005h
A	00000006h
Equi-Power (E)	00000007h
F8	00000008h
Black body defined by CCT	00000009h
Daylight defined by CCT	0000000Ah
B	0000000Bh
C	0000000Ch
F1	0000000Dh
F3	0000000Eh
F4	0000000Fh
F5	00000010h
F6	00000011h
F7	00000012h
F9	00000013h
F10	00000014h
F11	00000015h
F12	00000016h

The encoding for the optional ISO 13655 measurement condition value is shown in Table 61. If the length of the measurementInfo tag storage is less than 40 then the measurementType shall be assumed to be unknown (00000000h).

Table 61 — ISO 13655 measurement condition encodings

Type	Hexidecimal encoding
Unknown	00000000h
M0	00000001h
M1	00000002h
M2	00000003h
M3	00000004h

10.2.15 multiLocalizedUnicodeType

This tag structure contains a set of records each referencing a multilingual Unicode string associated with a profile. Each string is referenced in a separate record with the information about what language and region the string is for.

The byte assignment and encoding shall be as given in Table 62.

Note that the fourth field of this tag, the record size, should, for the time being, contain the value 12, which corresponds to the size in bytes of each record. Any code that needs to access the *n*-th record should determine the record's offset by multiplying *n* by the contents of this size field and adding 16. This minor extra effort allows for future expansion of the record encoding, should the need arise, without having to define a new tag type.

Multiple strings within this tag may share storage locations. For example, en/US and en/UK can refer to the same string data.

For the specification of Unicode, see *The Unicode Standard*[13] published by The Unicode Consortium or visit http://www.unicode.org. For the definition of language code and region codes, see ISO-639 and ISO 3166. The Unicode strings in storage should be encoded as 16-bit big-endian, UTF-16BE, and should not be NULL terminated.

NOTE For additional clarification on the encodings used, see the ICC technical note 01-2002 available on www.color.org.

If the specific record for the desired region is not stored in the tag, the record with the same language code should be used. If the specific record for the desired language is not stored in the tag, the first record in the tag is used if no other user preference is available.

Table 62 — multiLocalizedUnicodeType

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'mluc' (0x6D6C7563) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 11	4	Number of records (*n*)	uInt32Number
12 to 15	4	Record size: the length in bytes of every record. The value is 12.	0000000Ch
16 to 17	2	First record language code: language code specified in ISO-639	uInt16Number
18 to 19	2	First record country code: region code specified in ISO 3166	uInt16Number
20 to 23	4	First record string length: the length in bytes of the string	uInt32Number
24 to 27	4	First record string offset: the offset from the start of the tag to the start of the string, in bytes	uInt32Number
28 to 28 + [12(*n* − 1)] − 1 (or 15 + 12*n*)	12(*n* − 1)	Additional records as needed	
28 + [12(*n* − 1)] or (16 + 12*n*) to end	Variable	Storage area of strings of Unicode characters	

10.2.16 multiProcessElementsType

This structure represents a colour transform, containing a sequence of processing elements. The processing elements contained in the structure are defined in the structure itself, allowing for a flexible structure. Supported processing elements are defined in Clause 11 of this document. Other processing element types may be added in the future. Each type of processing element may be contained any

number of times in the structure. The processing elements support float32Number-encoded input and output ranges.

If undefined processing element types are present in a multiProcessElementsType tag, the multiProcessElementsType tag shall not be used and fall back behaviour shall be followed (if possible).

When using this type, it is necessary to assign each colour space component to an input and output channel. These assignments shall be as shown in Table 63.

The encoding of a multiProcessElementsType structure shall be as given in Table 63.

Table 63 — multiProcessElementsType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'mpet' (6D706574h) [multi-process elements table] type signature	
4…7	4	Reserved, shall be 0	
8..9	2	Number of input channels (F)	uInt16Number
10..11	2	Number of output channels (T)	uInt16Number
12..15	4	Number of processing elements (N)	uInt32Number
16..15+8N	8N	Process element positions table	positionNumber[...]
16+8N..end		Data	

The number of processing elements (n) shall be greater than or equal to 1. The process element positions table contains information on where and how large the process elements are. Offset locations are relative to the start of the multiProcessElementsType tag. Thus the offset of first stored process element shall be $16 + 8n$.

Each processing element shall start on a 4-byte boundary. To achieve this, each item shall be followed by up to three 00h pad bytes as needed.

It is permitted to share data between processing elements. For example, the offsets for some processing elements can be identical.

Processing elements in the multiProcessElementsType are processed in the order that they are defined in the processing elements position table. The results of a processing element are passed on to the next processing element. The last processing element provides the final result for the containing multiProcessElementsType. Therefore, the input/output channels specified by the processing elements and the containing multiProcessElementsType need to be in agreement.

The first processing element's input channels shall be the same as the input channels of the containing multiProcessElementsType. The input channels of a processing element shall be the same as the previous processing element's output channels. The last processing element's output channels shall be the same as the output channels of the containing multiProcessElementsType.

The definition of supported processing elements can be found in Clause 11 multiProcessElement Definitions.

10.2.17 parametricCurveType

The parametricCurveType describes a one-dimensional curve by specifying one of a predefined set of functions using the parameters. When used, the byte assignment and encoding shall be as given in Table 64.

Table 64 — parametricCurveType encoding

Byte position	Field length (bytes)	Content	Encoded as
0 to 3	4	'para' (70617261h) type signature	
4 to 7	4	Reserved, shall be 0	
8 to 9	2	Encoded value of the function type	uInt16Number (see Table 65)
10 to 11	2	Reserved, shall be 0	
12 to end	See Table 65	One or more parameters (see Table 65)	s15Fixed16Number [...]

The encoding for the function type field and the parameters are shown in Table 65.

Table 65 — parametricCurveType function type encoding

Field length (bytes)	Function type	Encoded value	Parameters	Note
4	$Y = X^g$	0000h	g	
12	$Y = (aX + b)^g \quad (X \geq -b/a)$ $Y = 0 \quad (X < -b/a)$	0001h	g a b	CIE 122-1966 [10]
16	$Y = (aX + b)^g + c \quad (X \geq -b/a)$ $Y = c \quad (X < -b/a)$	0002h	g a b c	IEC 61966-3
20	$Y = (aX + b)^g \quad (X \geq d)$ $Y = cX \quad (X < d)$	0003h	g a b c d	IEC 61966-2.1 (sRGB)
28	$Y = (aX + b)^g + e \quad (X \geq d)$ $Y = (cX + f) \quad (X < d)$	0004h	g a b c d e f	
NOTE More functions can be added as necessary.				

The order of the parameters in the data, Table 64, follows the left-to-right order of the parameters in Table 65.

The domain and range of each function shall be [0,0 1,0]. Any function value outside the range shall be clipped to the range of the function. When unsigned integer data are supplied as input, it shall be converted to the domain by dividing it by a factor of $(2^N) - 1$, where N is the number of bits used to represent the input data. When the output is required to be unsigned integer data, it shall be converted from the range by multiplying it by a factor of $(2^M) - 1$, where M is the number of bits used to represent the output data.

If the input is PCSXYZ, the PCSXYZ X, Y, or Z value 1+ (32 767÷32 768) shall be mapped to the function input value 1,0. If the output is PCSXYZ, the function output value 1,0 shall be mapped to the PCSXYZ X, Y, or Z value 1+ (32 767÷32 768).

NOTE The parameters selected for a parametric curve can result in complex or undefined values for the input range used. This can occur for example if d < -b/a. In such cases the behaviour of the curve is undefined.

10.2.18 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.

When used, the byte assignment and encoding shall be as given in Table 66.

Table 66 — s15Fixed16ArrayType encoding

Byte position	Field length (bytes)	Content
0 to 3	4	'sf32' (73663332h) type signature
4 to 7	4	Reserved, shall be 0
8 to end	Variable	An array of s15Fixed16Number values

10.2.19 signatureType

The signatureType contains a 4-byte sequence. Sequences of less than four characters are padded at the end with spaces, 20 h. Typically this type is used for registered tags that can be displayed on many development systems as a sequence of four characters.

When used, the byte assignment and encoding shall be as given in Table 67.

Table 67 — signatureType encoding

Byte position	Field length (bytes)	Content
0 to 3	4	'sig ' (73696720h) type signature
4 to 7	4	Reserved, shall be 0
8 to 11	4	4-byte signature

10.2.20 sparseMatrixArrayType

The sparseMatrixArrayType defines a tag type for encoding an array of sparse matrices. When used, the byte assignment and encoding shall be as given in Table 68.

Table 68 — sparseMatrixArrayType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'smat' (736d6174h) type signature	
4…7	4	Reserved, shall be 0	
8..9	2	Number of equivalent output channels used by sparse matrix encoding (Q)	uInt16Number
10..11	2	Sparse matrix LUT encoding type	sparseMatrixEncodingType
12..15	4	Number of sparse matrices in list	uInt32Number
12…end	N*B	List of (N) sparse matrices	List of compact sparseMatrixUInt8 or sparseMatrixUInt16 or sparseMatrixFloat16 or sparseMatrixFloat32

The sparse matrices encoded in the list shall all be encoded according to the value in the Sparse Matrix LUT Encoding type element.

The sparse matrices encoded in the list of sparse matrices shall use compact padding resulting in the Matrix Entry Data Values and end of each sparse matrix being aligned on a 4 byte boundary.

All sparse matrices in the sparseMatrixArrayType shall have the same number of rows and columns.

10.2.21 spectralViewingConditionsType

Spectral data are always coded equidistantly defined by a start wavelength, interval step wavelength and end wavelength such that the difference between the end wavelength and start wavelength is an integer number of interval steps.

A profile may encode both standard and custom settings for the colorimetric observer. In both cases, the observer's CMFs are stored in a 3XN matrix with N the spectral dimension defined by the fields "start wavelength colorimetric observer", "interval wavelength colorimetric observer" and "end wavelength colorimetric observer". The 3XN matrix is stored row by row, in the "Matrix colorimetric observer" field.

For object colours, both custom and standard illuminants are supported. The illuminant is specified both by its illuminant type as well as its power distribution function. When the illuminant type value is either "Black body defined by CCT" (00000009h) or "Daylight defined by CCT" (0000000Ah) the (correlated) colour temperature field is also used to define the illuminant. If the illuminant type is not one of these values then the (correlated) colour temperature field is merely informative and may be set to zero.

The power distribution of the illuminant is represented by an M-dimensional vector with M defined by the fields "start wavelength illuminant", "interval wavelength illuminant" and "end wavelength illuminant".

To remain compatible with the viewingConditions tag, the unnormalized XYZ values for the illuminant and surround are also provided, both defined in cd/m^2.

For luminous colours no illuminant is specified. In this case, the fields "start wavelength illuminant", "interval wavelength illuminant", "end wavelength illuminant" and "Vector illuminant" are replaced by the corresponding values for the white emission spectrum. And as a result the "un-normalized CIEXYZ values for illuminant" field is filled with the un-normalized CIEXYZ values for the reference white emission spectrum.

When used, the spectralViewingConditions Type byte assignment and encoding shall be as given in Table 69. Encodings for the standard observer field are provide in Table 70, and the Encodings for the standard illuminants are provided in Table 71.

Table 69 — spectralViewingConditions Type tag type

Byte position	Field length (bytes)	Content	Encoded as
0 .. 3	4	'svcn' (7376636eh) type signature	
4 .. 7	4	Reserved, shall be 0	
8 .. 11	4	Colorimetric observer type	See Table 70
12 .. 17	6	Spectral range for colorimetric observer with (N) steps	spectralRange
18..19	2	Reserved, shall be 0	
20.. 20+12*N-1	12N	Matrix colorimetric observer (X vector, then Y vector, then Z vector)	float32Number[]
20+12*N .. 23+12*N	4	Illuminant type	See Table 71

24+12*N .. 27+12*N	4	(Correlated) colour temperature	float32Number
28+12*N .. 33+12*N	6	Illuminant spectral range with (M) steps	spectralRange
34+12*N .. 35+12*N	2	Reserved, shall be 0	
36+12*N. 36+12*N+4*M-1	4M	Vector illuminant	float32Number[]
36+12*N+4*M .. 36+12*N+4*M+11	12	Un-normalized CIEXYZ values for illuminant (with Y in cd/m^2)	XYZNumber
48+12*N+4*M .. 48+12*N+4*M+11	12	Un-normalized CIEXYZ values for surround (with Y in cd/m^2)	XYZNumber

Table 70 — Standard observer encodings

Standard observer	Value
Custom colorimetric observer	00000000h
CIE 1931 standard colorimetric observer	00000001h
CIE 1964 standard colorimetric observer	00000002h

Table 71 — Illuminant encodings

Standard illuminant	Encoding
Custom	00000000h
D50	00000001h
D65	00000002h
D93	00000003h
F2	00000004h
D55	00000005h
A	00000006h
Equi-Power (E)	00000007h
F8	00000008h
Black body defined by CCT	00000009h
Daylight defined by CCT	0000000Ah
B	0000000Bh
C	0000000Ch
F1	0000000Dh
F3	0000000Eh
F4	0000000Fh
F5	00000010h
F6	00000011h
F7	00000012h
F9	00000013h
F10	00000014h
F11	00000015h
F12	00000016h

Having the ability to use custom reference viewing conditions introduces the need for additional processing by the CMM when connecting profiles that use a colorimetric-based PCS. The CMM needs to

both determine the compatibility of the implied PCS for each of the profiles and then insert the proper PCS transforms that are needed.

For both the source and destination profile, the reference observer and reference illuminant are determined in the following manner:

If the profile version is less than V5, then the CIE 1931 standard observer and a D50 illuminant is assumed. Else if no reference spectral viewing condition tag exists and the illuminant field in the profile header matches the CIE 1931 standard observer and a D50 illuminant, then the CIE 1931 standard observer and D50 illuminant are assumed. Else if a reference spectral viewing condition tag exists, then the observer and illuminant are defined by the observer and illuminant fields in the reference spectral viewing condition tag.

Note that the illuminant field in the profile header should always be in agreement with the reference spectral viewing conditions tag if available.

Once the reference observer and reference illuminant for both source and destination are determined, then the decision about what (if any) transformations are needed to connect the profiles can be made.

If both the reference observer and reference illuminant match between the two profiles then no additional transformations are needed.

Otherwise:

If the reference observer of the source profile is not the CIE 1931 standard observer or the reference illuminant of the source profile is not a D50 illuminant, then the transform from the source profile's customToStandardPccTag is first used, unless the CMM provides its own transform. If this tag is not present and the CMM does not provide an alternative, then the source profile cannot be connected.

If the reference observer of the destination profile is not the CIE 1931 standard observer or the reference illuminant of the destination profile is not a D50 illuminant, then the transform from the destination profile's standardToCustomPccTag profile is then used, unless the CMM provides its own transform. If this tag is not present and the CMM does not provide an alternative, then the profile cannot be connected.

10.2.22 tagArrayType

The tagArrayType structure encodes an array of tags that have an identical tag type. Clause 13 defines valid tag arrays with their associated array type identifiers.

The structure type indentifiers may vary when tag array elements are of tagStructType. How they vary shall be associated with the array type identifier. (See Clause 13).

The format of the tagArrayType structure can be found in Table 72.

Table 72 — tagArrayType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'tary' (74617279h) type signature	
4..7	4	Reserved, shall be 0	
8..11	4	Array Type Identifier	4-byte signature
12..15	4	Number of tag elements in array (N)	uInt32Number
16..23	8	Tag element 1 position	positionNumber
…	…	…	…
16+(N-1)*8 .. 16+N*8-1	8	Tag element N position	positionNumber
16+N*8 .. end		Tag element data	

Each tag array element has an offset and size. Each offset is relative to the beginning of the associated tagArrayType structure. Tag array elements should always begin at an offset divisible by 4 with padding between elements as needed.

The Element tag type signature shall match the signature of the tag type for all tag elements in the array.

The Element tag type signature can be the signature of any valid profile tag type.

If the Element tag type signature is 'tags' (74616773h) then the tag array is an array of tagStructType tags. In this case the Element struct type identifier shall be the same as the Struct Type Identifier (Byte position 8..11) in each of the tagStructType tags.

If the Element tag type signature is not 'tags' (74616773h) then the Element struct type identifier shall be zero (0h).

The Offset of multiple tag elements can be the same (IE tag elements can share tag data).

10.2.23 tagStructType

The tagStructType structure allows a collection of tag elements to be grouped into a single structure.

The format of the tagStructType structure can be found in Table 73.

Table 73 — tagStructType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'tstr' (74737472h) tagStructType signature	
4..7	4	Reserved, shall be 0	
8..11	4	Struct type Identifier	4-byte signature
12..15	4	Number of tag elements N in structure	uInt32Number
16..19	4	Tag element 1 signature	4-byte signature
20..27	8	Tag element 1 position	positionNumber
…	…	…	…
N*12+4..N*12+7	4	Tag element N signature	4-byte signature
N*12+8..N*12+15	8	Tag element N position	positionNumber
N*12+16..end		Tag element data	

Each tag element (or sub-tag) of a tagStructType has a tag signature, offset and size. Each offset is relative to the beginning of the associated tagStructType structure. All elements should begin at an offset divisible by 4 with padding between tag elements as needed. The struct type identifier shall be used to identify the required and optional sub-tag elements in the tag structure. (See Clause 12 for publicly defined tagStructType structure definitions.)

Tag elements can be any valid profile tag type.

Tag element signatures shall be unique within a tagStructType structure.

The Offset of multiple elements can be the same (i.e. elements can share tag data).

10.2.24 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.

When used, the byte assignment and encoding shall be as given in Table 74.

Table 74 — u16Fixed16ArrayType encoding

Byte position	Field length (bytes)	Content
0 to 3	4	‘uf32’ (75663332h) type signature
4 to 7	4	Reserved, shall be 0
8 to end	Variable	An array of u16Fixed16Number values

10.2.25 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.

When used, the byte assignment and encoding shall be as given in Table 75.

Table 75 — uInt16ArrayType encoding

Byte position	Field length (bytes)	Content
0 to 3	4	‘ui16’ (75693136h) type signature
4 to 7	4	Reserved, shall be 0
8 to end	Variable	An array of unsigned 16bit integers

10.2.26 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.

When used, the byte assignment and encoding shall be as given in Table 76.

Table 76 — uInt32ArrayType encoding

Byte position	Field length (bytes)	Content
0 to 3	4	‘ui32’ (75693332h) type signature
4 to 7	4	Reserved, shall be 0
8 to end	Variable	An array of unsigned 32-bit integers

10.2.27 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.

When used, the byte assignment and encoding shall be as given in Table 77.

Table 77 — uInt64ArrayType encoding

Byte position	Field length (bytes)	Content
0 to 3	4	‘ui64’ (75693634h) type signature
4 to 7	4	Reserved, shall be 0
8 to end	Variable	An array of unsigned 64-bit integers

10.2.28 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.

When used, the byte assignment and encoding shall be as given in Table 78.

Table 78 — uInt8ArrayType encoding

Byte position	Field length (bytes)	Content
0 to 3	4	'ui08' (75693038h) type signature
4 to 7	4	Reserved, shall be 0
8 to end	Variable	An array of unsigned 8-bit integers

10.2.29 utf16Type

This tag structure contains a text structure that contains a 16-bit UTF-16 string. The length of the string is obtained by subtracting 8 from the element size portion of the tag itself. For the specification of Unicode, see The Unicode Standard published by The Unicode Consortium or visit http://www.unicode.org.

The format of the utf16Type structure can be found in Table 79.

Table 79 — utf16Type encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'ut16' (75743136h) type signature	
4..7	4	Reserved, shall be 0	
8..end		UTF-16 data	uInt16Number[...]

10.2.30 utf8Type

This tag structure contains a text structure that contains an 8-bit UTF-8 string. The length of the string is obtained by subtracting 8 from the element size portion of the tag itself. For the specification of Unicode, see The Unicode Standard published by The Unicode Consortium or visit http://www.unicode.org.

The format of the utf8Type structure can be found in Table 80.

Table 80 — utf8Type encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'utf8' (75746638h) type signature	
4..7	4	Reserved, shall be 0	
8..end		UTF-8 data	

10.2.31 utf8ZipType

This tag structure is a container for a UTF-8 string that has been compressed using the DEFLATE compression method specified by RFC 1951 (http://tools.ietf.org/html/rfc1951) into the compressed data format specified by RFC 1950 (http://tools.ietf.org/html/rfc1950).

NOTE This is equivalent to the Zip data format produced by the ZLIB data compression library.

The length of the compressed data stream can be determined by subtracting 8 from the element size portion of the tag itself.

The data that is compressed is a UTF-8 string. For the specification of Unicode, see The Unicode Standard published by The Unicode Consortium or visit http://www.unicode.org.

The format of the utf8ZipType structure can be found in Table 81.

Table 81 — utf8ZipType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'zut8' (7a757438h) type signature	
4..7	4	Reserved, shall be 0	
8..end		Compressed data stream	

10.2.32 XYZType

The XYZType contains an array of three encoded values for PCSXYZ, CIEXYZ, or nCIEXYZ values. The number of sets of values is determined from the size of the tag. When used, the byte assignment and encoding shall be as given in Table 82. Tristimulus values shall be non-negative. The signed encoding allows for implementation optimizations by minimizing the number of fixed formats.

Table 82 — XYZType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0 to 3	4	'XYZ ' (58595A20h) type signature	
4 to 7	4	Reserved, shall be 0	
8 to end	Variable	An array of PCSXYZ, CIEXYZ, or nCIEXYZ values	XYZNumber

10.2.33 zipXmlType

This tag structure is a container for XML formatted data that has been compressed using the DEFLATE compression method specified by RFC 1951 (http://tools.ietf.org/html/rfc1951) into the compressed data format specified by RFC 1950 (http://tools.ietf.org/html/rfc1950).

NOTE This is equivalent to the Zip data format produced by the ZLIB data compression library.

The length of the compressed data stream can be determined by subtracting 8 from the element size portion of the tag itself.

The data that is compressed shall be encoded using XML. For the specification of XML, see the *XML 1.0 specification*[14] published by the World Wide Web Consortium or visit http://www.w3.org/TR/REC-xml.

The format of the zipXmlType structure can be found in Table 83.

Table 83 — zipXmlType encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'zxml' (7a786d6ch) type signature	
4..7	4	Reserved, shall be 0	
8..end		Compressed data stream	

11 multiProcessingElementType definitions

11.1 General

The multiProcessElementsType and several of the processing elements presented in this subclause are defined by ISO 15076-1. The use of multiProcessElementsType-based tags is more extensively utilized by this extended version of ICC colour management with both modifications/extensions to the processing elements defined by ISO 15076-1 as well as the inclusion of additional processing elements.

Processing elements in the multiProcessElementsType are processed in the order that they are defined in the processing elements position table. The results of a processing element are passed on to the next processing element. The last processing element provides the final result for the containing multiProcessElementsType tag. Therefore, the input/output channels specified by the processing elements and the containing multiProcessElementsType tag need to be in agreement.

The first processing element's input channels shall be the same as the input channels of the containing multiProcessElementsType tag. The input channels of a processing element shall be the same as the previous processing element's output channels. The last processing element's output channels shall be the same as the output channels of the containing multiProcessElementsType tag.

Clipping of the results of a processing element shall not be performed. Some processing elements may perform clipping as needed on input.

The specification for each processing element shall indicate whether that element performs clipping on input.

The general element encoding for multiProcessElementsType tag elements is shown in Table 84.

Table 84 — generalElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	Element type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12..end	4	Element Data	

11.2 Specific processing element listing

11.2.1 calculatorElement

11.2.1.1 General

A calculatorElement allows for the encoding of arbitrary functions of multiple data inputs.

A calculatorElement can be used to augment the other multi-processing elements. A calculatorElement can also contain sub-elements that can be conditionally evaluated within the context of the calculatorElement's main function.

In addition to defining input and output channels, a calculatorElement can also define and use temporary channel storage, which provides additional channels of data outside the channel data persisted between processing elements within a single Multi Processing Element Tag. Input and output channel data are maintained separately.

Temporary channel data are maintained and stored only within the context of a single Multi Processing Element Tag. Temporary channel data shall not be persisted from one Multi Processing Element tag to

another when Multi Processing Element tags are connected or referenced as sub-calculator elements. At each invocation of a calculator (or sub-calculator) element it shall be assumed that all Temporary channel data are initialized to zero.

For performance purposes it is recommended that temporary channel data be initialized before being referenced.

The maximum number of input, output and temporary channels shall be 65 535.

The encoding of a calculatorElement is shown in Table 85.

Table 85 — calculatorElement encoding

Byte position	Field length (bytes)	Content	Encoded as…
0…3	4	'calc' (63616C63h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12…15	4	Number of sub-elements (E)	uInt32Number
16…23	8	Main function position	positionNumber
24…24+8*E-1	8*E	Sub-element positions	Array of positionNumber
24+8*E … end		Data for calculator element	

The main function defines a sequence of operations using RPN (reverse polish notation). A data stack of numeric results is kept as operations are evaluated. The data stack is assumed to be empty at the start of each main function evaluation. Each operation in a function can use a constant parameter, input channel data, or temporary channel data to place results onto the data stack or otherwise manipulate the data stack.

During the course of interpretation, the main calculator function can place data results into the output channels or into temporary channel storage. Output channels shall be assumed to be zero until set by the main calculator function.

Main function validity checking shall be performed by checking for valid operations, valid channel index addressing, and valid stack access (with no stack underflow or overflow) before main function evaluation is performed to ensure system data integrity. The reserved storage for the data stack shall be for at least 65 535 values.

Mathematical error handling is the responsibility of the calculatorElement script implementer. calculatorElement operations generally take data from the stack and place results on the stack. In some cases the operation has no defined result (like dividing by zero) and non-real numbers (+INF, -INF, or NaN) may be placed on the stack as the result. Operations that use such non-real values as input may also result in non-real values as output. Regardless, the number of values consumed and produced by each operation shall be as specified for the operation. The 'rnum' (726e756dh) operator can be used to determine if values on the stack are real numbers, or stack values can be compared to the results of the '+INF', '-INF', and 'NaN ' operators. The behaviour of a CMM for non-real values placed in the output channels or passed to calculatorElement sub-elements is implementation dependent.

The encoding of a calculatorElement function is shown in Table 86.

Table 86 — calculatorElement Function encoding

Byte position	Field length (bytes)	Content	Encoded as…
0…3	4	'func' (66756e63h) type signature	
4…7	4	Reserved, shall be 0	
8…11	4	Number of operations (N)	uInt32Number
12…12 + N*8 - 1	8	Function operations	

Individual operations shall be encoded as a signature followed by four data bytes. There are eight types of operation encodings (Push floating point constant, channel vector, sub-element invocation, stack operation, matrix, sequence functional, function vector, and conditional).

NOTE Information about a textual representation of calculator elements with examples can be found in Annex F.

11.2.1.2 Floating point constant operations

The floating point constant operation is used to push a single float32Number onto the evaluation stack. The floating point constant operation encoding is shown in Table 87.

Table 87 — Floating point constant operation encoding

Byte position	Field length (bytes)	Content	Encoded as…
0…3	4	'data' (64617461h)	
4…7	4	float32Number to put on stack	float32Number

11.2.1.3 Channel vector operations

A channel vector operation is used to operate on input, output or temporary channel data either by pushing it onto the evaluation stack or storing evaluation stack data into channel storage. The encoding of a channel vector operation is shown in Table 88 with descriptions of the channel vector operation signatures shown in Table 89.

The **in** channel operation is limited to retrieving pixel data from input channels defined in the calculator element header. The **out** channel operator is limited to storing pixel data to output channels defined in the calculator element. Input channels are read only and therefore the use of the **out** channel operator shall not affect input channel values. Temporary channels are assumed to be zero at the start of each calculator element's main function evaluation.

Table 88 — Channel vector operation encoding

Byte position	Field length (bytes)	Content	Encoded as…
0…3	4	Operation signature	
4…5	2	Starting index (S)	uInt16Number
6…7	2	Additional count from start (T)	uInt16Number

The index is the starting index of the input, output, or temporary data channel to use

Table 89 — Channel vector operations by signature

Operation signature	Stack arguments	Operator definition	Stack results
'in ' (696e2020h)	None	Load from input pixel channel number S through S+T	in[S] … in[S+T]
'out ' (6f757420h)	A_0 … A_T	Store to output pixel channel number S through S+T. Thus: out[S]=A_0, …., out[S+T]=A_T	None
'tget' (74676574h)	None	Get temporary channels S through S+T	temp[S]… temp[S+T]
'tput' (74707574h)	A_0 … A_T	Put temporary channels S+T through S. Thus: temp[S]=A_0, …., temp[S+T]=A_T	None
'tsav' (74736176h)	A_0 … A_T	Saves arguments on stack as temporary channels S+T through S without affecting arguments on the stack. Thus: temp[S]=A_0, …., temp[S+T]=A_T	A_0 … A_T

NOTE For the **out**, **tput**, and **tsav** operators the topmost element on the stack is stored at the S+T channel position.

11.2.1.4 CMM environment variable operation

The CMM environment variable operation is used to provide environmental data information that can optionally be provided to the CMM as input onto the evaluation stack, thus allowing control or operations within the calculator element to be guided by external configuration. The encoding of the CMM environment variable operation is shown in Table 90 with the description of the CMM environment variable operation signature shown in Table 91.

Table 90 — Environment variable operation encoding

Byte position	Field length (bytes)	Content	Encoded as…
0…3	4	Operation signature	
4…7	4	Environment variable signature (X)	uInt32Number

Signature values for the environment variable signature (X) are open ended and workflow dependent. Interoperable usage of CMM environment variables shall use environment variable signatures that are specified in separate ICSs separate from this document's specifications or registered separately with the ICC.

Table 91 — Environment variable operation by signature

Operation signature	Stack arguments	Operator definition	Stack results
'env ' (656e7620h)	None	Places 32-bit floating point CMM environment variable value on stack (denoted by env(X)) if variable with signature X is available and supported. Additional value placed to indicate whether variable is supported	env(X) 1,0 if X is available and supported 0,0 0,0 otherwise

The **env** operation shall consume no values from the evaluation stack and shall always place two values onto the evaluation stack. The first value placed on the stack shall be the 32-bit floating point CMM environment variable value associated with the 32-bit environment variable signature (represented by the function env(X) in Table 91) if the CMM environment variable (X) is available to and supported by the CMM, or 0,0 otherwise. The second value placed on the stack shall be 1,0 if the CMM environment variable X is available to and supported by the CMM, or 0,0 otherwise.

It is the responsibility of the calculatorElement script implementer to provide appropriate handling of operations when a desired CMM environment variable is not available or supported.

Two CMM environment variables shall always be handled with the resulting stack values as shown in Table 92.

Table 92 — Required CMM environment variable support by signature

Environment variable signature	Stack results
'true' (74727565h)	1,0 1,0
‘ndef’ (6e646566h)	0,0 0,0

NOTE Support for and methods of supplying additional CMM environment values to the CMM are implementation dependent.

11.2.1.5 Sub-element invocation operations

A sub-element invocation operation allows for processing elements associated with the calculator element to be selectively applied. When a sub-processing element is invoked the input channels associated with the processing element are first taken from the evaluation stack. These values are then used by the sub-element to perform its processing to get output channel values which are then placed onto the evaluation stack.

Sub-elements are separate processing elements and therefore use their own temporary variables and evaluation stacks. In other words, if a sub-element is a calculator element it shall have independent scope from the calling calculator element.

The curv, mtx, and clut operators require that the indexed sub-element has the appropriate type. The elem operator performs no type check on the type of the element.

The Index parameter specifies the index of the element type to use. The encoding of a sub-element operation encoding is shown in Table 93 with descriptions of the sub-element operation signatures shown in Table 94.

Table 93 — Sub-element operation encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	Operation signature	
4…7	4	Element Index (S)	uInt32Number

Table 94 — Sub-element operations by signature

Operation signature	Stack arguments	Operator definition	Stack results
‘curv' (63757276h)	$X_1 \ldots X_{Input}$	Applies sub-element (S) as a curve set	$Y_1 \ldots Y_{0Output}$
‘mtx ' (6d747820h)	$X_1 \ldots X_{Input}$	Applies sub-element (S) as a matrix	$Y_1 \ldots Y_{0Output}$
'clut' (636c7574h)	$X_1 \ldots X_{Input}$	Applies sub-element (S) as a CLUT	$Y_1 \ldots Y_{0Output}$
'calc' (63616c63h)	$X_1 \ldots X_{Input}$	Applies sub-element (S) as a calculator	$Y_1 \ldots Y_{0Output}$
'tint' (74696e74h)	$X_1 \ldots X_{Input}$	Applies sub-element (S) as a tint	$Y_1 \ldots Y_{0Output}$
'elem' (656c656dh)	$X_1 \ldots X_{Input}$	Applies sub-element (S)	$Y_1 \ldots Y_{0Output}$

11.2.1.6 Stack operations

A stack operation is used to manipulate multiple elements of the evaluation stack directly. The encoding of a stack operation is shown in Table 95 with descriptions of the stack operation signatures shown in Table 96.

Table 95 — Stack operation encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	Operation signature	
4…5	2	Number of extra elements selector S	uInt16Number
6…7	2	Number of extra times selector T	uInt16Number

Table 96 — Stack operations by signature

Operation signature	Stack arguments	Operator definition	Stack results
'copy' (636f7079h)	A_0 … A_S	Duplicate top S+1 elements T+1 times (stack results shown for T=0)	A_0 … A_S A_0 … A_S
'rotl' (726f746ch)	A_0 … A_S	Rotate left top S+1 elements T+1 positions on stack (stack results shown for T=0)	A_1 … A_S A_0
'rotr' (726f7472h)	A_0 … A_S	Rotate right top S+1 elements T+1 positions on stack (stack results shown for T=0)	A_S A_0 … A_{S-1}
'posd' (706f7364h)	A_S … A_0	Duplicate the element at the Sth position from top of stack T+1 times (stack results shown for T=0)	A_S … A_0 A_S
‘flip’ (666c6970h)	A_0 … A_{S+1}	Reverse the top S+1 elements on the stack (T shall be zero)	A_{s+1} … A_0
‘pop ‘ (706f7020h)	A_0 … A_S	Remove top S+1 elements on the stack (T shall be zero)	

NOTE In the above table the last element listed is the first item in the evaluation stack.

11.2.1.7 Matrix operations

A matrix operation performs operations to matrix data or matrix data plus column vector data placed on the evaluation stack directly. The encoding of a matrix operation is shown in Table 97 with descriptions of the matrix operation signatures shown in Table 98.

Table 97 — Stack operation encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	Operation signature	
4…5	2	Index of last matrix row S	uInt16Number
6…7	2	Index of last matrix column T	uInt16Number

Table 98 — Stack operations by signature

Operation signature	Stack arguments	Operator definition	Stack results
'solv' (736f6c76h)	$A_{0,0}$ … $A_{0,T}$ … $A_{S,0}$ … $A_{S,T}$ Y_0 … Y_S	Solve for x in matrix vector equation **y**=**Ax** where **x** and **y** are column vectors and **A** is a matrix containing S+1 rows and T+1 columns. Z=1 indicates operation was successful and supported by implementation. Z=0 results in contents of **x** set to zero and indicates failure to invert **A** or lack of support by implementation.	X_0 … X_T Z
'tran' (7472616eh)	$A_{0,0}$ … $A_{0,T}$ … $A_{S,0}$ … $A_{S,T}$	Transpose matrix elements on stack with S+1 rows and T+1 columns	$A_{0,0}$ … $A_{S,0}$ … $A_{0,T}$… $A_{S,T}$

11.2.1.8 Sequence functional operations

The sequence functional operations take two or more arguments off the evaluation stack and place onto the evaluation stack a single value that represents the result. The Size parameter specifies how many more than two arguments to use. The encoding of a sequence function vector operation encoding is shown in Table 99 with descriptions of the function operation signatures shown in Table 100.

Table 99 — Sequence functional operation encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	Operation signature	
4…5	2	Additional Size (S)	uInt16Number
6…7	2	Reserved, shall be 0	

Table 100 — Variable length functional operations by signature

Operation signature	Stack arguments	Operator definition	Stack results
'sum ' (73756d20h)	X_0 …X_{S+1}	$Z = X_0 + … + X_{S+1}$	Z
'prod' (70726f64h)	X_0 … X_{S+1}	$Z = X_0 * … * X_{S+1}$	Z
'min ' (6d696e20h)	X_0 …X_{S+1}	Z is minimum of X_0 through X_{S+1}	Z
'max ' (6d617820h)	X_0 …X_{S+1}	Z is maximum of X_0 through X_{S+1}	Z
'and ' (616e6420h)	X_0 …X_{S+1}	Z=1 if ALL X_0 through XS+1 are greater than or equal to 0,5. Else Z=0	Z
'or ' (6f722020h)	X_0 …X_{S+1}	Z=1 if ANY X_0 through X_{S+1} is greater than or equal to 0,5. Else Z=0	Z

11.2.1.9 Functional vector operations

A functional vector operation (optionally) takes one or two vector arguments off the evaluation stack and places onto the evaluation stack a single vector result. The Size parameter specifies the last index of a vector to use with zero-based indexing. The encoding of a function vector operation encoding is shown in Table 101 with descriptions of the function operation signatures shown in Table 102.

Table 101 — Functional vector operation encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	Operation signature	
4…5	2	Vector index selector (S)	uInt16Number
6…7	2	Reserved, shall be 0	

Table 102 — Functional vector operations by signature

Operation signature	Stack arguments	Operator definition	Stack results
'pi ' (70692020h)	None	Mathematical value Π (S shall be zero)	Π
'+INF' (2b494e46h)	None	Floating point value for positive infinity (S shall be zero)	+INF
'-INF' (2d494e46h)	None	Floating point value for negative infinity (S shall be zero)	-INF
'NaN '(4e614e20h)	None	Floating point value for "Not a Number" (S shall be zero)	NaN
'add' (61646420h)	X_0 … X_S Y_0…Y_S	$Z_i = X_i + Y_i$ (for i=0…S)	Z_0 … Z_S
'sub' (73756220h)	X_0… X_S Y_0…Y_S	$Z_i = X_i - Y_i$ (for i=0…S)	Z_0… Z_S
'mul' (6d756c20h)	X_0… X_S Y_0…Y_S	$Z_i = X_i * Y_i$ (for i=0…S)	Z_0… Z_S
'div' (64697620h)	X_0… X_S Y_0…Y_S	$Z_i = X_i / Y_i$ (for i=0….S)	Z_0… Z_S
'mod' (6d6f6420h)	X_0… X_S Y_0…Y_S	$Z_i = X_i - \text{trunc}(X_i / Y_i) * Y_i$ (for i=0….S)	Z_0… Z_S
'pow ' (706f7720h)	X_0… X_S Y_0…Y_S	$Z_i = X_i^{Y_i}$ (for i=0…S)	Z_0… Z_S
'gama' (67616d61h)	X_0… X_S Y	$Z_i = X_i^{Y}$ (for i=0…S)	Z_0… Z_S
'sadd' (73616464h)	X_0… X_S Y	$Z_i = X_i + Y$ (for i=0…S)	Z_0… Z_S
'ssub' (73737562h)	X_0… X_S Y	$Z_i = X_i - Y$ (for i=0…S)	Z_0… Z_S
'smul' (736d756ch)	X_0… X_S Y	$Z_i = X_i * Y$ (for i=0…S)	Z_0… Z_S
'sdiv' (73646976h)	X_0… X_S Y	$Z_i = X_i / Y$ (for i=0….S)	Z_0… Z_S
'sq ' (73712020h)	X_0… X_S	$Z_i = X_i * X_i$ (for i=0…S)	Z_0… Z_S
'sqrt' (73717274h)	X_0… X_S	$Z_i = \sqrt{X_i}$ (for i=0…S)	Z_0… Z_S
'cb ' (63622020h)	X_0… X_S	$Z_i = X_i * X_i * X_i$ (for i=0…S)	Z_0… Z_S
'cbrt' (63627274h)	X_0… X_S	$Z_i = \sqrt[3]{X_i}$ (for i=0…S)	Z_0… Z_S

'abs ' (61627320h)	X_0… X_S	If (X_i < 0,0)
		$Z_i = - X_i$
		Else
		$Z_i = X_i$
		(for i=0…S)
			Z_0… Z_S

'neg ' (6e656720h)	X_0… X_S	$Z_i = - X_i$ (for i=0…S)	Z_0… Z_S

Operation signature	Stack arguments	Operator definition	Stack results

‘rond‘ (726f6e64h)	X_0... X_S	If (X_i < 0,0)
		Z_i = trunc(X_i – 0,5)
		Else
		Z_i = trunc(X_i + 0,5)
		(for i=0...S)
			Z_0... Z_S

‘flor‘ (666c6f72h)	X_0... X_S	Z_i = floor(X_i) (for i=0...S)	Z_0... Z_S
‘ceil‘ (6365696ch)	X_0... X_S	Z_i = ceil(X_i) (for i=0...S)	Z_0... Z_S
‘trnc‘ (74726e63h)	X_0... X_S	Z_i = trunc(X_i) (for i=0...S)	Z_0... Z_S

‘sign‘ (7369676eh)	X_0... X_S	If (X_i < 0,0)
		Z_i = - 1
		Else If (X_i > 0,0)
		Z_i = -1
		Else
		Z_i = 0
		(for i=0...S)
			Z_0... Z_S

'exp ' (65787020h)	X_0... X_S	$Z_i = e^{X_i}$ (for i=0...S)	Z_0... Z_S
'log ' (6c6f6720h)	X_0... X_S	Z_i = log(X_i) (for i=0...S)	Z_0... Z_S
'ln ' (6c6e2020h)	X_0... X_S	Z_i = ln(X_i) (for i=0...S)	Z_0... Z_S
'sin ' (73696e20h)	X_0... X_S	Z_i = sin(X_i) (for i=0...S) NOTE X is in radians	Z_0... Z_S
'cos ' (636f7320h)	X_0... X_S	Z_i = cos(X_i) (for i=0...S) NOTE X is in radians	Z_0... Z_S
'tan ' (74616e20h)	X_0... X_S	Z_i = tan(X_i) (for i=0...S) NOTE X is in radians	Z_0... Z_S
'asin' (6173696eh)	X_0... X_S	$Z_i = \sin^{-1}(X_i)$ (for i=0...S) NOTE result is in radians	Z_0... Z_S
'acos' (61636f73h)	X_0... X_S	$Z_i = \cos^{-1}(X_i)$ (for i=0...S) NOTE result is in radians	Z_0... Z_S
'atan' (6174616eh)	X_0... X_S	$Z_i = \tan^{-1}(X_i)$ (for i=0...S) NOTE result is in radians	Z_0... Z_S
'atn2' (61746e32h)	X_0... X_S Y_0...Y_S	$Z_i = \tan^{-1}\left(\frac{Y_i}{X_i}\right)$ (for i=0...S) NOTE result is in radians	Z_0... Z_S

Operation signature	Stack arguments	Operator definition	Stack results
'ctop' (63746f70h)	X_0... X_S Y_0...Y_S	$R_i = \sqrt{X_i^2 + Y_i^2}$ $A_i = \tan^{-1}\left(\frac{Y_i}{X_i}\right)\frac{180}{\pi}$ (for i=0...S) NOTE resulting Ai in degrees ranging from 0 to 360	R_0... R_S A_0...A_S
'ptoc' (70746f63h)	R_0... R_S A_0...A_S	$X_i = R_i \cos\left(A_i \frac{\pi}{180}\right)$ $Y_s = R_i \sin\left(A_i \frac{\pi}{180}\right)$ (for i=0...S) NOTE Ai in degrees ranging from 0 to 360	X_0... X_S Y_0...Y_S

'rnum' (726e756dh)	X_0 ... X_s	Checks for real numbers
		If (X_i=+INF or
		X_i=-INF or
		X_i=NaN)
		Zi = 0,0
		Else
		Zi = 1,0
			Z_0... Z_S

'lt ' (6c742020h)	X_0... X_S Y_0...Y_S	If (X_i < Y_i)
		Z_i = 1,0
		Else
		Z_i = 0,0
		(for i=0...S)
			Z_0... Z_S

'le ' (6c652020h)	X_0... X_S Y_0...Y_S	If (X_i ≤ Y_i)
		Z_i = 1,0
		Else
		Z_i = 0,0
		(for i=0...S)
			Z_0... Z_S

'eq ' (65712020h)	X_0... X_S Y_0...Y_S	If (X_i = Y_i)
		Z_i = 1,0
		Else
		Z_i = 0,0
		(for i=0...S)
		NOTE Differences in encoding might result in Z=0,0. Use 'near' to account for such differences
			Z_0... Z_S

Operation signature	Stack arguments	Operator definition	Stack results

'near' (6e656172h)	X_0... X_S Y_0...Y_S	If (Y_i -ξ ≤ X_i and X_i ≤ Y_i + ξ)
		Z_i = 1,0
		Else
		Z_i = 0,0
		(for i=0...S)
		NOTE ξ = $1x10^{-8}$ to allow for small differences in floating point encoding.
			Z_0... Z_S

'ge ' (67652020h)	X_0... X_S Y_0...Y_S	If (X_i ≥ Y_i)
		Z_i = 1,0
		Else
		Z_i = 0,0
		(for i=0...S)
			Z_0... Z_S

'gt ' (67742020h)	X_0... X_S Y_0...Y_S	If (X_i > Y_i)
		Z_i = 1,0
		Else
		Z_i = 0,0
		(for i=0...S)
			Z_0... Z_S

'vmin' (766d696eh)	X_0 ... X_S Y_0...Y_S	Z_i = min(X_i , Y_i) (for i=0...S)	Z_0 ... Z_S
'vmax' (766d6178h)	X_0 ... X_S Y_0...Y_S	Z_i = max(X_i , Y_i) (for i=0...S)	Z_0 ... Z_S

'vand' (76616e64h)	X_0 ... X_S Y_0...Y_S	if (X_i ≥ 0,5 and Y_i ≥ 0,5)
		Z_i = 1,0
		Else
		Z_i = 0
		(for i=0...S)
			Z_0 ... Z_S

Operation signature	Stack arguments	Operator definition	Stack results

'vor' (766f7220h)	X_0 ... X_S Y_0...Y_S	if ($X_i \geq 0,5$ or $Y_i \geq 0,5$)
		$Z_i = 1,0$
		Else
		$Z_i = 0$
		(for i=0...S)
			Z_0 ... Z_S

'tLab' (744c6162h)	X_0...X_S Y_0 ... Y_S Z_0 ... Z_S	$L_i = f(Y_i) - 16$ $a_i = 500\left[f(X_i) - f(Y_i)\right]$ $b_i = 200\left[f(Y_i) - f(Z_i)\right]$ (for i=0...S) Where: $f(t) = \begin{cases} t^{1/3} & \text{when } t > \left(\frac{6}{29}\right)^3 \\ \left(\frac{841}{108}\right) t + \frac{4}{29} & \text{when } t \leq \left(\frac{6}{29}\right)^3 \end{cases}$ NOTE X_i, Y_i, Z_i represent normalized values.	L_0...L_S a_0 ... a_S b_0...b_S
'tXYZ' (7458595ah)	L_0...L_S a_0 ... a_S b_0...b_S	$Y_i = f^{-1}\left(\frac{L_i + 16}{116}\right)$ $X_i = f^{-1}\left(\frac{L_i + 16}{116} + \frac{a_i}{500}\right)$ $Z_i = f^{-1}\left(\frac{L_i + 16}{116} - \frac{b_i}{200}\right)$ (for i=0...S) where: $f^{-1}(t) = \begin{cases} t^3 & \text{when } t > \frac{6}{29} \\ \frac{108}{841}\left(t - \frac{4}{29}\right) & \text{when } t \leq \frac{6}{29} \end{cases}$ NOTE X_i, Y_i, Z_i represent normalized values.	X_0...X_S Y_0 ... Y_S Z_0 ... Z_S
NOTE The last element listed in the argument stack is the first item in the evaluation stack.			

11.2.1.10 Conditional operations

The conditional operations allow the encoding and conditional evaluation of operation streams based upon comparing the topmost evaluation stack entry to 0,5. An 'if ' (69662020h) operation with its associated stream of operations can optionally be immediately followed by an ‘else’ operation with its stream of associated operations.

Only one associated stream of operations shall be evaluated depending upon the value on top of the evaluation stack.

If the topmost value is greater than or equal to 0,5 or ‘NaN ’ then the stream associated with the 'if ' operation shall be evaluated.

If the topmost value is less than 0,5 and an ‘else’ operation immediately succeeds an 'if ' operation, then the stream of operations associated with the succeeding ‘else’ operation shall be evaluated.

If the topmost value is less than 0,5 and an ‘else’ operation does not immediately succeed an 'if ' operation, the stream of operations associated with the 'if ' operation shall be skipped.

An ‘else’ operation shall always be preceded by an 'if ' operation.

Before evaluating either the associated 'if ' or ‘else’ operation streams (or skipping the associated 'if ' stream of operations) the topmost value shall be removed from the stack and no further arguments are placed on the stack before evaluating the selected operation stream.

The number of operations in an 'if ' or ‘else’ associated operation stream shall be zero or more.

The encoding of the 'if ' conditional operation is shown in Table 103, and the encoding of an 'if ' with accompanying ‘else’ conditional operation is shown in Table 104.

Table 103 — Conditional if operation encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'if ' (69662020h)	
4…7	4	Number of operations (T) to evaluate if stack argument is greater than or equal to 0,5 or NaN	uInt32Number
8…7 + 8T	8T	Operations to evaluate if stack argument was greater than or equal to 0,5	

Table 104 — Conditional if with else operation encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'if ' (69662020h)	
4…7	4	Number of operations (T) to evaluate if stack argument is greater than or equal to 0,5 or NaN	uInt32Number
8…11	4	'else' (656c7365h)	
12…15	4	Number of operations (U) to evaluate if either the previous 'if ' conditional operation found a stack argument less than 0,5	uInt32Number
8…7 + 8T	8T	Operations to evaluate if stack argument was greater than or equal to 0,5	
8+8T…7 + 8T+8U	8U	Operations to evaluate if either the previous 'if ' conditional operation found a stack argument less than 0,5	

11.2.1.11 Selection operations

The selection operations allow the encoding and conditional evaluation of operation streams based upon the rounded integer value of the topmost evaluation stack entry to select a single stream of associated operations to be evaluated. The selection 'sel ' (73656c20h) operation shall be immediately followed by one or more ‘case’ (63617365h) operations with associated ‘case’ streams of operations. Additionally, a ‘dflt’ (64666c74h) operation with its stream of associated operations can follow immediately after the last ‘case’ operation which has an associated stream of operations.

At most only one associated stream of operations shall be evaluated depending upon the rounded integer value of the topmost evaluation stack entry.

The list of case following a 'sel ' operation can be considered as a zero-based array of case streams of length N+1.

Evaluation of a 'sel ' operation shall be performed by removing the top value from the evaluation stack and rounding to its nearest integer value (in identical manner as the ‘rond’ operation) to define the selection index S.

Then, if S is in the range between and including zero and N then S shall be used to index the subsequent zero-based array of ‘case’ operations to select which associated stream of operations to evaluate. If S is less than zero or greater than or equal to N and a ‘dflt’ operation with its associated stream of operations follows the list of case streams then the stream of operations associated with the ‘dflt’ operation shall be evaluated. Otherwise if S is less than zero or greater than or equal to N and no ‘dflt’ operation follows the last ‘case’ stream then no stream of operations shall be evaluated and the next operation to be evaluated shall be the operation immediately after last ‘case’ stream.

A 'sel ' operation shall always be followed by one or more ‘case’ operations.

A ‘case’ operation shall always be preceded by a 'sel ' or ‘case’ operation.

A ‘dflt’ operation shall always be preceded by a ‘case’ operation.

The topmost value shall be removed from the stack before evaluating either the associated ‘case’ or ‘dflt’ operation streams (if one of these streams is selected), and no further arguments shall be placed on the stack before evaluating the selected operation stream.

The number of associated operations for either a ‘case‘ or ‘dflt’ operation stream shall be zero or more.

The encoding of the 'sel ' conditional operation with ‘case’ operations and ‘dflt’ operation is shown in Table 105; the encoding of the 'sel ' conditional operation with ‘case operations and no ‘dflt’ operation is shown in Table 106.

Table 105 — Selection 'sel ' operation with ‘dflt’ encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'sel ' (73656c20h)	
4…7	4	Reserved: shall be zero	
8..11	4	'case' (63617365h)	
12…15	4	Number of operations (U_0) to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case 0 to be evaluated.	uInt32Number
16..19	4	'case' (63617365h)	
20…23	4	Number of operations (U_1) to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case 1 to be evaluated.	uInt32Number
…			
12+8N... 15+8N	4	'case' (63617365h)	
16+8N… 19+8N	4	Number of operations (U_N) to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case N to be evaluated.	uInt32Number
20+8N.. 23+8N	4	'dflt' (64666c74h)	
24+8N… 27+8N	4	Number of operations (T) to evaluate if the rounded stack argument is less than zero or greater than or equal to N.	uInt32Number

28+8N… 27+8N + $8U_0$	$8U_0$	Operations to evaluate if the rounded stack argument evaluated by a 'sel ' operation selected case 0 to be evaluated	
28+8N+$8U_0$… 27+8N+$8U_0$+$8U_1$	$8U_1$	Operations to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case 1 to be evaluated	
…			
28+8N+$\sum_{i=0}^{N} 8U_i$ … 27+8N+$\sum_{i=0}^{N} 8U_i$	$8U_N$	Operations to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case N to be evaluated	
32+8N+$\sum_{i=0}^{N} 8U_i$ … 31+8N+$\sum_{i=0}^{N} 8U_i$ +8T	8T	Operations to evaluate if the rounded argument is less than zero or greater than N	

Table 106 — Selection 'sel ' operation without ‘dflt’ encoding

Byte position	Field length (bytes)	Content	Encoded as…
0…3	4	'sel ' (73656c20h)	
4…7	4	Reserved: shall be zero	
8..11	4	'case' (63617365h)	
12…15	4	Number of operations (U_0) to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case 0 to be evaluated.	uInt32Number
16..19	4	'case' (63617365h)	
20…23	4	Number of operations (U_1) to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case 1 to be evaluated.	uInt32Number
…			
12+8N.. 15+8N	4	'case' (63617365h)	
16+8N… 19+8N	4	Number of operations (U_N) to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case N to be evaluated.	uInt32Number
16+8N… 19+8N + $8U_0$	$8U_0$	Operations to evaluate if the rounded stack argument evaluated by a 'sel ' operation selected case 0 to be evaluated	

Table 106 *(continued)*

16+8N+8U$_0$... 19+8N+8U$_0$+8U$_1$	8U$_1$	Operations to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case 1 to be evaluated	
...			
16+8N+$\sum_{i=0}^{N} 8U_i$ ... 19+8N+$\sum_{i=0}^{N} 8U_i$	8U$_N$	Operations to evaluate if the rounded stack argument evaluated by the 'sel ' operation selected case N to be evaluated	

11.2.2 curveSetElement

11.2.2.1 General

The Curve Set element encodes multiple one-dimensional curves. The encoding is shown in Table 107.

Table 107 — curveSetElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0...3	4	'cvst' (63767374h) type signature	
4...7	4	Reserved, shall be 0	
8...9	2	Number of input channels (P)	uInt16Number
10...11	2	Number of output channels (Q)	uInt16Number
12..11+8P	8P	Curve positions (offset and size)	positionNumber[...]
12+8P to end		Data	

Encoding values for both input and output channels is for consistency with other processing elements. Since each one-dimensional curve maps a single input to a single output, the number of outputs shall be the same as the number of inputs. Thus, the number of output channels (Q) shall be the same value as the number of input channels (P).

The output value for an input shall be specified by the first segment in the segment list that contains that input. Successive break-points shall not be decreasing.

Each channel shall have a curve position element. Offset locations are relative to the start of the containing curveSetElement. Thus the offset of first stored curve in the curve set shall be 12+8P.

The one-dimensional curves are stored sequentially. Each curve shall start on a 4-byte boundary. To achieve this, each curve shall be followed by up to three 00h pad bytes as needed.

It is permitted to share data between one-dimensional curves. For example, the offsets for some one-dimensional curves can be identical.

Each curve can be defined by a singleSampledCurve or a segmentedCurve.

11.2.2.2 singleSampledCurve

The singleSampledCurve curve type allows for efficiently defining a single sampled curve segment with simple endpoint extension parameters. The encoding of an extended CLUT Element is defined in Table 108.

Table 108 — singleSampledCurve segment encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'sngf' (736e6766h) type signature	
4…7	4	Reserved, shall be 0	
8…11	4	Number of data entries(N)	uInt32Number
12..15	4	Input value of first entry(F)	float32Number
16..19	4	Input value of last entry(L)	float32Number
20..21	2	Lookup extension type(E)	uInt16Number
22..23	2	Data encoding type	uInt16Number as a valueEncodingType
24…end		Data	Defined by data encoding type

The number of Data entries shall be greater than or equal to two. The first Data entry shall correspond to the input value of the first entry. The last Data entry shall correspond to the input value of the last entry. The value stored for (F) shall be less than the value stored for (L). If more than two Data entries exist then each intermediate entry shall correspond to equidistant sampling between (F) and (L). Linear interpolation shall be used to determine the output value for intermediate input values.

The Lookup extension type defines how to determine output values for input values that are less than (F) or greater than (L).

If the lookup extension type (E) is zero (0) then clipping shall be performed by using the value of the first entry if the input value is less than (F) or using the value of the last entry if the input value is greater than (L).

IF the lookup extension type (E) is one (1) then linear extrapolation shall be used. If the input value is less than (F) then the output shall be determined by the corresponding output value of a line defined by the first and second data entries (and their corresponding input values). If the input value is greater than (L) then the output value shall be determined by the corresponding output value of a line defined by the last two data entries (and their corresponding input values).

NOTE ISO 15076-1 does not include a definition for the use of a singleSampledCurve.

11.2.2.3 segmentedCurve

A segmentedCurve is stored using one or more curve segments, with break-points specified between curve segments. The first curve segment always starts at −∞, and the last curve segment always ends at +∞. The first and last curve segments shall be specified in terms of a formula, whereas the other segments shall be specified either in terms of a formula, or by a sampled curve.

If a curve has a single curve segment, no break-points shall be specified, and the curve shall be specified in terms of a formula.

If a curve has more than one curve segment, break-points shall be specified between curve segments. If there are *n* segments, *n*−1 break-points are specified. The encoding for such a curve is shown in Table 109.

Table 109 — segmentedCurve encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'curf' (63757266h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Number of segment(s) (N)	uInt16Number
10..11	2	Reserved, shall be 0	
12..4N+7	4 × (N-1)	N-1 Break-Points	float32Number[...]
4N+8..end		Segments 1 to N	

Break-points separate two curve segments. The first curve segment is defined between −∞ and break-point 1 (included). The *k*th curve segment (*k* in the range 2 to N−1) is defined between the break-point *k*−1 (not included) and the break-point *k* (included). The Nth curve-segment is defined between break-point N−1 (not included) and +∞. Curve segments that are specified in terms of a formula shall be encoded as shown in Table 110.

Table 110 — Curve segments encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'parf' (70617266h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Encoded value of the function type	uInt16Number
10..11	2	Reserved, shall be 0	
12..end	See Table 111	Parameters (see Table 111)	float32Number[...]

The encoding for the function type field and the parameters is shown in Table 111.

Table 111 — Formula curve segments function encoding

Field length (bytes)	Function type	Encoded value	Parameters
16	$Y = (a * X + b)^{\gamma} + c$	0000h	γ, a, b, c
20	$Y = a * \log (b * X^{\gamma} + c) + d$	0001h	γ, a, b, c, d
20	$Y = a * b^{(c*X+d)} + e$	0002h	a, b, c, d, e
20	$Y = a * (b * X + c)^{\gamma} + d$	0003h	γ, a, b, c, d

NOTE ISO 15076-1 does not include a definition for a curve segment function encoding of 0003h.

The functional inputs and outputs are defined over the values that can be represented as float32Number. The curve-segment shall be defined to result in float32Number values for the entire curve-segment.

Curve segments that are specified as sampled curves shall be encoded as shown in Table 112.

Table 112 — Sampled curve segment encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'samf' (73616D66h) type signature	
4..7	4	Reserved, shall be 0	
8..11	4	Count (N) specifying the number of entries that follow	uInt32Number
12..end	4 × N	Curve entries	float32Number[...]

The count (N) shall be greater than or equal to 1.

The curve samples shall be equally-spaced within the segment, and shall include one break-point, as previously described. If the sampled curve represents the curve-segment between break-point k (B_{Pk}) and break-point k+1 ($B_{P,\ k+1}$), the jth sample ($j \in [1, N]$) shall correspond to the input value $B_{P,\ k} + j\ (B_{P,\ k+1} - B_{P,\ k})\ /\ N$. Thus $B_{P,\ k}$ is excluded.

NOTE The first point used for interpolation of a sampled curve segment is not directly stored in a sampled curve segment.

11.2.3 CLUTElement

The CLUT appears as an n-dimensional array, with each dimension having a number of entries corresponding to the number of grid points.

The CLUT values are arrays of float32Number.

The CLUT is organized as a P-dimensional array with a variable number of grid points in each dimension, where p is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is a Q-float32Number array, where Q is the number of output channels. The first sequential float32Number of the entry contains the function value for the first output function, the second sequential float32Number of the entry contains the function value for the second output function and so on until all of the output functions have been supplied. Formula (9) gives the computation for the byte size of the CLUT.

$$\text{NGrid1} \times \text{NGrid2} \times ... \times \text{NGridP} \times \text{number of output channels } (\text{Q}) \times 4 \tag{9}$$

When used, the byte assignment and encoding for the CLUT shall be as given in Table 113.

Table 113 — CLUTElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'clut' (636C7574h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Number of input channels (P)	uInt16Number
10..11	2	Number of output channels (Q)	uInt16Number
12..27	16	Number of grid points in each dimension. Only the first P entries are used, where P is the number of input channels. Unused entries shall be 00h.	uInt8Number
28..end	See Formula (9)	CLUT data points (arranged as described in the text)	float32Number[...]

The input range for the CLUT is 0,0 to 1,0. For any input value outside this range, the nearest range limit value shall be the input value. The range of the Output Channels is the range of values that can be represented as float32Number.

If the number of grid points in a particular dimension of the CLUT is two, the data for those points shall be set so that the correct results are obtained when linear interpolation is used to generate intermediate values. CLUT elements require a minimum of two grid points for each dimension.

11.2.4 emissionCLUTElement

The emissionCLUTElement encodes spectral emission information as entries of a colour lookup table (CLUT) that are first converted to colorimetric information before applying interpolation to perform colour transformations.

The emissionCLUTElement allows for the encoding of a colour lookup table (CLUT) using flexible and more efficient encoding of values. Values in the CLUT can be encoded using uInt8Number, uInt16Number,

float16Number, and float32Number types. The encoding of an emission CLUT Element is defined in Table 114.

Table 114 — emissionCLUTElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'eclt' (65636c74h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12…17	6	Spectral data wavelength range: start (F), end(E), steps(S)	spectralRange
18..19	2	CLUT Encoding type	uInt16Number as a valueEncodingType
20..35	16	Array containing number of grid points for each input channel (G)	uInt8Number[16]
36…36+M-1	M	Data for CLUT	Defined by CLUT Encoding type
36+M..36+M+S-1	S	Spectral emission of White	Defined by CLUT Encoding type

The number of input channels (P) shall be greater than or equal to one and less than or equal to 16.

The number of output channels (Q) shall be three.

The CLUT appears as a P-dimensional array, with each dimension having a number of entries corresponding to the number of grid points.

The CLUT values are arrays of numbers defining emission vectors determined by the CLUT Encoding type in Formula (10).

$$\text{matrix array} = [e_1, e_2, \ldots, e_S] \tag{10}$$

The spectral data wavelength range field encoded as a spectralRange shall define the starting wavelength (F), ending wavelength (E), and number of steps (S) defined for the reference white emission vector and spectral CLUT elements. The number of steps shall be two or greater.

The CLUT is organized as a P-dimensional array with a variable number of grid points in each dimension, where P is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is an S-number array, where S is the number of steps in the spectral range. The first sequential entry contains the function value for the first emission array, the second sequential entry contains the function value for the second emission array, and so on until all of the emission arrays have been supplied. Formula (11) gives the computation for the byte size of the CLUT.

$$G[0] \times G[1] \times \ldots \times G[P-1] \times \text{number of spectral steps } (S) \times \text{number of encoding type bytes} \tag{11}$$

The reference white emission vector shall be organized as an array of *s* elements with the first element corresponding to the start wavelength (F), and the last element corresponding to the ending wavelength (E) with the intervening S-2 elements corresponding to evenly spaced wavelengths between the starting and ending wavelengths. The array is organized as shown in Formula (12):

$$\text{white array} = [w_1, w_2, \ldots, w_S] \tag{12}$$

The colorimetric conversion of the emissive CLUT shall be determined by applying white normalized observer CMFs associated with the PCC to each CLUT value array, resulting in a CLUT containing value

arrays of tristimulus values. This CLUT shall then be used for transforming input channels to output channels with the output channels being defined by the converted colorimetric arrays in the CLUT, as shown in Formulae (13) to (15).

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = kC \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_S \end{bmatrix}, \tag{13}$$

$$C = \begin{bmatrix} c_{x,1} & c_{x,2} & \cdots & c_{x,S} \\ c_{y,1} & c_{y,2} & \cdots & c_{y,S} \\ c_{z,1} & c_{z,2} & \cdots & c_{z,S} \end{bmatrix}, \tag{14}$$

$$k = \frac{1}{\sum_{i=1}^{S} c_{y,i} w_i} \tag{15}$$

If the observer CMFs have a different sampling range they shall first be resampled to the spectral range defined by the processing element. For CIE relative colorimetric processing, normalization shall be performed by dividing by the scalar product of the reference white vector and the second CMF (c_y).

11.2.5 emissionMatrixElement

The emissionMatrixElement encodes spectral emission information for chromatic primaries and offset that shall be first converted to colorimetric information which forms a matrix that is used for pixel information.

The emission matrix element encoding is shown in Table 115.

Table 115 — emissionMatrixElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'emtx' (656d7478h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Number of input channels (P)	uInt16Number
10..11	2	Number of output channels (Q)	uInt16Number
12..17	6	Spectral data wavelength range: start (F), end (E), steps(S)	spectralRange
18..19	2	Absolute flag	uInt16Number
20..20+ 4×S-1	4xS	Spectral emission of White	float32Number[...]
20+4×S..end	4x(Q+1)xS	Spectral Matrix Elements	float32Number[...]

The number of input channels (P) shall match the number of input channels to the processing element.

The number of output channels (Q) shall be three.

The spectral data wavelength range field encoded as a spectralRange shall define the starting wavelength (F), ending wavelength (E), and number of steps (S) defined for the reference white emission vector and spectral matrix elements. The number of steps shall be two or greater.

The reference white emission vector shall be organized as an array of S elements with the first element corresponding to the start wavelength (F), and the last element corresponding to the ending wavelength

(E) with the intervening S-2 elements corresponding to evenly spaced wavelengths between the starting and ending wavelengths. The array is organized as shown in Formula (16):

white array = [w_1, w_2, …, w_S] (16)

The spectral matrix encoding shall be organized as an array of S × Q elements, and Q is the number of output channels to the matrix. Wavelengths shall be assigned to the elements of each row with the first element corresponding to the start wavelength (F), and the last element corresponding to the ending wavelength (E) with the intervening S-2 elements corresponding to evenly spaced wavelengths between the starting and ending wavelengths. Each matrix array element is a float32Number. The matrix array is organized as shown in Formula (17):

matrix array = [e_{11}, e_{12}, …, e_{1S}, e_{21}, e_{22}, …, e_{2S}, e_{31}, e_{32}, …, e_{3S}, e_1, e_2, …, e_S] (17)

The elements of this array shall first be converted to colorimetric vectors that form a matrix (M) and offset vector, and then the matrix M and offset vector shall be used to perform colour transformations by the processing element as shown in Formula (18):

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = M \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} \tag{18}$$

The range of the input values X_1, X_2, …, X_P and output values Y_1, Y_2, …, Y_Q is the range of values that can be represented as float32Number.

The colorimetric matrix M and offset vector shall be determined by applying white normalized observer CMFs associated the PCC. If the observer CMFs have a different sampling range they shall first be resampled to the spectral range defined by the processing element as shown in Formulae (19) to (21).

$$M = kC \begin{bmatrix} e_{11} & e_{21} & \cdots & e_{P1} \\ e_{12} & e_{12} & \cdots & e_{P2} \\ \vdots & \vdots & \ddots & \vdots \\ e_{1S} & e_{2S} & \cdots & e_{PS} \end{bmatrix}, \tag{19}$$

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = kC \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_S \end{bmatrix}, \tag{20}$$

$$C = \begin{bmatrix} c_{x,1} & c_{x,2} & \cdots & c_{x,S} \\ c_{y,1} & c_{y,2} & \cdots & c_{y,S} \\ c_{z,1} & c_{z,2} & \cdots & c_{z,S} \end{bmatrix} \tag{21}$$

If the Absolute flag is zero (for CIE relative colorimetric processing) then normalization scalar k shall be performed by dividing by the scalar product of the reference white vector and the second CMF (c_y) as shown in Formula (22):

$$k = \frac{1}{\sum_{i=1}^{S} c_{y,i} w_i} \tag{22}$$

Otherwise, if the Absolute flag is set to 1 then the normalization scalar k shall be 1.

11.2.6 emissionObserverElement

The emissionObserverElement transforms input channel data into output channel data as colorimetric representation of the incoming spectral emission.

The emission observer element encoding is shown in Table 116.

Table 116 — emissionObserverElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'eobs' (656f6273h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Number of input channels (P)	uInt16Number
10..11	2	Number of output channels (Q)	uInt16Number
12..17	6	Spectral data wavelength range: start (F), end(E), steps(S)	spectralRange
18..19	2	Absolute flag	uInt16Number
20..20+ 4×S - 1	4×S	Spectral emission of white	float32Number[...]

The number of input channels (P) shall match the number of input channels to the processing element.

The number of output channels (Q) shall be three.

The spectral data wavelength range field encoded as a spectralRange shall define the starting wavelength (F), ending wavelength (E), and number of steps (S) defined for the reference white emission vector and spectral matrix elements. The number of steps shall be two or greater.

The reference white emission vector shall be organized as an array of S elements with the first element corresponding to the start wavelength (F), and the last element corresponding to the ending wavelength (E) with the intervening S-2 elements corresponding to evenly spaced wavelengths between the starting and ending wavelengths. The array is organized as shown in Formula (23):

$$\text{white array} = [w_1, w_2, \ldots, w_S] \tag{23}$$

The colorimetric output channels shall be determined by applying white normalized observer CMFs associated the PCC. If the observer CMFs have a different sampling range they shall first be resampled to the spectral range defined by the processing element. For CIE relative colorimetric processing, normalization shall be performed by dividing by the scalar product of the reference white vector and the second CMF (c_y).

The conversion of input emission spectral channel data to colorimetric output channel data shall be calculated as shown in Formulae (24) to (26):

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = kC \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_S \end{bmatrix}, \tag{24}$$

$$C = \begin{bmatrix} c_{x,1} & c_{x,2} & \cdots & c_{x,S} \\ c_{y,1} & c_{y,2} & \cdots & c_{y,S} \\ c_{z,1} & c_{z,2} & \cdots & c_{z,S} \end{bmatrix}, \tag{25}$$

$$k = \frac{1}{\sum_{i=1}^{S} c_{y,i} w_i} \tag{26}$$

11.2.7 extendedCLUTElement

The extendedCLUTElement allows for the encoding of a colour lookup table (CLUT) using a flexible and more efficient encoding of values. Values in the CLUT can be encoded using uInt8Number, uInt16Number, float16Number, and float32Number types. The encoding of an extended CLUT Element is defined in Table 117.

Table 117 — extendedCLUTElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'xclt' (78636c74h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12..15	4	CLUT encoding type	uInt32Number as a valueEncodingType
16..31	16	Number of grid points for each input channel (G)	uInt8Number[16]
32…end		Data for CLUT	Defined by CLUT Encoding type

The number of input channels (P) shall be greater than or equal to 1 and less than or equal to 16.

The CLUT appears as an *n*-dimensional array, with each dimension having a number of entries corresponding to the number of grid points.

The CLUT values are arrays of numbers determined by the CLUT encoding type.

The CLUT is organized as a P-dimensional array with a variable number of grid points in each dimension, where P is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is a Q-number array, where Q is the number of output channels. The first sequential entry contains the function value for the first output function, the second sequential entry contains the function value for the second output function and so on until all of the output functions have been supplied. Formula (27) gives the computation for the byte size of the CLUT.

$$\begin{aligned} &\text{NGrid1} \times \text{NGrid2} \times \ldots \times \text{NGridP} \times \text{number of output channels } (\text{Q}) \\ &\times \text{number of encoding type bytes} \end{aligned} \tag{27}$$

11.2.8 inverseEmissionMatrixElement

The inverseEmissionMatrixElement encodes spectral emission row vector information for three trichromatic primaries and offset that shall be first converted to colorimetric column vector information which forms a matrix that is inverted before applying pixel information.

The inverse emission matrix element shall be encoded as shown in Table 118.

Table 118 — inverseEmissionMatrixElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'iemx' (69656d78h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Number of input channels (P)	uInt16Number
10..11	2	Number of output channels (Q)	uInt16Number
12..17	6	Spectral data wavelength range: start (F), end (E), steps(S)	spectralRange
18..19	2	Reserved, shall be 0	
20..20+ 4×S - 1	4×S	Spectral emission of white	float32Number[...]
20+ 4×S .. end	4× (Q+1) ×S	Spectral matrix elements	float32Number[...]

The number of input channels (P) shall be 3.

The number of output channels (Q) shall be 3.

The spectral data wavelength range field encoded as a spectralRange shall define the starting wavelength (F), ending wavelength (E), and number of steps (S), defined for the reference white emission vector and spectral matrix elements. The number of steps shall be two or greater.

The reference white emission vector shall be organized as an array of S elements, with the first element corresponding to the start wavelength (F), and the last element corresponding to the ending wavelength (E), and with the intervening S-2 elements corresponding to evenly spaced wavelengths between the starting and ending wavelengths. The array is organized as shown in Formula (28):

$$\text{white array} = [w_1, w_2, \ldots, w_S] \tag{28}$$

The spectral matrix encoding shall be organized as an array of S × Q element, and Q is the number of output channels to the matrix. Wavelengths shall be assigned to the elements of each row with the first element corresponding to the start wavelength (F), and the last element corresponding to the ending wavelength (E) with the intervening S-2 elements corresponding to evenly spaced wavelengths between the starting and ending wavelengths. Each matrix array element is a float32Number. The matrix array is organized as shown in Formula (29):

$$\text{matrix array} = [e_{11}, e_{12}, \ldots, e_{1S}, e_{21}, e_{22}, \ldots, e_{2S}, e_{31}, e_{32}, \ldots, e_{3S}, e_1, e_2, \ldots, e_S] \tag{29}$$

The elements of this array shall first be converted to colorimetric vectors that form a matrix (M) and offset vector, and then the inverse of the colorimetric matrix M^{-1} and offset vector shall be used to perform colour transformations by the processing element as shown in Formula (30):

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = (M)^{-1} \left(\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} - \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} \right) \tag{30}$$

The range of the input values $X_1, X_2, \ldots, X_P$ and output values $Y_1, Y_2, \ldots, Y_Q$ is the range of values that can be represented as float32Number.

The colorimetric matrix M and offset vector shall be determined by applying white normalized observer CMFs associated the PCC. If the observer CMFs have a different sampling range they shall first be resampled to the spectral range defined by the processing element. For CIE relative colorimetric

processing, normalization shall be performed by dividing by the scalar product of the reference white vector and the second CMF (c_y), as shown in Formulae (31) to (33).

$$M = kC \begin{bmatrix} e_{11} & e_{21} & e_{31} \\ e_{12} & e_{22} & e_{32} \\ \vdots & \vdots & \vdots \\ e_{1S} & e_{2S} & e_{3S} \end{bmatrix}, \tag{31}$$

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = kC \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_S \end{bmatrix}, \tag{32}$$

$$C = \begin{bmatrix} c_{x,1} & c_{x,2} & \cdots & c_{x,S} \\ c_{y,1} & c_{y,2} & \cdots & c_{y,S} \\ c_{z,1} & c_{z,2} & \cdots & c_{z,S} \end{bmatrix} \tag{33}$$

If the Absolute flag is zero (for CIE relative colorimetric processing) then normalization scalar *k* shall be calculated by dividing by the scalar product of the reference white vector and the second CMF (c_y) as shown in Formula (34):

$$k = \frac{1}{\sum_{i=1}^{S} c_{y,i} w_i} \tag{34}$$

Otherwise, if the Absolute flag is set to 1 then the normalization scalar *k* shall be 1.

11.2.9 JabToXYZElement

The XYZToJabElement allows for the encoding of appearance parameters for the purpose of converting from colorimetry under the viewing conditions to CIECAM02 Cartesian appearance correlates Jab.

The encoding of a JabToXYZElement is shown in Table 119.

Table 119 — JabToXYZElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'JtoX' (4a746f58h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12…23	12	White point XYZ	floatXYZNumber
24…27	4	Luminance in cd/m^2	float32Number
28…31	4	Background luminant in cd/m^2	float32Number
32…35	4	Impact of surround (ranging from 0,0 to 1,0)	float32Number
36..39	4	Chromatic induction factor	float32Number
40…43	4	Adaptation factor	float32Number

Both the number of input channels (P) and number of output channels (Q) shall be three.

The logic to convert XYZToJab is given in Appendix C.

11.2.10 matrixElement

The matrix is organized as an array of P × Q elements, where P is the number of input channels to the matrix, and Q is the number of output channels. Each matrix elements is a float32Number. The array is organized as shown in Formula (35):

$$\text{array} = [e_{11}, e_{12}, \ldots, e_{1P}, e_{21}, e_{22}, \ldots, e_{2P}, \ldots, e_{Q1}, e_{Q2}, \ldots, e_{QP}, e_1, e_2, \ldots, e_Q] \quad (35)$$

The matrix element encoding is shown in Table 120.

Table 120 — Matrix Element encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'matf' (6D617466h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Number of input channels (P)	uInt16Number
10..11	2	Number of output channels (Q)	uInt16Number
12..end	4x(P+1)xQ	Matrix elements	float32Number[...]

The matrix is used to convert data to a different colour space, according to Formula (36):

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \ldots \\ Y_Q \end{bmatrix} = \begin{bmatrix} e_{11} & e_{12} & \ldots & e_{1P} \\ e_{21} & e_{22} & \ldots & e_{2P} \\ \ldots & \ldots & \ldots & \ldots \\ e_{Q1} & e_{Q2} & & e_{QP} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ \ldots \\ X_P \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \ldots \\ e_Q \end{bmatrix} \quad (36)$$

The range of the input values $X_1, X_2, \ldots, X_P$ and output values $Y_1, Y_2, \ldots, Y_Q$ is the range of values that can be represented as float32Number.

11.2.11 sparseMatrixElement

The sparseMatrixElement is organized as a P-dimensional LUT with a variable number of grid points in each dimension, where P is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is a sparse matrix of B bytes.

When used, the byte assignment and encoding for the sparseMatrixElement shall be as given in Table 121.

Table 121 — sparseMatrixElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'smet' (736d6574h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of equivalent output channels reserved for internal sparse matrix encoding (Q)	uInt16Number
12..13	2	Sparse matrix LUT encoding type	sparseMatrixEncodingType

Byte position	Field length (bytes)	Content	Encoded as…
14..15	2	Reserved, shall be 0	
16..31	16	Number of grid points for each input channel (G)	uInt8Number[16]
32…end		LUT of sparse matrices	sparseMatrixUInt8[] or sparseMatrixUInt16[] or sparseMatrixFloat16[] or sparseMatrixFloat32[]

The equation for computing the number of sparse matrices in the LUT of sparse matrices is as shown in Formula (37):

numMatrices = NGrid1 * NGrid2 *... * NGridP (37)

The sparse matrices encoded in the LUT of sparse matrices shall use compact padding resulting in the Matrix Entry Data Values and end of each sparse matrix being aligned on a 4-byte boundary.

All sparse matrices in the sparseMatrixElement shall have the same number of rows and columns.

11.2.12 reflectanceCLUTElement

The reflectanceCLUTElement encodes spectral reflectance information as entries of a colour lookup table (CLUT) that are first converted to colorimetric information before applying interpolation to perform colour transformations.

The reflectanceCLUTElement allows for the encoding of a colour lookup table (CLUT) using flexible and more efficient encoding of values. Values in the CLUT can be encoded using uInt8Number, uInt16Number, float16Number, and float32Number types. The encoding of a reflectanceCLUTElement is defined in Table 122.

Table 122 — reflectanceCLUTElement encoding

Byte position	Field length (bytes)	Content	Encoded as…
0…3	4	'rclt' (72636c74h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12..15	4	Flags	uInt16Number
16..21	6	Spectral data wavelength range: start (F), end (E), steps (S)	spectralRange
22..23	2	CLUT Encoding type (T)	uInt16Number as a valueEncodingType
24..39	16	Number of grid points for each input channel (G)	uInt8Number[16]
40…40+M-1	M	Data for CLUT	Defined by CLUT Encoding type (T)
40+M..40+M+N-1	N	Spectral reflectance of white	Defined by CLUT Encoding type (T)

The number of input channels (P) shall be greater than or equal to one and less than or equal to 16.

The number of output channels (Q) shall be 3.

The CLUT appears as an *n*-dimensional array, with each dimension having a number of entries corresponding to the number of grid points.

The CLUT values are arrays of numbers defining reflectance vectors determined by the CLUT Encoding type (T), as shown in Formula (38).

$$\text{matrix array} = [r_1, r_2, \ldots, r_S] \tag{38}$$

The spectral data wavelength range field encoded as a spectralRange shall define the starting wavelength (F), ending wavelength (E), and number of steps (S) defined for the reference white media vector and spectral CLUT elements. The number of steps shall be two or greater.

The CLUT is organized as a P-dimensional array with a variable number of grid points in each dimension, where P is the number of input channels in the transform. The dimension corresponding to the first channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value is an S-number array, where S is the number of steps in the spectral range. The first sequential entry contains the function value for the first reflectance array, the second sequential entry contains the function value for the second reflectance array and so on until all of the reflectance arrays have been supplied. Formula (39) gives the computation for the byte size of the CLUT.

$$\text{NGrid1} \times \text{NGrid2} \times \ldots \times \text{NGridP} \times \text{number of spectral steps (S)} \times \text{number of encoding type bytes} \tag{39}$$

The reference white reflectance vector shall be organized as an array of *s* elements with the first element corresponding to the start wavelength (F), and the last element corresponding to the ending wavelength (E) with the intervening S-2 elements corresponding to evenly spaced wavelengths between the starting and ending wavelengths. The array is organized as shown in Formula (40):

$$\text{white array} = [w_1, w_2, \ldots, w_S] \tag{40}$$

The colorimetric conversion of the reflectance CLUT shall be determined by applying normalized observer CMFs associated the PCC to each CLUT value array resulting in a CLUT containing value arrays of tristimulus values. This CLUT shall then be used for transforming input channels to output channels with the output channels being defined by the converted colorimetric arrays in the CLUT, as shown in Formulae (41) and (42).

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = kC \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_S \end{bmatrix}, \tag{41}$$

$$C = \begin{bmatrix} c_{x,1}l_1 & c_{x,2}l_2 & \cdots & c_{x,S}l_S \\ c_{y,1}l_1 & c_{y,2}l_2 & \cdots & c_{y,S}l_S \\ c_{z,1}l_1 & c_{z,2}l_2 & \cdots & c_{z,S}l_S \end{bmatrix}, \text{ and} \tag{42}$$

$$k = \frac{1}{\sum_{i=1}^{S} c_{y,i} w_i}$$

where

c_x, c_y and c_z are normalized observer colour matching functions with *S* entries.

The spectral reflectances and observer CMFs shall be resampled if they differ from the sampling range of the illuminant.

For relative colorimetric processing (when the Absolute flag is zero) the illuminant white point from the PCC and media white point (calculated based on the white array, observer and illuminant from the PCC used to determine the initial reflectance CLUT colorimetry) shall be used additionally to normalize each of the entries in the CLUT, as shown in Formulae (43) to (46).

$$\begin{bmatrix} O_1' \\ O_2' \\ O_3' \end{bmatrix} = M \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix}, \tag{43}$$

$$M = \begin{bmatrix} l_{w,x} / m_{w,x} & 0 & 0 \\ 0 & l_{w,y} / m_{w,y} & 0 \\ 0 & 0 & l_{w,z} / m_{w,z} \end{bmatrix}, \tag{44}$$

$$l_{w,x} = \sum_{i=1}^{S} c_{x,i} l_i \text{ , } l_{w,y} = \sum_{i=1}^{S} c_{y,i} l_i \text{ , } l_{w,z} = \sum_{i=1}^{S} c_{z,i} l_i \text{ , and} \tag{45}$$

$$\begin{bmatrix} m_{w,x} \\ m_{w,y} \\ m_{w,z} \end{bmatrix} = kC \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_S \end{bmatrix} \tag{46}$$

where

l is the spectral power distribution of the PCC illuminant white point.

11.2.13 reflectanceObserverElement

The reflectanceObserverElement transforms input channel data into output channel data as colorimetric representation of the incoming spectral reflectance.

The emission observer element encoding is shown in Table 123.

Table 123 — reflectanceObserverElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0..3	4	'robs' (726f6273h) type signature	
4..7	4	Reserved, shall be 0	
8..9	2	Number of input channels (P)	uInt16Number
10..11	2	Number of output channels (Q)	uInt16Number
12..17	6	Spectral data wavelength range: start (F), end(E), steps(S)	spectralRange
18..19	2	Absolute flag	uInt16Number
20..20+ 4×S - 1	4×S	Spectral reflectance of white	float32Number[...]

The number of input channels (P) shall match the number of input channels to the processing element.

The number of output channels (Q) shall be three.

The spectral data wavelength range field encoded as a spectralRange shall define the starting wavelength (F), ending wavelength (E), and number of steps (S) defined for the reference white emission vector and spectral matrix elements. The number of steps shall be two or greater.

The reference white reflectance vector shall be organized as an array of S elements with the first element corresponding to the start wavelength (F), and the last element corresponding to the ending wavelength (E) with the intervening S-2 elements corresponding to evenly spaced wavelengths between the starting and ending wavelengths. The array is organized as shown in Formula (47):

$$\text{white array} = [w_1, w_2, \ldots, w_S] \tag{47}$$

The colorimetric conversion of incoming reflectance vectors shall be determined by applying normalized observer CMFs associated the PCC to each CLUT value array resulting in a CLUT containing value arrays of tristimulus values. This CLUT shall then be used for transforming input channels to output channels with the output channels being defined by the converted colorimetric arrays in the CLUT, as shown in Formulae (48) to (50).

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = kC \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_S \end{bmatrix}, \tag{48}$$

$$C = \begin{bmatrix} c_{x,1}l_1 & c_{x,2}l_2 & \cdots & c_{x,S}l_S \\ c_{y,1}l_1 & c_{y,2}l_2 & \cdots & c_{y,S}l_S \\ c_{z,1}l_1 & c_{z,2}l_2 & \cdots & c_{z,S}l_S \end{bmatrix}, \text{ and} \tag{49}$$

$$k = \frac{1}{\sum_{i=1}^{S} c_{y,i}w_i} \tag{50}$$

The spectral reflectances and observer CMFs shall be resampled if they differ from the sampling range of the illuminant.

For relative colorimetric processing (when the Absolute flag is zero) the illuminant white point from the PCC and media white point (calculated based on the white array, observer and illuminant from the PCC used to determine the initial reflectance colorimetry) shall be used to additionally normalize the output colorimetry, as shown in Formulae (51) to (54).

$$\begin{bmatrix} O_1' \\ O_2' \\ O_3' \end{bmatrix} = M \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix}, \tag{51}$$

$$M = \begin{bmatrix} l_{w,x}/m_{w,x} & 0 & 0 \\ 0 & l_{w,y}/m_{w,y} & 0 \\ 0 & 0 & l_{w,z}/m_{w,z} \end{bmatrix}, \tag{52}$$

$$l_{w,x} = \sum_{i=1}^{S} c_{x,i}l_i \text{ , } l_{w,y} = \sum_{i=1}^{S} c_{y,i}l_i \text{ , } l_{w,z} = \sum_{i=1}^{S} c_{z,i}l_i \text{ , and} \tag{53}$$

$$\begin{bmatrix} m_{w,x} \\ m_{w,y} \\ m_{w,z} \end{bmatrix} = kC \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_S \end{bmatrix} \tag{54}$$

11.2.14 tintArrayElement

The tint tintArrayElement allows for the encoding of a one-dimensional input to *n*-dimensional output colour lookup transform using flexible encoding of values.

NOTE This processing element differs from the segmentedCurve, which defines independent 1-dimension transformations for *n*-dimensional input, thus resulting in an N-dimension to N-dimension transform.

Values in the tint array can be encoded using uInt8Number, uInt16Number, float16Number, and float32Number types. The range of uInt8Number and uInt16Number values shall correspond to output channel values ranging from 0,0 to 1,0. The encoding of a tintArrayElement is defined in Table 124.

Table 124 — tintArrayElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'tint' (74696e74h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12..15	4	Tint encoding type	uInt32Number as a valueEncodingType
16..19	4	Reserved, shall be 0	
20…end		Tint array	Defined by Tint Encoding type

The number of input channels (P) shall be one.

The number of entries in the tint array shall be evenly divisible by the number of Output Channels (Q) and there shall be at least 2Q entries in the TintArray.

The first Q values in the tint array shall define the output channel values for an input tint of 0,0. The last Q values in the tint array shall define the output channel values for an input tint of 1,0. Intermediate entries in the tint array shall define output values for a uniform sampling of the input tint value.

Output values for intermediate tint values shall be defined using linear interpolation of corresponding channel entries.

If an input tint is less than 0,0 then the output shall be for an input tint of 0,0. If an input tint is greater than 1,0 then the output shall be for an input tint of 1,0.

11.2.15 XYZToJabElement

The XYZToJabElement allows for the encoding of appearance parameters for the purpose of converting from colorimetry under the viewing conditions to CIECAM02 Cartesian appearance correlates Jab.

The encoding of an XYZToJabElement is shown in Table 125.

Table 125 — XYZToJabElement encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'XtoJ' (58746f4ah) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12…23	12	White Point XYZ	floatXYZNumber
24…27	4	Luminance in cd/m^2	float32Number
28…31	4	Background luminance in cd/m^2	float32Number
32…35	4	Impact of surround (ranging from 0,0 to 1,0)	float32Number
36..39	4	Chromatic induction factor	float32Number
40…43	4	Adaptation factor	float32Number

Both the number of input channels (P) and number of output channels (Q) shall be three.

The logic to convert XYZToJab is given in Annex C.

11.2.16 "Future" expansion elements

The 'bACS' and 'eACS' element types were provided in version 4.3 as placeholders for future expansion. The intent of these elements has been superseded by the full specification of a spectrally-based PCS in this document. If used, these elements shall be considered as pass through elements with no modification of channel data. Their encoding shall be as shown in Table 126 and Table 127.

Table 126 — bACS element encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'bACS' (62414353h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12…15	4	Signature	

Table 127 — eACS element encoding

Byte position	Field length (bytes)	Content	Encoded as...
0…3	4	'eACS' (65414353h) type signature	
4…7	4	Reserved, shall be 0	
8…9	2	Number of input channels (P)	uInt16Number
10…11	2	Number of output channels (Q)	uInt16Number
12…15	4	Signature	

For both the 'bACS' and 'eACS' element types the number of input channels (P) shall be the same as the number of output channels (Q).

12 Struct tag type definitions

12.1 General

The tagStructType tag type provides the means of associating multiple tag elements into a single data structure with each contained sub-tag element having a unique signature. Each tagStructType has a Structure Type Identifier that shall be used to identify the possible sub-tag element s and the purposes for each sub-tag element in the structure.

The public tagStructType tag structure types defined by the ICC are listed in 12.2 in alphabetical order.

12.2 Struct tag type listing

12.2.1 brdfTransformStructure

12.2.1.1 General

Structure Type Identifier: 'brdf' (62726466h).

The brdfTransformStructure defines information used to transform device values to BRDF parameters that can be used to simulate colour appearance under various viewing/illumination geometries. Table 128 shows publically defined element sub-tag members of a brdfStructure. Descriptions for each sub-tag member can be found in 12.2.1.2.

Table 128 — brdfTransformStructure element sub-tags

Id	Signature	Description	Sub-tag type	Use
brdfTypeMbr	'type' (74797065h)	Type of BRDF colour – ‘mono’ or ‘colr’ (see 12.2.1.2.1)	signature	Required
brdfFunctionMbr	'func' (66756e63h)	BRDF function signature (see 12.2.1.2.2)	signature	Required
brdfParamsPerChannelMbr	'nump' (6e756d70h)	Number of BRDF parameters stored per output channel in xform (see 12.2.1.2.3)	uInt16Number	Required
brdfTransformMbr	'xfrm' (7866726dh)	The transform (see 12.2.1.2.4)	multiProcessElement	Required

12.2.1.2 brdfTransformStructure sub-tag member elements

12.2.1.2.1 brdfTypeMbr

Tag element signature: 'type' (74797065h).

Permitted tag element types: signature.

Element usage: required.

The brdfTransformStructure *brdfTypeMbr* element defines the type of BRDF transform.

If the *brdfTypeMbr* sub-tag contains the signature ‘mono’ (6d6f6e6fh) then the brdfTransform sub-tag shall be assumed to define output BRDF parameters using only a single channel of data. The total number of parameters shall be defined by the value stored in the paramsPerChannel sub-tag element.

If the *brdfTypeMbr* sub-tag contains the signature ‘colr’ (636f6c72h) then the brdfTransform sub-tag shall be assumed to define output BRDF parameters for each channel defined by the associated PCS

elements in the header. The total number of parameters for each output entry shall be defined as the number of PCS channels multiplied by the value stored in the paramsPerChannel sub-tag element.

12.2.1.2.2 brdfFunctionMbr

Tag element signature: 'func' (66756e63h).

Permitted tag element types: signature.

Element usage: required.

The brdfTransformStructure *brdfFunctionMbr* element shall contain a signature representing the BRDF function to be used. Table 129 shows the signatures and function types that are defined. Additional BRDF function signatures may be registered at the ICC signature registry at http://www.color.org (see Clause 5).

Table 129 — brdfTransformStructure brdfTypeMbr signatures

Signature	Description	Implied number of parameters
'BPh0' (42506830h)	Blinn-Phong with monochrome parameters	4
'BPh1' (42506831h)	Blinn-Phong with full colour parameters	3
'CT10' (43543130h)	Cook-Torrance with 1 lobe and monochrome parameters	6
'CT20' (43543230h)	Cook-Torrance with 2 lobes and monochrome parameters	9
'CT30' (43543330h)	Cook-Torrance with 3 lobes and monochrome parameters	11
'CT11' (43543131h)	Cook-Torrance with 1 lobe and full colour parameters	5
'CT21' (43543231h)	Cook-Torrance with 2 lobes and full colour parameters	8
'CT31' (43543331h)	Cook-Torrance with 3 lobes and full colour parameters	10
'War0' (57617230h)	Ward with monochrome parameters	5
'War1' (57617231h)	Ward with full colour parameters	4
'La10' (4c613130h)	Lafortune with 1 lobe and monochrome parameters?	9
'La20' (4c613230h)	Lafortune with 2 lobes and monochrome parameters?	16
'La30' (4c613330h)	Lafortune with 3 lobes and monochrome parameters?	23
'La11' (4c613131h)	Lafortune with 1 lobe and chromatic parameters?	5
'La21' (4c613231h)	Lafortune with 2 lobes and chromatic parameters?	9
'La31' (4c613331h)	Lafortune with 3 lobes and chromatic parameters?	13

The lighting equation used by the Blinn-Phong reflectance model is as shown in Formula (55):

$$I_P = \sum_{m \in \text{lights}} \left(k_d \left(L_m \cdot N \right) i_{m,d} + k_s \left(N \cdot H_m \right)^n i_{m,s} \right) \tag{55}$$

where

$i_{m,d}$ is the intensity of the diffuse component of light in normalised digital counts;

$i_{m,s}$ is the intensity of the specular component of light in normalised digital counts;

L_m is the direction vector from the light to the location on the surface;

N is the normal for the location on the surface;

H_m is the direction vector midway between *L* and the viewpoint vector *V*;

I_P is the total light reflected from the surface of the object towards the viewer in normalised digital counts.

The following are the Blinn-Phong parameters that specify the material:

k_d is the diffuse reflection constant for the material;

k_s is the specular reflection constant for the material;

n is the shininess constant for the material.

For the full colour Blinn-Phong function the three parameters shall be k_d, k_s, and n. The order of the parameters in the transform shall be:

k_d, k_s, n.

The monochrome function combines the output of the absolute transform with three parameters to compute the Blinn-Phong parameters k_d and k_s, as shown in Formulae (56) and (57).

$$k_d = I_d B \tag{56}$$

$$k_s = l_s B + I_{gs} \tag{57}$$

where

B is the output of the absolute transform;

l_d is the diffuse scaling factor;

l_s is the specular scaling factor;

l_{gs} is a global specular component.

The order of the four parameters in the transform shall be l_d, l_s, l_{gs}, n.

The documentation for the Blinn-Phong reflectance model can be found in James F. Blinn, 'Models of light reflection for computer synthesized pictures'. Proc. 4th annual conference on computer graphics and interactive techniques, 1977, pp. 192–198.

The lighting equation used by the Cook-Torrance reflectance model is as shown in Formulae (58) to (60):

$$I_P = \sum_{m \in \text{lights}} i_m \left(k_d \left(L_m \cdot N \right) + k_s \rho_{s,m} \right) \tag{58}$$

where $\rho_{s,m} = \dfrac{F_m}{\pi} \dfrac{D_m G_m}{\left(N \cdot L_m\right)\left(N \cdot V\right)}$

and:

$$G_m = \min\left\{1, \frac{2\left(N \cdot H_m\right)\left(N \cdot V\right)}{\left(V \cdot H_m\right)}, \frac{2\left(N \cdot Hm\right)\left(N \cdot L_m\right)}{\left(V \cdot H_m\right)}\right\} \tag{59}$$

$$D_m = \frac{1}{m^2 \cos^4 \alpha_m} e^{-\left(\frac{\tan \alpha_m}{m}\right)^2} \quad (60)$$

where

i_m is the average incident light intensity, in normalised digital counts;

L_m is the direction vector from the light to the location on the surface;

F_m is the Fresnel term for the material;

N is the normal for the location on the surface;

H_m is the direction vector midway between L and the viewpoint vector V;

V is the viewpoint vector;

α is the angle between N and H.

F_m describes how light is reflected from each smooth microfacet, and can be obtained from the Fresnel equations. The parameters to the Fresnel Equations are index of refraction (n), extinction coefficient (k), and angle of illumination.

The parameters for the Cook-Torrance model are summarized thus:

k_d is the diffuse reflection constant for the material;

k_s is the specular reflection constant for the material;

m is the rms slope;

n is the index of refraction;

k is the extinction coefficient (or attenuation coefficient) in m^{-1}.

The order of the parameters in the transform for the full colour function shall be k_d, k_s, m, n, k.

The monochrome function combines the output of the absolute transform with three parameters to compute the Cook-Torrance parameters k_d and k_s, as shown in Formulae (61) and (62):

$$k_d = l_d B \quad (61)$$

$$k_s = l_s B + l_{gs} \quad (62)$$

The order of the parameters for the monochrome single lobe Cook-Torrance function shall be l_d, l_s, l_{gs}, m, n, k.

For multi-lobe versions of the function D is defined according to Formula (63):

$$D = \sum_j w_j D\left(m_j\right) \quad (63)$$

where

m_j is the rms slope for the jth lobe;

w_j is the weighting for the lobe.

The parameters for the full colour two lobed Cook-Torrance function shall be present in the following order: k_d, k_s, m_1, w_1, m_2, w_2, n, k.

The parameters for the monochrome two lobed Cook-Torrance function shall be present in the following order: l_d, l_s, l_{gs}, m_1, w_1, m_2, w_2, n, k.

The parameters for the full colour three lobed Cook-Torrance function shall be present in the following order: k_d, k_s, m_1, w_1, m_2, w_2, m_3, w_3, n, k.

The parameters for the monochrome three lobed Cook-Torrance function shall be present in the following order: l_d, l_s, l_{gs}, m_1, w_1, m_2, w_2, m_3, w_3, n, k.

Documentation for the Cook-Torrance model can be found in R. Cook and K. Torrance, 'A reflectance model for computer graphics' http://inst.eecs.berkeley.edu/~cs283/sp13/lectures/cookpaper.pdf. *Computer Graphics* (SIGGRAPH '81 Proceedings), Vol. 15, No. 3, July 1981, pp. 301–316.

The lighting equation used by the isotropic Ward reflectance model is as shown in Formulae (64):

$$I_P = \sum_{m \in \text{lights}} i_m \left(k_d \left(L_m \cdot N \right) + k_s \rho_{s,m} \right) \tag{64}$$

where

$$\rho_{s,m} = \frac{1}{\sqrt{\left(N \cdot L_m\right)\left(N \cdot R_m\right)}} \frac{N \cdot L}{4\pi \alpha_x \alpha_y} \exp\left[-2 \frac{\left(\frac{H_m \cdot X}{\alpha_x} \right)^2 + \left(\frac{H_m \cdot Y}{\alpha_x} \right)^2}{1 + \left(H_m \cdot N \right)} \right];$$

R_m is the vector of light reflection for light m;

X and Y are orthogonal vectors in the normal plane which specify anisotropic directions.

The parameters α_x and α_y control the shininess in two dimensions. When these parameters are not equal the Ward model is anisotropic and when they are equal the model is isotropic.

The parameters for the full colour Ward function shall be present in the following order: k_d, k_s, α_x, α_y.

The monochrome function combines the output of the absolute transform with three parameters to compute the Ward parameters k_d and k_s as shown in Formulae (65) and (66):

$$k_d = l_d B \tag{65}$$

$$k_s = l_s B + l_{gs} \tag{66}$$

The order of the parameters for the monochrome Ward function shall be: l_d, l_s, l_{gs}, m, n, k.

Documentation for the Ward model can be found in G. Ward, 'Measuring and Modeling Anisotropic Reflection.' *Computer Graphics*, Vol. 26, No. 2, July 1992, pp. 265–272.

The lighting equation used by the Lafortune BRDF model is as shown in Formula (67):

$$I_p = \sum_{m \in \text{lights}} i_m \left(k_d \left(L_m \cdot N \right) + \rho_{s,m} \right) \tag{67}$$

where

$$\rho_{s,m} = \sum_{i \in \text{lobes}} \left[C_{x,i}\mu_{x,m}v_x + C_{y,i}\mu_{y,m}v_y + C_{z,i}\mu_{z,m}v_z \right]^{n_i} ;$$

$L_m = [\mu_{x,m}, \mu_{y,m}, \mu_{z,m}]$;

v_x, v_y and v_z are the components of the viewing vector V in the x, y and z dimensions, respectively.

The monochrome function combines the output of the absolute transform with three parameters to compute the Lafortune parameters k_d and C_x,C_y, *and* C_z, as shown in Formulae (68) to (71):

$$k_d = l_d B \tag{68}$$

$$C_{x,i} = l_{s,x,i} B + l_{gs,x,i} \tag{69}$$

$$C_{y,i} = l_{s,y,i} B + l_{gs,y,i} \tag{70}$$

$$C_{z,i} = l_{s,z,i} B + l_{gs,z,i} \tag{71}$$

The parameters for the full colour single lobe Lafortune function shall be present in the following order: K_d, C_x, C_y, C_z, n.

The parameters for the full colour single lobe Lafortune function shall be present in the following order: K_d, l_d, $l_{s,x}$, $l_{s,y}$, $l_{s,z}$, $l_{gs,x}$, $l_{gs,y}$, $l_{gs,z}$, n.

The parameters for the full colour two lobe Lafortune function shall be present in the following order: K_d, $C_{x,1}$, $C_{y,1}$, $C_{z,1}$, $n_{,1}$, $C_{x,2}$, $C_{y,2}$, $C_{z,2}$, $n_{,2}$.

The parameters for the full colour two lobe Lafortune function shall be present in the following order: K_d ,l_d, $l_{s,x,1}$, $l_{s,y,1}$, $l_{s,z,1}$, $l_{gs,x,1}$, $l_{gs,y,1}$, $l_{gs,z,1}$, $n_{1,}$ $l_{s,x,2}$, $l_{s,y,2}$, $l_{s,z,2}$, $l_{gs,x,2}$, $l_{gs,y,2}$, $l_{gs,z,2}$, n_2.

The parameters for the full colour three lobe Lafortune function shall be present in the following order: K_d, $C_{x,1}$, $C_{y,1}$, $C_{z,1}$, $n_{,1}$, $C_{x,2}$, $C_{y,2}$, $C_{z,2}$, $n_{,2}$, $C_{x,3}$, $C_{y,3}$, $C_{z,3}$, $n_{,3}$.

The parameters for the full colour three lobe Lafortune function shall be present in the following order: K_d, l_d, $l_{s,x,1}$, $l_{s,y,1}$, $l_{s,z,1}$, $l_{gs,x,1}$, $l_{gs,y,1}$, $l_{gs,z,1}$, $n_{1,}$ $l_{s,x,2}$, $l_{s,y,2}$, $l_{s,z,2}$, $l_{gs,x,2}$, $l_{gs,y,2}$, $l_{gs,z,2}$, $n_{2,}$ $l_{s,x,3}$, $l_{s,y,2}$, $l_{s,z,3}$, $l_{gs,x,3}$, $l_{gs,y,3}$, $l_{gs,z,3}$, n_3.

Documentation for the Lafortune model can be found in Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance and Donald P. Greenberg, 'Non-linear approximation of reflectance functions.' SIGGRAPH 97 Conference Proceedings, Annual Conference Series, pp. 117–126.

The value in the brdfParamsPerChannel sub-tag shall be greater than or equal to the number of parameters needed by the BRDF.

12.2.1.2.3 brdfParamsPerChannelMbr

Tag element signature: 'nprm' (6e70726dh).

Permitted tag element types: uInt16Number.

Element usage: required.

The brdfTransformStructure *brdfParamsPerChannelMbr* element shall contain an integer specifying the number of parameters that are stored for each output channel. This integer shall match the implied number of parameters indicated in Table 129.

12.2.1.2.4 brdfTransformMbr

Tag element signature: 'xfrm' (7866726dh).

Permitted tag element type: multiProcessElementType.

Element usage: required.

The brdfTransformStructure *brdfTransformMbr* element shall contain a multiProcessElementType subtag that provides a transform from device values to BRDF parameters for each output channel. If the brdfTransformStructure *brdfTypeMbr* element contains the 'mono' signature then the transform element shall provide only a single set of BRDF parameters representing the reflectance properties of the material. Otherwise the transform element shall provide a set of BRDF parameters for each channel in the associated PCS with the brdfTransformStructure tag.

12.2.2 colorantInfoStructure

12.2.2.1 General

Structure Type signature identifier: 'cinf' (63696e66h).

A colorInfoStructure is used by the colorantTableTag and colorantOutTableTag to define relevant information about the colorants used by the profile. Publically defined elements for sub-tag members of a colorantInfoStructure are shown in Table 130. Descriptions for each sub-tag member can be found in 12.2.2.2.

Table 130 — colorantInfoStructure element sub-tags

Member	Signature	Description	Sub-tag type	Use
cinfNameMbr	'name' (6e616d65h)	Name of named colour (see 12.2.2.2.1)	utf8Type	Shall be present
cinfLocalizedNamesMbr	'lcnm' (6c636e6dh)	Localized names of Named colour (see 12.2.2.2.2)	multiLocalizedUnicodeType	May be present
cinfPcsDataMbr	'pcs ' (70637320h)	PCS values associated with colour (see 12.2.2.2.3)	uInt8Number uInt16Number float16Number float32Number	May be present
cinfSpectralDataMbr	'spec' (73706563h)	Spectral values associated with colour (see 12.2.2.2.4)	uInt8Number uInt16Number float16Number float32Number sparseMatrixArrayType	May be present when non-zero spectralPCS defined in profile header

12.2.2.2 colorantInfoStructure sub-tag member elements

12.2.2.2.1 cinfNameMbr

Tag element signature: 'name' (6e616d65h).

Permitted tag element types: utf8Type.

Element usage: required.

The colorantInfoStructure *cinfNameMbr* element contains the unique name of the colour to associate with the structure data.

12.2.2.2.2 cinfLocalizedNameMbr

Tag element signature: 'lcnm' (6c636e6dh).

Permitted tag element types: multiLocalizedUnicodeType.

Element usage: optional.

The colorantInfoStructure *cinfLocalizedNameMbr* element contains the localized versions of the name of the colour to associate with the structure data that can be used to display the name for various locales.

12.2.2.2.3 cinfPcsDataMbr

Tag element signature: 'pcs ' (70637320h).

Permitted tag element types: uInt8Number, uInt16Number, float16Number, or float32Number.

Element usage: required if PCS values defined in profile header.

The colorantInfoStructure *cinfPcsDataMbr* element shall contain a set of colorimetric PCS values to associate with the colorantInfoStructure. The number of entries and encoding of the values in the pcsData element tag shall agree with the number of entries and encoding implied by the pcsColorSpace entry in the profile header. If either the pcsColorSpace is zero or the profile is a DeviceLink profile then the number of entries shall be three and the encoding shall be assumed to be PCSLAB for the 1931 standard observer under D50 illumination.

12.2.2.2.4 cinfSpectralDataMbr

Tag element signature: 'spec' (73706563h).

Permitted tag element types: uInt8Number, uInt16Number, float16Number, float32Number or sparseMatrixArrayType.

Element usage: required if spectralPCS value is defined in profile header.

The colorantInfoStructure *cinfSpectralDataMbr* element shall contain a set of spectral values to associate with the colorantInfoStructure. If the spectralPCS entry in the profile header is a sparseMatrixReflectanceData colour space then the element type shall be a sparseMatrix. Otherwise, the number of entries in the *cinfSpectralDataMbr* element tag shall be the same as the number of entries implied by the spectralPCS entry in the profile header.

12.2.3 colorEncodingParamsStructure

12.2.3.1 General

Structure Type signature identifier: ‘cept’ (63657074h).

A colorEncodingParametersStructure is used by the colorEncodingParametersTag to define encoding parameters for the three component colour space. Publically defined element sub-tag members of a colorEncodingParametersStructure are shown in Table 131. Descriptions for each sub-tag member can be found in 12.2.3.2.

Table 131 — colorEncodingParamsStructure element sub-tags

Id	Signature	Description	Sub-tag type
ceptBluePrimaryXYZMbr	'bXYZ' (6258595ah)	nCIEXYZ values of blue colour space encoding primary	float32Number array
ceptGreenPrimaryXYZMbr	'gXYZ' (6758595ah)	nCIEXYZ values of green colour space encoding primary	float32Number array
ceptRedPrimaryXYZMbr	'rXYZ' (7258595ah)	nCIEXYZ values of red colour space encoding primary	float32Number array
ceptTransferFunctionMbr	'func' (66756e63h)	colour component transfer function	segmentedCurve Type
ceptLumaChromaMatrixMbr	'lmat' (6c6d6174h)	matrix that converts RGB values to luma-chroma values	float32Number array
ceptWhitePointLuminanceMbr	'wlum' (776c756dh)	colour space white point luminance in cd/m^2	float32Number
ceptWhitePointChromaticityMbr	'wXYZ' (7758595ah)	colour space white point chromaticity	float32Number array
ceptEncodingRangeMbr	'eRng' (65526e67h)	Describes the range of the encoding data	float32Number array
ceptBitDepthMbr	'bits' (62697473h)	bit depths for encoding	uInt8Number array
ceptImageStateMbr	'imst' (696d7374h)	image state associated with the encoding	signature
ceptImageBackgroundMbr	'ibkg' (69626b67h)	reference viewing environment image background (proximal field) in cd/m^2	float32Number
ceptViewingSurroundMbr	'srnd' (73726e64h)	reference viewing environment viewing surround in cd/m^2	float32Number
ceptAmbientIlluminanceMbr	'ailm' (61696c6dh)	reference viewing environment ambient illuminance in lux	float32Number
ceptAmbientWhitePointLuminanceMbr	'awlm' (61776c6dh)	reference viewing environment adapted white point luminance in cd/m^2	float32Number
ceptAmbientWhitePointChromaticityMbr	'awpc' (61777063h)	reference medium white point chromaticity	float32Number array
ceptViewingFlareMbr	'flar' (666c6172h)	Viewing Flare as percent of white point luminance (excluding viewing flare and veiling flare	float32Number
ceptValidRelativeLuminanceRangeMbr	'lrng' (6c726e67h)	Describes the valid relative luminance range	float32Number array
ceptMediumWhitePointLuminanceMbr	'mwpl' (6d77706ch)	reference medium white point luminance in cd/m^2	float32Number
ceptMediumWhitePointChromaticityMbr	'mwpc' (6d777063h)	reference medium adapted white point chromaticity	float32Number array
ceptMediumBlackPointLuminanceMbr	'mbpl' (6d62706ch)	reference medium black point luminance in cd/m^2	float32Number
ceptMediumBlackPointChromaticityMbr	'mbpc' (6d627063h)	reference medium black point chromaticity	float32Number array

12.2.3.2 colorEncodingParamsStructure sub-tag member elements

12.2.3.2.1 ceptBluePrimaryXYZMbr

Tag Element Signature: 'bXYZ' (6258595ah).

Permitted tag type: float32Number array.

Element usage: optional.

The colorEncodingParamsStructure *ceptBluePrimaryXYZMbr* element represents the nCIEXYZ values of blue colour space encoding primary encoded using either 2 or 3 numbers. The first value represents the x chromaticity. The second value represents the y chromaticity. The third value (if present) represents the z chromaticity. If only two numbers are present the z chromaticity is assumed to be the value of one minus the sum of the two numbers. If three values are present the sum of the three values shall be 1,0.

12.2.3.2.2 ceptGreenPrimaryXYZMbr

Tag Element Signature: 'gXYZ' (6758595ah)

Permitted tag type: float32Number array

The colorEncodingParamsStructure *ceptGreenPrimaryXYZMbr* element represents the nCIEXYZ values of green colour space encoding primary encoded using either two or three numbers. The first value represents the x chromaticity. The second value represents the y chromaticity. The third value (if present) represents the z chromaticity. If only two numbers are present the z chromaticity is assumed to be the value of one minus the sum of the two numbers. If three values are present the sum of the three values shall be 1,0.

12.2.3.2.3 ceptRedPrimaryXYZMbr

Tag Element Signature: 'rXYZ' (7258595ah).

Permitted tag types: float32Number array.

The colorEncodingParamsStructure *ceptRedPrimaryXYZMbr* element represents the nCIEXYZ values of blue colour space encoding primary encoded using either two or three numbers. The first value represents the x chromaticity. The second value represents the y chromaticity. The third value (if present) represents the z chromaticity. If only two numbers are present the z chromaticity is assumed to be the value of one minus the sum of the two numbers. If three values are present the sum of the three values shall be 1,0.

12.2.3.2.4 ceptTransferFunctionMbr

Tag Element Signature: ‘func’ (66756e63h).

Permitted tag type: segmentedCurveType.

The colorEncodingParamsStructure *ceptTransferFunctionMbr* element describes the colour component transfer function.

12.2.3.2.5 ceptLumaChromaMatrixMbr

Tag Element Signature: 'lmat' (6c6d6174h).

Permitted tag type: float32Number array.

The colorEncodingParamsStructure *ceptLumaChromaMatrixMbr* element contains nine float32Number values that define a matrix that converts RGB values to lumina-chroma values.

12.2.3.2.6 ceptWhitePointLuminanceMbr

Tag Element Signature: 'wlum' (776c756dh).

Permitted tag type: float32Number.

The colorEncodingParamsStructure *ceptWhitePointLuminanceMbr* element describes the white point luminance in cd/m^2.

12.2.3.2.7 ceptWhitePointChromaticityMbr

Tag Element Signature: 'wXYZ' (7758595ah).

Permitted tag type: float32Number array.

The colorEncodingParamsStructure *ceptWhitePointChromaticityMbr* element value describes the white point chromaticity encoded using either two or three numbers. The first value represents the x chromaticity. The second value represents the y chromaticity. The third value (if present) represents the z chromaticity. If only two numbers are present the z chromaticity is assumed to be the value of one minus the sum of the two numbers. If three values are present the sum of the three values shall be 1,0.

12.2.3.2.8 ceptEncodingRangeMbr

Tag Element Signature: 'eRng' (65526e67h).

Permitted tag types: float32Number array.

The colorEncodingParamsStructure *ceptEncodingRangeMbr* element contains two floating point numbers describing the range of the encoding data where the first number defines the minimum range value and the second number defines the maximum range value.

12.2.3.2.9 ceptBitDepthMbr

Tag Element Signature: 'bits' (62697473h).

Permitted tag types: uInt8Number array.

The colorEncodingParamsStructure *ceptBitDepthMbr* element contains one or more bit depths for encoding. A value of zero indicates floating point support.

12.2.3.2.10 ceptImageStateMbr

Tag Element Signature: 'imst' (696d7374h).

Permitted tag types: signatureType.

The colorEncodingParamsStructure *ceptImageStateMbr* element describes the image state associated with the encoding. The signature values shall be any of the valid signatures defined for the colorimetricIntentImageStateTag defined in 9.2.54 with the addition of signatures defined in Table 132.

Table 132 — imageStateData element signatures

Image state	Signature	Hexidecimal encoding
Display output referred colorimetry	‘dorc’	646f7263h

12.2.3.2.11 ceptImageBackgroundMbr

Tag Element Signature: 'ibkg' (69626b67h).

Permitted tag types: float32Number.

The colorEncodingParamsStructure *ceptImageBackgroundMbr* element describes the image background (proximal field) in cd/m^2 of the reference viewing environment.

12.2.3.2.12 ceptViewingSurroundMbr

Tag Element Signature: 'srnd' (73726e64h).

Permitted tag types: float32Number.

The colorEncodingParamsStructure *ceptViewingSurroundMbr* element describes the viewing surround in cd/m^2 of the reference viewing environment.

12.2.3.2.13 ceptAmbientIlluminanceMbr

Tag Element Signature: 'ailm' (61696c6dh).

Permitted tag types: float32Number.

The colorEncodingParamsStructure *ceptAmbientIlluminanceMbr* element describes the ambient illuminance in lux of the reference viewing environment.

12.2.3.2.14 ceptAmbientWhitePointLuminanceMbr

Tag Element Signature: 'awlm' (61776c6dh).

Permitted tag types: float32Number.

The colorEncodingParamsStructure *ceptAmbientWhitePointLuminanceMbr* element describes the adapted white point luminance in cd/m^2 of the reference viewing environment.

12.2.3.2.15 ceptAmbientWhitePointChromaticityMbr

Tag Element Signature: 'awpc' (61777063h).

Permitted tag types: float32Number array.

The colorEncodingParamsStructure *ceptAmbientWhitePointChromaticityMbr* element describes the white point chromaticity of the reference medium, encoded using either two or three numbers. The first value represents the x chromaticity. The second value represents the y chromaticity. The third value (if present) represents the z chromaticity. If only two numbers are present the z chromaticity is assumed to be the value of one minus the sum of the two numbers. If three values are present the sum of the three values shall be 1,0.

12.2.3.2.16 ceptMediumWhitePointLuminanceMbr

Tag Element Signature: 'mwpl' (6d77706ch).

Permitted tag types: float32Number.

The colorEncodingParamsStructure *ceptMediumWhitePointLuminanceMbr* element describes the white point luminance in cd/m^2 of the reference medium.

12.2.3.2.17 ceptMediumWhitePointChromaticityMbr

Tag Element Signature: 'mwpc' (6d777063h).

Permitted tag types: float32Number array.

The colorEncodingParamsStructure *ceptMediumWhitePointChromaticityMbr* element describes the adapted white point chromaticity of the reference medium, encoded using either two or three numbers. The first value represents the x chromaticity. The second value represents the y chromaticity. The third value (if present) represents the z chromaticity. If only two numbers are present the z chromaticity is assumed to be the value of one minus the sum of the two numbers. If three values are present the sum of the three values shall be 1,0.

12.2.3.2.18 ceptMediumBlackPointLuminanceMbr

Tag Element Signature: 'mbpl' (6d62706ch).

Permitted tag types: float32Number.

The colorEncodingParamsStructure *ceptMediumBlackPointLuminanceMbr* element describes the reference medium's black point luminance in cd/m^2.

12.2.3.2.19 ceptMediumBlackPointChromaticityMbr

Tag Element Signature: 'mbpc' (6d627063h).

Permitted tag types: float32Number array.

The colorEncodingParamsStructure *ceptMediumBlackPointChromaticityMbr* element describes the black point chromaticity of the reference medium, encoded using either two or three numbers. The first value represents the x chromaticity. The second value represents the y chromaticity. The third value (if present) represents the z chromaticity. If only two numbers are present the z chromaticity is assumed to be the value of one minus the sum of the two numbers. If three values are present the sum of the three values shall be 1,0.

12.2.4 measurementInfoStructure

12.2.4.1 General

Structure Type Identifier: 'meas' (6d656173h).

The measurementInfoStructure is used by the measurementInfoTag (see 9.2.86) and the measurementInputInfoTag (see 9.2.87) which define aspects of the measurement data in the PCS and input side of abstract profiles, respectively. Publically defined element sub-tag members of a measurementInfoStructure are shown in Table 133. Descriptions for each sub-tag member can be found in 12.2.4.2.

Table 133 — measurementInfoStructure element tags

Id	Signature	Description	Sub-tag type	Use
measBackingMbr	'mbak' (6d62616bh)	Measurement backing (see 12.2.4.2.1)	uInt32Number	May be present
measFlareMbr	'mflr' (6d666c72h)	Measurement flare (see 12.2.4.2.2)	float32Number	May be present
measGeometryMbr	'mgeo' (6d67656fh)	Measurement geometry (see 12.2.4.2.3)	uInt32Number	May be present
measIlluminantMbr	'mill' (6d696c6ch)	Measurement illuminant spectral power distribution (SPD) (see 12.2.4.2.4)	array of float16Number, array of float32Number	May be present
measIlluminantRangeMbr	'miwr' (6d697772h)	Spectral range of measurement illuminant spectral power distribution (SPD) functionally based spectral BRDF (see 12.2.4.2.5)	spectralRange	May be present
measModeMbr	'mmod' (6d6d6f64h)	Measurement mode (see 12.2.4.2.6)	uInt32Number	May be present

12.2.4.2 measurementInfoStructure sub-tag member elements

12.2.4.2.1 measBackingMbr

Tag signature: ‘mbak’ (6d62616bh).

Permitted tag types: uInt32Number.

The measurementInfoStructure *measBackingMbr* element defines the backing used for reflectance-based measurements. If the element is not present the backing shall be assumed to be white. The encoding for the measurement backing is shown in Table 134.

Table 134 — measBackingMbr encoding

Geometry	Hexidecimal encoding
Undefined	00000000h
White backing	00000001h
Black backing	00000002h
Media (self) backing	00000003h

12.2.4.2.2 measFlareMbr

Tag signature: ‘mflr’ (6d666c72h).

Permitted tag types: float32Number.

The measurementInfoStructure *measFlareMbr* element defines the level of flare involved to make a measurement. If the element is not present, the flare shall be assumed to be zero. The encoded value for the measureFlare member variable shall range from 0,0 to 1,0.

12.2.4.2.3 measGeometryMbr

Tag signature: ‘mgeo’ (6d67656fh).

Permitted tag types: uInt32Number.

The measurementInfoStructure measGeometryMbr element defines the geometry used to make a measurement. If the element is not present the geometry shall be assumed to be 0°:45°. The encoding for the measurement geometry is shown in Table 135.

Table 135 — measGeometryMbr encoding

Geometry	Hexidecimal encoding
Unknown	00000000h
0°:45° or 45°:0°	00000001h
0°:d or d:0°	00000002h

12.2.4.2.4 measIlluminantMbr

Tag signature: ‘mill’ (6d696c6ch).

Permitted tag types: uInt32Number.

The measurementInfoStructure *measIlluminantMbr* element defines the actual spectral power distribution (SPD) of the illuminant used to make measurements. It contains an array of values that defines the spectral output for each of the wavelengths defined by the measureIlluminantRange sub-tag. The measureIlluminant element shall contain the same number of elements as are defined by the

steps value of the measureIlluminantRange sub-tag. If the element is not present the exact SPD of the measurementilluminant shall be assumed to be unknown.

12.2.4.2.5 measIlluminantRangeMbr

Tag signature: ‘miwr’ (6d697772h).

Permitted tag types: spectralRange.

The measurementInfoStructure *measIlluminantRangeMbr* element defines the starting wavelength, ending wavelength and number of steps for the actual spectral power distribution (SPD) of the illuminant (defined in a measurementIlluminant sub-tag element) used to make measurements. The measureIlluminantRange tag shall be present when a measureIlluminant sub-tag is present. The value of wavelength steps in the measureIlluminantRange shall be the same as the number of elements in the measureIlluminant sub-tag.

12.2.4.2.6 measModeMbr

Tag signature: ‘mmod’ (6d6d6f64h).

Permitted tag types: uInt32Number.

The measurementInfoStructure *measModeMbr* element defines the measurement mode used to make a measurement (as defined by ISO 13655-2009). If the element is not present the measurement mode shall be assumed to be M1. The encoding for the measurement geometry is shown in Table 136.

Table 136 — measModeMbr Encoding

Measurement Mode	Hexidecimal encoding
Undefined	00000000h
M0 – Default (tungsten)	00000001h
M1 – D50	00000002h
M2 – UV-Cut	00000003h
M3 – Polarizing filter	00000004h

12.2.5 namedColorStructure

12.2.5.1 General

Structure Type Identifier: 'nmcl' (6e6d636ch).

The namedColorStructure is used by the namedColorTag (see 9.2.99), which contains a namedColorArray (see 13.2.1) defined as a tagArrayType with a single tintZeroStructure followed by additional namedColorStructure elements. Publically defined element sub-tag members of a namedColorStructure are shown in Table 137. Descriptions for each sub-tag member can be found in 12.2.5.2.

Table 137 — namedColorStructure element sub-tags

Id	Signature	Description	Sub-tag type	Use
nmclBrdfColorimetricMbr	‘bcol’ (62636f6ch)	Functionally based colorimetric BRDF (see 12.2.5.2.1)	multiProcessElementType	May be present
nmclBrdfColorimetricParamsMbr	‘bcpr’ (62637072h)	Colorimetric parametric BRDF specification (see 12.2.5.2.2)	tagStructType of type brdfTransformStructure	May be present
nmclBrdfSpectralMbr	‘bspc’ (62737063h)	Functionally based spectral BRDF (see 12.2.5.2.3)	multiProcessElementType	May be present
nmclBrdfSpectralParamsMbr	‘bspr’ (62737072h)	Spectral parametric BRDF specification (see 12.2.5.2.4)	tagStructType of type brdfTransformStructure	May be present
nmclDeviceDataMbr	'dev ' (64657620h)	Device values used to reproduce colour (see 12.2.5.2.7)	uInt8Number uInt16Number float16Number float32Number	May be present
nmclLocalizedNamesMbr	'lcnm' (6c636e6dh)	Localized names of named colour (see 12.2.5.2.6)	multiLocalizedUnicodeType	May be present
nmclNameMbr	'name' (6e616d65h)	Name of named colour (see 12.2.5.2.5)	utf8Type	May be present
nmclNormalMapMbr	‘nmap’ (6e6d6170h)	Surface normal map (see 12.2.5.2.8)	embeddedNormalImageType	May be present
nmclPcsDataMbr	'pcs ' (70637320h)	PCS values associated with colour (see 12.2.5.2.9)	uInt8Number uInt16Number float16Number float32Number	Shall be present if PCS field is non-zero in profile header
nmclSpectralDataMbr	'spec' (73706563h)	Spectral values associated with colour (see 12.2.5.2.10)	uInt8Number uInt16Number float16Number float32Number sparseMatrixArrayType	Shall be present if spectralPCS field is non-zero in profile header

Table 137 *(continued)*

nmclSpectralOverBlackDataMbr	'spcb' (73706362h)	Spectral values associated with colour overprinted on black medium (see 12.2.5.2.11)	uInt8Number uInt16Number float16Number float32Number sparseMatrixArrayType	May be present – if present spectralPCS field shall be non-zero in profile header
nmclSpectralOverGrayDataMbr	'spcg' (73706367h)	Spectral values associated with colour overprinted on gray medium (see 12.2.5.2.12)	uInt8Number uInt16Number float16Number float32Number sparseMatrixArrayType	May be present – if present spectralPCS field shall be non-zero in profile header
nmclTintValuesMbr	'tint' (74696e74h)	Tint values of named colour (see 12.2.5.2.13)	uInt8Number uInt16Number float32Number	May be present

12.2.5.2 namedColorStructure sub-tag member elements

12.2.5.2.1 nmclBrdfColorimetricMbr

Tag signature: 'bcol' (62636f6ch).

Permitted tag types: multiProcessElementType.

The namedColorStructure *nmclBrdfColorimetricMbr* element defines a transform in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle and tint to the colorimetric-based PCS specified by the PCS field in the profile header.

The number of input channels to the multiProcessElementsType-based tag shall be five. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 138.

Table 138 — BRDF device channel encoding

Input channel index	Channel identification	Encoding type
0	Viewing azimuth angle Φr	azimuthNumber
1	Viewing zenith angle θr	zenithNumber
2	Lighting azimuth angle Φi	azimuthNumber
3	Lighting zenith angle θi	zenithNumber
4	Tint	

The domain of Tint values input to this multiProcessElementType based tag shall include the tint value of zero. There shall therefore be no reference or use of a *tnt0BrdfColorimetricMbr* subtag [i.e a sub-tag with a tag signature of 'bcol' (62636f6ch)] in the tintZeroStructure.

The output channels are defined by the encoding implied by the PCS field in the profile header.

12.2.5.2.2 nmclColorimetricParametersMbr

Tag signature: 'bcpr' (62637072h).

Permitted tag types: tagStructType of type brdfTransformStructure.

The namedColorStructure *nmclColorimetricParametersMbr* element defines colorimetric BRDF parameters. Specifically, it specifies a transform from tint to colorimetric BRDF parameters. See 12.2.1 for a description of the brdfTransformStructure. A monochrome brdfTransformStructure shall use the *nmclPcsDataMbr* sub-tag as the source of PCS colour. A monochrome brdfTransformStructure shall not be encoded in the profile if the *nmclPcsDataMbr* sub-tag is not present.

12.2.5.2.3 nmclBrdfSpectralMbr

Tag signature: 'bspc' (62737063h).

Permitted tag types: multiProcessElementType.

The namedColorStructure *nmclBrdfSpectralMbr* element defines a transform in relation to viewing and lighting angles. Specifically, it describes the colour transform from viewing angle, lighting angle, and tint to the spectrally-based PCS specified by the spectralPCS field in the profile header.

The number of input channels to the multiProcessElementsType-based tag shall be five. The order and encoding of the BRDF and device channels provided to the multiProcessElementType are shown in Table 138.

The domain of Tint values input to this multiProcessElementType based tag shall include the tint value of zero. There shall therefore be no reference or use of a *tnt0BrdfSpectralMbr* sub-tag (i.e a sub-tag with a tag signature of 'bcpr' (62637072h)) in the tintZeroStructure.

The output channels are defined by the encoding implied by the spectralPCS field in the profile header.

12.2.5.2.4 nmclBrdfSpectralParamsMbr

Tag signature: 'bspr' (62737072h).

Permitted tag types: tagStructType of type brdfTransformStructure.

The namedColorStructure *nmclBrdfSpectralMbr* element defines spectral BRDF parameters. Specifically, it specifies a transform from tint to spectral BRDF parameters. See 12.2.1 for a description of the brdfTransformStructure. A monochrome brdfTransformStructure shall use the namedColorStructure *nmclSpectralDataMbr* sub-tag as the source of PCS colour. A monochrome brdfTransformStructure shall not be encoded in the profile if the namedColorStructure *nmclSpectralDataMbr* sub-tag is not present.

12.2.5.2.5 nmclNameMbr

Tag element signature: 'name' (6e616d65h).

Permitted tag element types: utf8Type.

Element usage: required.

The namedColorStructure *nmclNameMbr* element contains the unique name of the colour to associate with the structure data.

12.2.5.2.6 nmclLocalizedNameMbr

Tag element signature: 'lcnm' (6c636e6dh).

Permitted tag element types: multiLocalizedUnicodeType.

Element usage: optional.

The namedColorStructure *nmclLocalizedNameMbr* element contains the localized versions of the name of the colour to associate with the structure data that can be used to display the name for various locales.

12.2.5.2.7 nmclDeviceDataMbr

Tag element signature: 'dev ' (64657620h).

Permitted tag element types: uInt8Number or uInt16Number, float16Numbmer, or float32Number.

Element usage: optional.

The namedColorStructure *nmclDeviceDataMbr* element shall contain a tint array of device values for each non-zero tint used to produce the named colour. The number of entries in the deviceData element tag shall agree with the number of entries implied by the dataColorSpace entry in the profile header multiplied by the number of tints being defined. If no namedColorStructure *nmclTintValuesMbr* element member is present then tints are assumed to be equally spaced. Otherwise, tint spacing values for each *nmclDeviceDataMbr* vector are provided in the namedColorStructure *nmclTintValuesMbr* element. Intermediate device tint values can be determined using linear interpolation. The value for the deviceData zero tint is defined in the associated tintZeroStructure of first element of the containing tagArrayType. Linear interpolation is assumed between the zero tint (defined in the tintZeroStructure tnt0*DeviceDataMbr* sub-tag in 12.2.7.2.1) and the first tint vector in the namedColorStructure nmclr*DeviceDataMbr* sub-tag.

12.2.5.2.8 nmclNormalMapMbr

Tag element signature: 'nmap ' (6e6d6170h).

Permitted tag element type: embeddedNormalImageType.

Element usage: optional.

The namedColorStructure *nmclNormalMapMbr* element provides a normal map image that can be associated with the named colour.

12.2.5.2.9 nmclPcsDataMbr

Tag element signature: 'pcs ' (70637320h).

Permitted tag element types: uInt8Number, uInt16Number, float16Number, or float32Number.

Element usage: required if PCS values defined in profile header.

The namedColorStructure *nmclPcsDataMbr* element shall contain a tint array of pcsData values for each non-zero tint used to produce the named colour. The number of entries in the pcsData element tag shall agree with the number of entries implied by the PCS entry in the profile header multiplied by the number of tints being defined. If no namedColorStructure *nmclrTintValuesMbr* element is present then tints are assumed to be equally spaced. Otherwise, tint spacing values for each *nmclPcsDataMbr* vector are provided in the namedColorStructure *nmclrTintValuesMbr* (See 12.2.5.2.13) element. Intermediate pcsData tint values can be determined using linear interpolation. The value for the pcsData zero tint is defined in the associated tintZeroStructure of the containing tagArrayType. Linear interpolation is assumed between the zero tint (defined in the tintZeroStructure tnt0P*csDataMbr* sub-tag in 12.2.7.2.2) and the first tint vector in the namedColorStructure nmcl*PcsDataMbr* sub-tag.

12.2.5.2.10 nmclSpectralDataMbr

Tag element signature: 'spec' (73706563h).

Permitted tag element types: uInt8Number, uInt16Number, float16Number, float32Number, or sparseMatrixArrayType.

Element usage: required if spectralPCS value is non-zero in profile header.

The namedColorStructure *nmclSpectralDataMbr* element shall contain a tint array of spectralData values for each non-zero tint used to produce the named colour. If the spectralPCS entry in the profile header

is a sparseMatrixReflectanceData colour space then the element type shall be sparseMatrixArrayType array. Otherwise, the number of entries in the *spectralData* element tag shall be the same as the number of entries implied by the spectralPCS entry in the profile header multiplied by the number of tints being defined. If no namedColorStructure *nmclTintValuesMbr* element member is present then tints shall be assumed to be equally spaced. Otherwise, tint spacing values for each *nmclSpectralDataMbr* vector shall be provided in the namedColorStructure *nmclTintValuesMbr* (See 12.2.5.2.13) element. Intermediate spectralData tint values can be determined using linear interpolation. The value for the spectralData zero tint is defined in the associated tintZeroStructure of the containing tagArrayType. Linear interpolation is assumed between the zero tint (defined in the tintZeroStructure *tnt0SpectralDataMbr* sub-tag in 12.2.7.2.3) and the first tint vector in the namedColorStructure nmcl*SpectralDataMbr* sub-tag.

12.2.5.2.11 nmclSpectralOverBlackDataMbr

Tag element signature: 'spcb' (73706362h).

Permitted tag element types: uInt8Number, uInt16Number, float16Number, float32Number, or sparseMatrixArrayType.

Element usage: option – when used, spectralPCS value shall be non-zero in profile header.

The namedColorStructure *nmclSpectralOverBlackDataMbr* element shall contain a tint array of spectralOverBlackData values for each non-zero tint used to produce the named colour over a black medium. If the spectralPCS entry in the profile header is a sparseMatrixReflectanceData colour space then the element type shall be sparseMatrixArrayType array. Otherwise, the number of entries in the *nmclSpectralOverBlackDataMbr* element tag shall be the same as the number of entries implied by the spectralPCS entry in the profile header multiplied by the number of tints being defined. If no namedColorStructure *nmclTintValuesMbr* element member is present, then tints are assumed to be equally spaced. Otherwise, tint spacing values for each *nmclSpectralOverBlackDataMbr* vector are provided in the namedColorStructure *nmclrTintValusMbr* (see 12.2.5.2.13) element. Intermediate spectralOverBlackData tint values can be determined using linear interpolation. The value for the spectralOverBlackData zero tint is defined in the associated tintZeroStructure of the containing tagArrayType. Linear interpolation is assumed between the zero tint (defined in the tintZeroStructure tnt0*SpectralOverBlackDataMbr* sub-tag in 12.2.7.2.4) and the first tint vector in the namedColorStructure *nmclSpectralOverBlackDataMbr* Sub-tag.

12.2.5.2.12 nmclSpectralOverGrayDataMbr

Tag element signature: 'spcg' (73706367h).

Permitted tag element types: uInt8Number, uInt16Number, float16Number, float32Number, or sparseMatrixArrayType.

Element usage: option – when used, spectralPCS value shall be non-zero in profile header.

The namedColorStructure *nmclSpectralOverGrayDataMbr* element shall contain a tint array of spectralOverGrayData values for each non-zero tint used to produce the named colour over a gray medium. If the spectralPCS entry in the profile header is a sparseMatrixReflectanceData colour space then the element type shall be sparseMatrixArrayType array. Otherwise, the number of entries in the *nmclSpectralOverGrayDataMbr* element tag shall be the same as the number of entries implied by the spectralPCS entry in the profile header multiplied by the number of tints being defined. If no namedColorStructure nmcl*TintValuesMbr* element is present, then tints are assumed to be equally spaced. Otherwise, tint spacing values for each spectralOverBlackData vector are provided in the namedColorStructure *nmclTintValuesMbr* (see 12.2.5.2.13) element. Intermediate spectralOverGrayData tint values can be determined using linear interpolation. The value for the spectralOverGrayData zero tint is defined in the associated tintZeroStructure of the containing tagArrayType. Linear interpolation is assumed between the zero tint (defined in the tintZeroStructure *tnt0SpectralOverGrayDataMbr* sub-tag in 12.2.7.2.5) and the first tint vector in the namedColorStructure *nmclSpectralOverGrayDataMbr* sub-tag.

12.2.5.2.13 nmclTintValuesMbr

Tag element signature: 'tint' (74696e74h).

Permitted tag element types: uInt8Number or uInt16Number or float32Number.

Element usage: optional.

The namedColorStructure *nmclTintValuesMbr* element can contain an array of tint values for each tint defined for a named colour. Each element in the array defines the associated tint value for each position in the vector arrays of sub-tag elements in a namedColorStructure.

The number of values this array shall correspond to the number of vector entries in the *nmclDeviceDataMbr* (12.2.5.2.7), *nmclPcsDataMbr* (12.2.5.2.9), *nmclSpectralDataMbr* (12.2.5.2.10) *nmclSpectralOverBlackDataMbr* (12.2.5.2.11), or *nmclSpectralOverGrayDataMbr* (12.2.5.2.12) sub-tags (if they exist). If the namedColorStructure *nmclTintValuesMbr* element is not present for a named colour then equal spacing of tint values is assumed for the *nmclDeviceDataMbr*, *nmclPcsDataMbr*, *nmclSpectralDataMbr*, *nmclSpectralOverBlackDataMbr*, and *nmclSpectralOverGrayDataMbr* sub-tags.

The *nmclTintValuesMbr* array shall be a monotonically increasing array that represents tint values for the first non-zero tint through a maximum amount of the named colour. The named colour tint can ranging from 1 to 255 for uInt8Number encoding, 1 to 65 535 for uInt16Number encoding or greater than 0,0 to 1,0 for float16Number and float32Number encoding. The zero tint value shall be defined in the associated tintZeroStructure of the containing tagArrayType.

12.2.6 profileInfoStructure

12.2.6.1 General

Structure Type Identifier: 'pinf' (70696e66h).

The profileInfoStructure is used by the profileSequenceInfoTag (see 9.2.102) which contains a tagArrayType of profileInfoStructure elements as a profileInfoArray (see 13.2.2). Each entry in the array contains information about a profile used in a sequence of profiles. This provides a description of the profile sequence from source to destination, typically used with the DeviceLink profile. Each element is optional and each has an assumed value defined for each sub-tag if not present. Publically-defined element sub-tag members of a profileInfoStructure are shown in Table 139. Descriptions for each sub-tag member can be found in 12.2.6.2.

Table 139 — profileInfoStructure element sub-tags

Id	Signature	Description	Sub-tag type	Use
pinfAttributesMbr	'attr' (61747472h)	Device attributes (see 12.2.6.2.1)	uInt64Number	May be present
pinfProfileDescMbr	'pdsc' (70647363h)	Profile description (see 12.2.6.2.2)	multiLocalizedUnicodeType	May be present
pinfProfileIDMbr	'pid ' (70696420h)	Profile ID (see 12.2.6.2.3)	uInt8Number array	May be present
pinfManufacturerDescMbr	'dmnd ' (646d6e64h)	Device manufacturer description (see 12.2.6.2.4)	multiLocalizedUnicodeType	May be present
pinfManufacturerSigMbr	'dmns ' (646d6e73h)	Device manufacturer signature (see 12.2.6.2.5)	signatureType	May be present
pinfModelDescMbr	'dmdd ' (646d6464h)	Device model description (see 12.2.6.2.6)	multiLocalizedUnicodeType	May be present

Table 139 *(continued)*

Id	Signature	Description	Sub-tag type	Use
pinfModelSigMbr	'mod ' (6d6f6420h)	Device model signature (see 12.2.6.2.7)	signatureType	May be present
pinfRenderTransformMbr	'rtrn' (7274726eh)	Rendering intent transform ID (see 12.2.6.2.8)	uInt32Number	May be present
pinfTechnologyMbr	'tech' (74656368h)	Device technology (see 12.2.6.2.9)	signatureType	May be present

12.2.6.2 profileInfoStructure sub-tag member elements

12.2.6.2.1 pinfAttributesMbr

Tag element signature: 'attr' (61747472h).

Permitted tag element types: uInt64Number.

Element usage: optional.

The profileInfoStructure *pinfAttributesMbr* element contains information from the device attributes from the header of the corresponding profile. Assumed to be zero if element is not present.

12.2.6.2.2 pinfProfileDescMbr

Tag element signature: 'pdsc' (70647363h).

Permitted tag element types: multiLocalizedUnicodeType.

Element usage: optional.

The profileInfoStructure *pinfProfileDescMbr* element contains the contents of the profileDescriptionTag from the corresponding profile. If element not present then an empty description is assumed.

12.2.6.2.3 pinfProfileIDMbr

Tag element signature: 'pid' (70696420h).

Permitted tag element types: uInt8Number array of 16 bytes.

Element usage: optional.

The profileInfoStructure *pinfProfileIDMbr* element contains the Profile ID from the header of the corresponding profile. If the corresponding profile contains a Profile ID in the Profile Header, it shall be used in the Profile Identifier structure. If the profile does not contain a Profile ID in the Profile Header, then either this element can be excluded, or an all-zero Profile ID or a computed Profile ID shall be used. The ProfileID is assumed to be zero filled if the element is not present.

12.2.6.2.4 pinfManufacturerDescMbr

Tag element signature: 'dmnd' (646d6e64h).

Permitted tag element types: multiLocalizedUnicodeType.

Element usage: optional.

The profileInfoStructure *pinfManufacturerDescMbr* element contains contents of the deviceMfgDescTag from the corresponding profile. If the element not present then an empty description is assumed.

12.2.6.2.5 pinfManufacturerSigMbr

Tag element signature: 'dmns' (646d6e73h).

Permitted tag element types: signatureType.

Element usage: optional.

The profileInfoStructure *pinfManufacturerSigMbr* element contains information from the device model from the header of the corresponding profile. Assumed to be zero if the element is not present.

12.2.6.2.6 pinfModelDescMbr

Tag element signature: 'dmdd' (646d6464h).

Permitted tag element types: multiLocalizedUnicodeType.

Element usage: optional.

The profileInfoStructure *pinfoModelDescMbr* element contains the contents of the deviceModelDescTag from the corresponding profile. If the element not present then an empty description shall be assumed.

12.2.6.2.7 pinfModelSigMbr

Tag element signature: 'mod' (6d6f6420h).

Permitted tag element types: signatureType.

Element usage: optional.

The profileInfoStructure *pinfModelSigMbr* element contains information from the device model from the header of the corresponding profile. Assumed to be zero if the element is not present.

12.2.6.2.8 pinfRenderingTransformMbr

Tag element signature: ‘rtrn’ (7274726eh).

Permitted tag element types: uInt32Number.

Element usage: optional.

The profileInfoStructure *pinfRenderingTransformMbr* element defines the rendering intent transform from the corresponding profile that was used to establish the combined transform. The value of the uInt32Number shall be one of the values in Table 20.

If this sub-tag is not present then the rendering intent transform shall be defined by the rendering intent field in the profile header (see 7.2.17).

12.2.6.2.9 pinfTechnologyMbr

Tag element signature: 'tech' (74656368h).

Permitted tag element types: signatureType.

Element usage: optional.

The profileInfoStructure *pinfTechnologyMbr* element contains contents of the technologyTag from the corresponding profile. If not present, then a zero technology signature shall be assumed.

12.2.7 tintZeroStructure

12.2.7.1 General

Structure Type Identifier: 'tnt0' (746e7430h).

The tintZeroStructure is used by the namedColorTag (see 9.2.57), which contains a namedColorArray (see 13.2.1) defined as a tagArrayType with a single tintZeroStucture followed by additional namedColorStructure elements. Publically defined element sub-tag members of a tintZeroStructure are shown in Table 140. Descriptions for each sub-tag member can be found in 12.2.7.2.

Table 140 — tintZeroStructure element sub-tags

Id	Signature	Description	Sub-tag type	Use
tnt0DeviceDataMbr	'dev ' (64657620h)	Device values used to reproduce zero tint (see 12.2.7.2.1)	uInt8Number uInt16Number float16Number float32Number	May be present
tnt0PcsDataMbr	'pcs ' (70637320h)	PCS values associated with zero tint (see 12.2.7.2.2)	uInt8Number uInt16Number float16Number float32Number	Shall be present if PCS field is non-zero in profile header
tnt0SpectralDataMbr	'spec' (73706563h)	Spectral values associated with zero tint (see 12.2.7.2.3)	uInt8Number uInt16Number float16Number float32Number sparseMatrixArrayType	Shall be present if spectralPCS field is non-zero in profile header
tnt0SpectralOverBlackDataMbr	'spcb' (73706362h)	Spectral values associated with colour overprinted on black medium (see 12.2.7.2.4)	uInt8Number uInt16Number float16Number float32Number sparseMatrixArrayType	May be present – if present, spectralPCS field shall be non-zero in profile header
tnt0SpectralOverGrayDataMbr	'spcg' (73706367h)	Spectral values associated with colour overprinted on gray medium (see 12.2.7.2.5)	uInt8Number uInt16Number float16Number float32Number sparseMatrixArrayType	May be present – if present, spectralPCS field shall be non-zero in profile header

12.2.7.2 tintZeroStructure sub-tag member elements

12.2.7.2.1 tnt0DeviceDataMbr

Tag element signature: 'dev' (64657620h).

Permitted tag element types: uInt8Number or uInt16Number, float16Number or float32Number.

Element usage: optional.

The tintZeroStructure *tnt0DeviceDataMbr* element shall contain an array of deviceData sample values used to define the zero tint used to estimate named colour tints. The number of entries in the

tnt0DeviceDataMbr element tag shall agree with the number of entries implied by the dataColorSpace entry in the profile header. Near zero tint deviceData values are determined using the tnt0*DeviceDataMbr* sub-tag element in the tintZeroStructure in combination with the nmcl*DeviceDataMbr* sub-tag element in namedColorStructure (see 12.2.5.2.7).

12.2.7.2.2 tnt0PcsDataMbr

Tag element signature: 'pcs ' (70637320h).

Permitted tag element types: uInt8Number or uInt16Number, float16Number, or float32Number.

Element usage: required if PCS values defined in profile header.

The tintZeroStructure *tnt0PcsDataMbr* element shall contain an array of pcsData sample values used to define the zero tint used to estimate named colour tints. The number of entries in the *tnt0PcsDataMbr* element tag shall agree with the number of entries implied by the PCS entry in the profile header. Near zero tint pcsData values are determined using the *tnt0PcsDataMbr* sub-tag element in the tintZeroStructure in combination with the *nmclPcsDataMbr* sub-tag element in a namedColorStructure (See 12.2.5.2.9).

12.2.7.2.3 tnt0SpectralDataMbr

Tag element signature: 'spec' (73706563h).

Permitted tag element types: uInt8Number or uInt16Number or float16number or float32Number or sparseMatrixArrayType.

Element usage: required if spectralPCS value is non-zero in profile header.

The tintZeroStructure *tnt0SpectralDataMbr* element shall contain the array of spectralData sample values used to define the zero tint used to estimate named colour tints. If the spectralPCS entry in the profile header is a sparseMatrixReflectanceData colour space then the element type shall be sparseMatrixArrayType. Otherwise, the number of entries in the *tnt0SpectralDataMbr* element tag shall be the same as the number of entries implied by the spectralPCS entry in the profile header. Near zero tint spectralData values are determined using the *tnt0SpectralDataMbr* sub-tag element in the tintZeroStructure in combination with the nmcl*SpectralDataMbr* sub-tag element in a namedColorStructure (see 12.2.5.2.10).

12.2.7.2.4 *tnt0SpectralOverBlackDataMbr*

Tag element signature: 'spcb' (73706362h).

Permitted tag element types: uInt8Number or uInt16Number or float16number or float32Number or sparseMatrixArrayType.

Element usage: Optional – required when *nmclSpectralOverBlackDataMbr* sub-tags are used in other namedColorStructures in the namedColorArray.

The tintZeroStructure *tnt0SpectralOverBlackDataMbr* element shall contain the array of spectralOverBlackData sample values used to define the zero tint used to estimate named colour tints over a black medium. When this tag is used the spectralPCS value shall be defined in the profile header. If the spectralPCS entry in the profile header is a sparseMatrixReflectanceData colour space then the element type shall be sparseMatrixArrayType. Otherwise, the number of entries in the *tnt0SpectralOverBlackDataMbr* element tag shall agree with the number of entries implied by the spectralPCS entry in the profile header. Near zero tint spectralOverBlackData values are determined using the *tnt0SpectralOverBlackDataMbr* sub-tag element in the tintZeroStructure in combination with the *nmclSpectralOverBlackDataMbr* sub-tag element in a namedColorStructure (See 12.2.5.2.11).

12.2.7.2.5 *tnt0SpectralOverGrayDataMbr*

Tag element signature: 'spcg' (73706367h).

Permitted tag element types: uInt8Number or uInt16Number or float16number or float32Number or sparseMatrixArrayType.

Element usage: Optional – required when nm*SpectralOverGrayDataMbr* sub-tags are used in other namedColorStructures in the namedColorArray.

The tintZeroStructure *tnt0SpectralOverGrayDataMbr* element shall contain the array of *spectralOverGrayData* sample values used to define the zero tint used to estimate named colour tints over a gray medium. When this tag is used the spectralPCS value shall be defined in the profile header. If the spectralPCS entry in the profile header is a sparseMatrixReflectanceData colour space then the element type shall be sparseMatrix array. Otherwise, the number of entries in the *tnt0SpectralOverGrayDataMbr* element tag shall agree with the number of entries implied by the spectralPCS entry in the profile header. Near zero tint *spectralOverGrayData* values are determined using the *tnt0SpectralOverGrayDataMbr* sub-tag element in the tintZeroStructure in combination with the *nmclSpectralOverGrayDataMbr* sub-tag element in a namedColorStructure (see 12.2.5.2.12).

13 Tag Array Type definitions

13.1 General

The tagArrayType provides the means of encoding multiple tag elements into a single indexed array of contained sub-tag elements. Each tagArrayType has an Array Type Identifier that shall be used to identify the possible sub-tag elements and the purposes for each sub-tag element in the array.

The public tagArrayType array identifier types defined by the ICC are listed in 12.2 in alphabetical order.

13.2 Tag array indentifier type listing

13.2.1 namedColorArray

Array Type Identifier: 'ncol' (6e636f6ch).

A namedColorArray shall contain an array of tintZeroStructure and namedColorStructure elements. Information related to a named colour can include PCS and as optional device representation for a list of named colours. The first element in the array shall be a tintZeroStructure which corresponds to colour values when a zero tint of any named colour is used. See 12.2.7 for a complete description of contents and usage of a tintZeroStructure. Succeeding elements shall be defined as a namedColorStructure. See 12.2.5 for a complete description of contents and usage of a namedColorStructure. The namedColorArray is utilized by the namedColorTag (see 9.2.99).

13.2.2 profileInfoArray

Array Type Identifier: 'pinf' (70696e66h).

A profileInfoArray shall contain an array of profileInfoStructure structures that each contain information about a single profile. The successive elements of the array provide a description of the successive profiles in a sequence from source to destination. A profileInfoArray is utilized by the profileSequenceInformationTag (see 9.2.102) which is is typically used with the DeviceLink profile. See 12.2.6 for a complete description of contents and usage of a profileInfoStructure.

Annex A
(informative)

Elemental calculations and inter-PCS operations

A.1 Elemental calculations

A.1.1 General overview

The inter-PCS operations are described in A.2. These operations use the elemental calculations defined in A.1 to convert between the various supported PCS encodings. Nearly all of the operations can be represented as a linear matrix operation resulting in the ability to concatenate the calculations used in performing inter-PCS operations.

The process of converting spectral reflectance/transmission to tristimulus (colorimetric) values can be subdivided into transform steps from incident light to reflected/transmitted light to observer/sensor capture of reflected/transmitted light. This subdivision allows for both resampling of spectral data to match illuminant and observer spectralRange requirements and providing building blocks for constructing conversions between the various types of spectral PCS data.

A.1.2 Spectral resampling

A spectralRange defines the start wavelength, end wavelength, and total number of equally-spaced steps. Spectral reflectance, transmission, emission, radiance, as well as illuminant, and observer are specified using a spectral Range. The spectral Range for a fluorescent PCS defines both a spectralInputRange as well as a spectralOutputRange.

Spectral operations on spectral vector data require that the spectralRanges of the data being operated on are the same. When they are not the same, resampling is performed. Spectral resampling is performed using linear interpolation of input wavelengths for each resulting wavelength. When a wavelength is extended on either end of the range, the value for the extension is defined by the value for the wavelength closest to the extended wavelength.

This approach of spectral resampling results allows resampling to be performed as a matrix operation.

The example in Formula (A.1) (greatly reduced for example purposes only) shows resampling of 400 nm to 700 nm with 4 intervals to 350 nm to 750 nm with 9 intervals:

$$\begin{bmatrix} R_{350} \\ R_{400} \\ R_{450} \\ R_{500} \\ R_{550} \\ R_{600} \\ R_{650} \\ R_{700} \\ R_{750} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0{,}5 & 0{,}5 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0{,}5 & 0{,}5 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0{,}5 & 0{,}5 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{400} \\ R_{500} \\ R_{600} \\ R_{700} \end{bmatrix} \quad \text{(A.1)}$$

where R_λ is the spectral value (reflectance) at wavelength λ.

This method of spectral resampling can be applied to reflectance, transmission, emission, radiance, or irradiance vectors.

A.1.3 Reflectance/transmission to radiance/emission

The conversion of reflectance/transmission to radiance/emission represents incident light from an illuminant being reflected or transmitted by the object represented by the reflectance/transmission data. The conversion of a reflectance/transmission vector to a radiance/emission vector can be represented as the scalar product of a reflectance/transmission vector and a vector for the illuminant.

This requires that the vectors for the illuminant and reflectance/transmission share the same scalarRange.

This conversion is represented in Formula (A.2):

$$E_\lambda = S_\lambda R_\lambda \tag{A.2}$$

where

E_λ is radiance/emission at wavelength λ;

S_λ is illuminant emission at wavelength λ;

R_λ is reflectance/transmission at wavelength λ.

Alternatively, this can be represented with the matrix/vector Formula (A.3):

$$e = Sr \tag{A.3}$$

where

e is the resulting radiance/emission vector;

S is a diagonal matrix containing illuminant emissions;

r is the starting reflectance/transmission vector.

A.1.4 Fluorescence to radiance/emission

The conversion of fluorescence to emitted radiance represents incident light from an illuminant being reflected (or transmitted) with fluorescence by the object represented by the fluorescence data. Fluorescence data are represented by a Donaldson matrix, and the application of a Donaldson matrix to a vector representing the illuminant emission results in a vector of light radiated from the object.

The number of columns corresponds to the spectralRange of the source illuminant, and the number of rows corresponds to the spectralRange of the resulting radiance/emission.

This is represented by the matrix Formula (A.4):

$$\begin{pmatrix} E_1 \\ E_2 \\ \\ . \\ : \\ \\ E_n \end{pmatrix} = \begin{pmatrix} D_{1,1} & \cdots & D_{1,m} \\ . & \cdots & . \\ : & \cdots & : \\ & \cdots & \\ D_{n,1} & \cdots & D_{n,m} \end{pmatrix} \begin{pmatrix} S_1 \\ S_2 \\ \\ . \\ : \\ \\ S_m \end{pmatrix} \quad \text{(A.4)}$$

where

S_j is the starting illuminant emission value for the j^{th} wavelength;

$D_{i,j}$ is the i,j^{th} element of the Donaldson matrix with dimension $n \times m$;

E_j is the resulting radiance/emission vector for the i^{th} wavelength.

A.1.5 Radiance/emission to reflection/transmission

The conversion of radiance/emission to reflectance/transmission represents the factoring out of the incident light from the light being reflected or transmitted by the object. The conversion of a radiance/emission vector to a reflectance/transmission vector can be represented as scalar product of a reflectance/transmission vector and a vector containing reciprocals of the illuminant emission values for each wavelength. Because a reciprocal of the source illuminant values for each wavelength is used this requires that the source illuminant values are non-zero. If any of the source illuminant values are zero then Reflectance/Transmission cannot be determined.

This requires that the vectors for the illuminant and reflectance/transmission share the same scalarRange.

The conversion is represented in Formula (A.5):

$$R_\lambda = \left(\frac{1}{S_\lambda} \right) E_\lambda \quad \text{(A.5)}$$

where

R_λ is reflectance (or transmittance) at wavelength λ;

S_λ is illuminant power at wavelength λ;

E_λ is emitted radiance at wavelength λ.

Alternatively this can be represented with Formula (A.6):

$$r = S^{-1} e \quad \text{(A.6)}$$

where

e is the emitted radiance vector;

S^{-1} is a diagonal matrix containing reciprocals of illuminant power;

r is the starting reflectance (or transmittance) vector.

A.1.6 Intensity radiance/emission to XYZ colorimetry

The conversion of radiance/emission intensities to XYZ colorimetry represents the application of CMFs to the radiance/emission data. This can be represented by applying a matrix containing the CMF to a vector containing the radiance/emission data. The resulting tristimulus values are expressed as absolute intensities since no relative white point is taken into consideration.

This requires that the spectralRange of the radiance/emission data matches the spectral range of the CMF.

This conversion is represented by Formula (A.7):

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = 683 \begin{pmatrix} \bar{x}_1 & \bar{x}_2 & \dots & \bar{x}_n \\ \bar{y}_1 & \bar{y}_2 & \dots & \bar{y}_n \\ \bar{z}_1 & \bar{z}_2 & \dots & \bar{z}_n \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ . \\ : \\ E_n \end{pmatrix} \tag{A.7}$$

where

$\bar{x}_i \ \bar{y}_i \ \bar{z}_i$ are the CMF values for wavelength *i*;

E_i is the radiance at wavelength *i*, in cd/m².

A.1.7 Relative radiance/emission to XYZ colorimetry

The conversion of radiance/emission to XYZ colorimetry relative to a given illuminant represents the application of an observer's CMFs to the radiance/emission data and factoring this by the Y tristimulus value of the illuminant. This can be represented by applying a normalization factor to the application of a matrix containing the CMF to a vector containing the radiance/emission data. The resulting tristimulus values are expressed as absolute intensities since no relative white point is taken into consideration.

This requires that the spectralRange of both the radiance/emission data and the relative illuminant match the spectral range of the CMF.

This conversion is represented by Formula (A.8):

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = k \begin{pmatrix} \bar{x}_1 & \bar{x}_2 & \dots & \bar{x}_n \\ \bar{y}_1 & \bar{y}_2 & \dots & \bar{y}_n \\ \bar{z}_1 & \bar{z}_2 & \dots & \bar{z}_n \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ . \\ : \\ E_n \end{pmatrix} \tag{A.8}$$

with

E_i is the emission spectrum at wavelength *i*;

$\bar{x}_i$, $\bar{y}_i$, $\bar{z}_i$ are the CMF values for wavelength *i*;

X, Y, Z are the resulting tristimulus values;

k is a normalizing constant for the relative illuminant (with values I_i for each wavelength *i*) and CMF, where $k = \frac{1}{\sum_{i=1}^{n} \bar{y}_i I_i}$.

A.1.8 Absolute/relative intent adjustments

ICC colour management supports two modes of operation defined by rendering intents. With the absolute intent PCS values directly correspond to measurement data. With the relative intent PCS values correspond to measurement values that have been adjusted relative to the media white point. ICC profiles can have tags that provide transforms for either relative or absolute or both.

Relative to absolute adjustment is performed by applying a diagonal matrix containing relative white divided by absolute white values. Absolute to relative adjustment is performed by applying a diagonal matrix containing absolute white divided by relative white values.

For a colorimetric PCS, the relative white values are defined as the CIEXYZ values found in the mediaWhitePoint tag, and the absolute white values are defined by the CIEXYZ values for the illuminant relative to the observer used for the colorimetric PCS. The illuminant and observer are defined by the PCC that describe the PCS if a custom colorimetric PCS is used. If a standard PCS is used, then the absolute white point is the colorimetry of the D50 illuminant for the CIE 1931 Standard 2-degree observer.

For a spectral PCS, the relative white values are defined as vector of PCS encoded values found in the contents of the spectralWhitePoint tag. Absolute white values for a spectral PCS are dependent on the PCS type:

— For either a reflective or transmissive PCS are defined as vectors containing 100 % reflectance/transmission for all wavelengths.

— Absolute white values are defined by the PCC illuminant in the case of emissive PCS.

— Absolute white values are not defined for spectrofluorescent based PCS. (Thus relative/absolute conversions are not possible in the case of a spectrofluorescent PCS.)

Conversions between absolute and relative data are performed using Formula (A.9):

$$\begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_N \end{pmatrix} = \begin{pmatrix} W_{\mathrm{Rel},1}/W_{\mathrm{Abs},1} & 0 & \cdots & 0 \\ 0 & W_{\mathrm{Rel},2}/W_{\mathrm{Abs},2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W_{\mathrm{Rel,N}}/W_{\mathrm{Abs,N}} \end{pmatrix} \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{pmatrix} \quad \text{(A.9)}$$

where

A_i is the i^{th} entry of an absolute PCS coordinate vector (CIEX, CIEY, or CIEZ for colorimetric PCS; or value for the i^{th} wavelength for a spectrally-based PCS);

$W_{\mathrm{Rel},i}$ is the i^{th} entry of the relative white vector (CIEX, CIEY, or CIEZ for colorimetric PCS; or value for the i^{th} wavelength for a spectrally-based PCS);

$W_{\mathrm{Abs},i}$ is the i^{th} entry of the absolute white vector (CIEX, CIEY, or CIEZ for colorimetric PCS; or value for the i^{th} wavelength for a spectrally-based PCS);

R_i is the i^{th} entry of a relative PCS coordinate vector (CIEX, CIEY, or CIEZ for colorimetric PCS; or value for the i^{th} wavelength for a spectrally-based PCS).

Conversion between relative and absolute date is performed using Formula (A.10):

$$\begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{pmatrix} = \begin{pmatrix} W_{\text{Abs},1}/W_{\text{Rel},1} & 0 & \cdots & 0 \\ 0 & W_{\text{Abs},2}/W_{\text{Rel},2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W_{\text{Abs,N}}/W_{\text{Rel,N}} \end{pmatrix} \begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_N \end{pmatrix} \quad \text{(A.10)}$$

where

R_i is the i^{th} entry of a relative PCS coordinate vector (CIEX, CIEY, or CIEZ for colorimetric PCS; or value for the i^{th} wavelength for a spectrally-based PCS);

$W_{\text{Abs},i}$ is the i^{th} entry of the absolute white vector (CIEX, CIEY, or CIEZ for colorimetric PCS; or value for the i^{th} wavelength for a spectrally-based PCS);

$W_{\text{Rel},i}$ is the i^{th} entry of the relative white vector (CIEX, CIEY, or CIEZ for colorimetric PCS; or value for the i^{th} wavelength for a spectrally-based PCS);

A_i is the i^{th} entry of an absolute PCS coordinate vector (CIEX, CIEY, or CIEZ for colorimetric PCS; or value for the i^{th} wavelength for a spectrally-based PCS).

Absolute/Relative PCS conversions are not possible for spectrally-based PCS operations when the either the absolute or relative white vector contains values of zero (which can happen in the case of using an emission based PCS). In this case it would be expected that a failure would occur when the adjustment transform operations are initialized.

A.1.9 Black point compensation

Black point compensation is defined as a colorimetric PCS operation. It is defined in ISO 18619.

A.1.10 Luminance Matching

The ISO 15076-1 standard colorimetric PCS, as well colorimetric PCS values in this document, are encoded in terms of normalized tristimulus values. This means that tristimulus values are normalized to a reference white, which is assumed to be the adapted white point. For a surface colour this is a perfect white diffuser viewed under a D50 illuminant, for a display it is the assumed white point (often the display white point), and in both cases the values are scaled so that the tristimulus Y of the reference white is 1,0. (These are relative tristimulus values, as described in CIE.15.) One consequence of this normalization is that differences in luminance between source and destination are not accounted for when connecting a pair of profiles using a colorimetric PCS.

In some cases it may be desirable for the colour management goal to take the actual luminances of source and destination into account, and this can be accomplished by using luminance information from the profiles to provide a scaling of tristimulus values during PCS processing by the CMM. The absolute photometric luminance in cd/m^2 can be provided in the CIE Y value of the illuminant field of the spectralViewingConditions tag (of a profile based on this document) or the CIE Y value of the luminanceTag (of a profile based on ISO 15076-1). When such tags are not available then the default luminance of 160 cd/m^2 (associated with the perceptual PCS defined by ISO 15076-1, or the standard viewing condition P2 specified for graphic arts and photography in ISO 3664) can be used. Additionally, an override of a profile's luminance can be provided by the CIE Y value of the illuminant field of the spectralViewingConditions defined by alternate PCC when they are provided to the CMM for a profile.

When luminance matching is desired, the instructed CMM performs a Colorimetric PCS operation that scales the tristimulus values by the ratio of the luminance associated with the source profile divided by the luminance associated with the destination profile.

An important caution related to the use of luminance matching is that clipping may occur when tristimulus values are scaled outside the range that the destination profile is capable of adequately handling.

A.2 Various PCS operations

Figure A.1 provides a high-level overview of various PCS operations and conversions. When two profiles are connected only one ICC in and one ICC out connection point is used for the profiles. The location of these connection points are determined by the type of PCS that is used for each profile. The closest connection distance is used, and profiles that have equivalent PCC will connect to the same PCS.

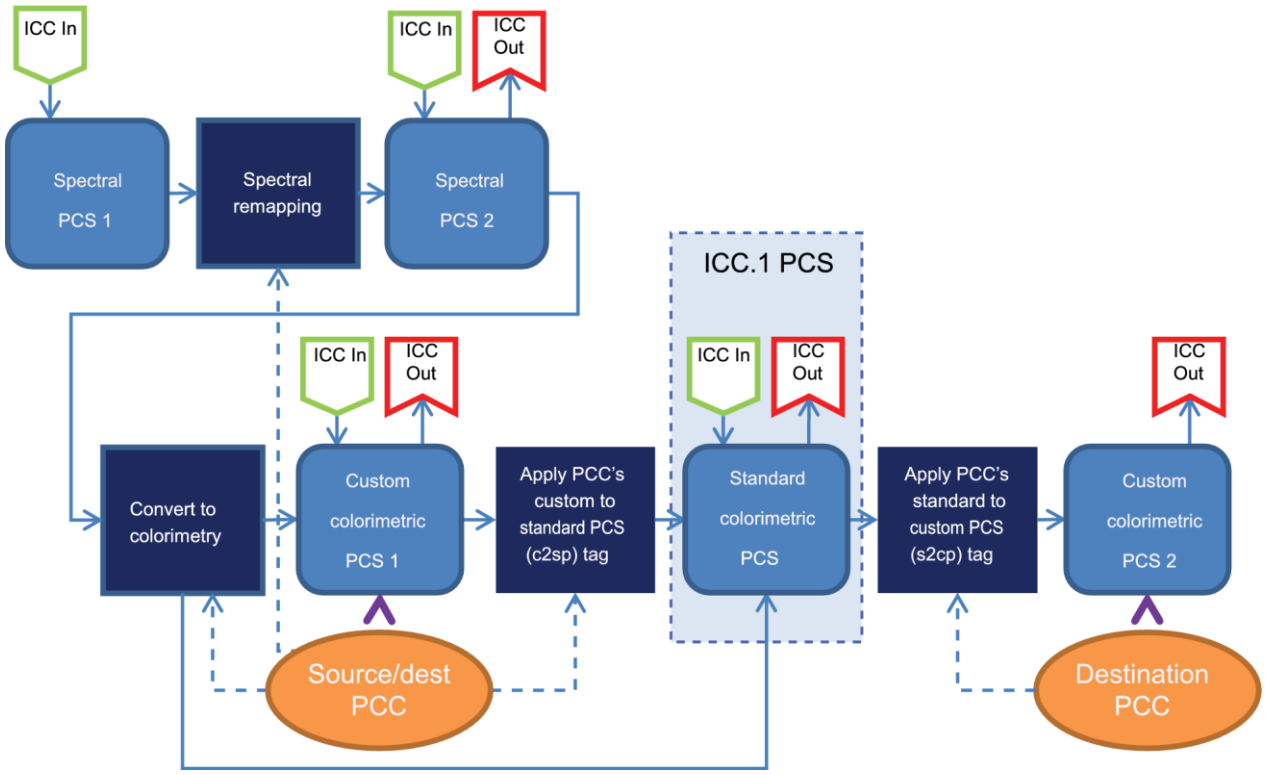


Figure A.1 — Spectral and colorimetric PCS operations

A.3 Pseudo-code description of PCS to PCS transformations

A.3.1 Overview

The various PCS mapping possibilities are outlined in Table A.1.

Table A.1 — Various PCS mapping possibilities

	From Lab	From XYZ	From Reflectance	From Transmittance/ Transmissive	From Radiant/ Emission	From Fluorescence
To Lab	Yes (A.3.2)	Yes (A.3.4)	Using PCC (A.3.6)	Using PCC (A.3.11)	Using PCC (A.3.16)	Using PCC (A.3.21)
To XYZ	Yes (A.3.3)	Yes (A.3.5)	Using PCC (A.3.7)	Using PCC (A.3.12)	Using PCC (A.3.17)	Using PCC (A.3.22)
To Reflectance	No	No	Yes (A.3.8)	Yes (A.3.13)	Extract PCC illuminant (A.3.18)	Apply then extract PCC illuminant (A.3.23)
To Transmittance/ Transmissive	No	No	Yes (A.3.9)	Yes (A.3.14)	Use PCC illuminant (A.3.19)	Apply then extract PCC illuminant (A.3.24)
To Radiant / Emission	No	No	Apply PCC Illuminant (A.3.10)	Apply PCC illuminant (A.3.15)	Yes (A.3.20)	Apply PCC illuminant (A.3.25)
To Fluorescence	No	No	No	No	No	Exact match required (A.3.26)

PCS processing uses PCC to determine the transforms that are needed to convert between the various connections outlined in Table A.1. Each profile has PCC that are made up of colorimetric and spectral PCS information from the profile header (see 7.2), information from spectralViewingConditionsTag (see 9.2.105), and transformations in the customToStandardPCCTag (see 9.2.56) and standardToCustomPCCTag (see 9.2.107). PCC information is obtained either from the profiles involved in the connection or from outside sources provided to the CMM.

PCS connection transforms involving a colorimetric PCS (either Lab or XYZ) are generally performed relative to the ISO 15076-1 standard colorimetric PCS, which uses the CIE 1931 Standard 2-degree observer with a D50 illuminant. The conversion is done in two general steps, which are each made up of sub-steps. The first general step converts to XYZ values for the standard colorimetric PCS. The second step converts from XYZ values for the standard colorimetric PCS. However, the standard colorimetric PCS is not needed or used when identical colorimetric PCS usage is indicated by the PCC information for both sides of the connection. In this case only the custom colorimetric PCS is used.

For each of these general steps the sub-steps used to define them will involve zero or more of the following steps: conversion to/from XYZ, potential application of absolute/relative white point adjustment, potential application of black point adjustment, conversion using the PCC customToStandardPCCTag, or conversion using the standardToCustomPCCTag.

PCS connection transforms involving spectrally-based PCS can involve resampling with expansion or compression of spectral information by wavelength to match observer, illuminant definitions defined by the relevant PCC or connection spectral sampling requirements between the two profiles. Additionally, illuminant information is applied or factored out.

When connecting a spectrally-based PCS to a colorimetric PCS a conversion to colorimetry is needed. First the conversion to colorimetry is performed by applying the source PCC observer and illuminant. Then colorimetric PCS connection is performed (as if the source PCS was colorimetric).

Fluorescent PCS connection is different from other spectral PCS connection in that spectralRange conversion is performed on the illuminant (if it doesn't match the input spectral input range of the Fluorescent PCS). This is due to the difficulty of resampling Donaldson matrices that correspond to the spectral PCS. Linear resampling of the illuminant is performed. However, this is generally not advisable

so it is recommended that workflows be set up to use an illuminant that matches the spectral input range of the fluorescent PCS.

NOTE Most of the PCS operations described in this annex can be implemented as matrix operations that can be concatenated for performance purposes.

A.3.2 From Lab to Lab

The following pseudo-code describes this PCS conversion:

```
XYZ = Convert_from_PCSLAB();
If source relative/absolute intent and/or black point compensation adjustment is needed
    XYZ = Scale_and_shift(XYZ, …);
Endif
If sourcePCS != destinationPCS
    If sourcePCS != standardPCS
        XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
    Endif
    If destinationPCS != standardPCS
        XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
    Endif
Endif
If luminance matching is needed
    XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If destination relative/absolute intent and/or black point compensation adjustment is needed
    XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSLAB(XYZ);
```

A.3.3 From Lab to XYZ

The following pseudo-code describes this PCS conversion:

```
XYZ = Convert_from_PCSLAB();
If source relative/absolute intent and/or black point compensation adjustment is needed
    XYZ = Scale_and_shift(XYZ, …);
Endif
If sourcePCS != destinationPCS
    If sourcePCS != standardPCS
        XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
```



```
        Endif
        If destinationPCS != standardPCS
            XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
    Endif
    If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
    Endif
    If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
    Endif
    Convert_to_PCSXYZ(XYZ);
```

A.3.4 From XYZ to Lab

The following pseudo-code describes this PCS conversion:

```
    XYZ = Convert_from_PCSXYZ();
    If source relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
    Endif
    If sourcePCS != destinationPCS
        If sourcePCS != standardPCS
            XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
            XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
    Endif
    If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
    Endif
    If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
    Endif
    Convert_to_PCSLAB(XYZ);
```

A.3.5 From XYZ to XYZ

The following pseudo-code describes this PCS conversion:

```
XYZ = Convert_from_PCSXYZ();
If source relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
If sourcePCS != destinationPCS
        If sourcePCS != standardPCS
                XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
                XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSXYZ(XYZ);
```

A.3.6 From Reflectance to Lab

The following pseudo-code describes this PCS conversion:

```
reflectance = Convert_from_PCS_Reflectance();
If source relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
If spectralRange(reflectance) != spectralRange(source_PCC_illuminant)
        reflectance = Adjust_range(reflectance, spectralRange(source_PCC_illuminant));
Endif
radiance = Apply_illuminant(reflectance, source_PCC_illuminant);
If spectralRange(radiance) != spectralRange(source_PCC_observer)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_observer));
```

```
Endif
XYZ = Apply_relative_observer(radiance, source_PCC_Observer, src_PCC_illuminant);
If black point compensation adjustment is needed
        XYZ=Scale_and_shift(XYZ, ...);
Endif
        If sourcePCC_PCS != destinationPCS
        If sourcePCS != standardPCS
                XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
                XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSLAB(XYZ);
```

A.3.7 From Reflectance to XYZ

The following pseudo-code describes this PCS conversion:

```
reflectance = Convert_from_PCS_Reflectance();
If source relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
If spectralRange(reflectance) != spectralRange(source_PCC_illuminant)
        reflectance = Adjust_range(reflectance, spectralRange(source_PCC_illuminant));
Endif
radiance = Apply_illuminant(reflectance, source_PCC_illuminant);
If spectralRange(radiance) != spectralRange(source_PCC_observer)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_observer));
Endif
XYZ = Apply_relative_observer(radiance, source_PCC_Observer, src_PCC_illuminant);
```

```
If black point compensation adjustment is needed
        XYZ=Scale_and_shift(XYZ, ...);
Endif
If sourcePCC_PCS != destinationPCS
        If sourcePCS != standardPCS
                XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
                XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSXYZ(XYZ);
```

A.3.8 From Reflectance to Reflectance

The following pseudo-code describes this PCS conversion:

```
reflectance = Convert_from_PCS_Reflectance();
If source relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
If spectralRange(reflectance) != spectralRange(destination_PCS)
        reflectance = Adjust_range(reflectance, spectralRange(destination_PCS));
Endif
If destination relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
Convert_to_PCS_Reflectance(reflectance);
```

A.3.9 From Reflectance to Transmittance

The following pseudo-code describes this PCS conversion:

```
reflectance = Convert_from_PCS_Reflectance();
```



```
If source relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
If spectralRange(reflectance) != spectralRange(destination_PCS)
        reflectance = Adjust_range(reflectance, spectralRange(destination_PCS));
Endif
If destination relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
Convert_to_PCS_Transmittance(reflectance);
```

A.3.10 From Reflectance to Radiance

The following pseudo-code describes this PCS conversion:

```
reflectance = Convert_from_PCS_Reflectance();
If source relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
If spectralRange(reflectance) != spectralRange(source_PCC_illuminant)
        reflectance = Adjust_range(reflectance, spectralRange(source_PCC_illuminant));
Endif
radiance = ApplyIlluminant(reflectance, source_PCC_illuminant);
If spectralRange(radiance) != spectralRange(destination_PCS)
        radiance = Adjust_range(radiance, spectralRange(destination_PCS));
Endif
If destination relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
Endif
Convert_to_PCS_Radiance(radiance);
```

A.3.11 From Transmittance to Lab

The following pseudo-code describes this PCS conversion:

```
transmittance = Convert_from_PCS_Transmittance();
If source relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
Endif
```



```
If spectralRange(transmittance) != spectralRange(source_PCC_illuminant)
        transmittance = Adjust_range(transmittance, spectralRange(source_PCC_illuminant));
Endif
radiance = Apply_illuminant(transmittance, source_PCC_illuminant);
If spectralRange(radiance) != spectralRange(source_PCC_observer)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_observer));
Endif
XYZ = Apply_relative_observer(radiance, source_PCC_Observer, src_PCC_illuminant);
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If black point compensation adjustment is needed
        XYZ=Scale_and_shift(XYZ, ...);
Endif
If sourcePCC_PCS != destinationPCS
        If sourcePCS != standardPCS
                XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
                XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
Endif
If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSLab(XYZ);
```

A.3.12 From Transmittance to XYZ

The following pseudo-code describes this PCS conversion:

```
transmittance = Convert_from_PCS_Transmittance();
If source relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
Endif
If spectralRange(transmittance) != spectralRange(source_PCC_illuminant)
```



```
        transmittance = Adjust_range(transmittance, spectralRange(source_PCC_illuminant));
    Endif
    radiance = Apply_illuminant(transmittance, source_PCC_illuminant);
    If spectralRange(radiance) != spectralRange(source_PCC_observer)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_observer));
    Endif
    XYZ = Apply_relative_observer(radiance, source_PCC_Observer, src_PCC_illuminant);
    If black point compensation adjustment is needed
        XYZ=Scale_and_shift(XYZ, ...);
    Endif
    If sourcePCC_PCS != destinationPCS
        If sourcePCS != standardPCS
            XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
            XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
    Endif
    If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
    Endif
    If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
    Endif
    Convert_to_PCSLab(XYZ);
```

A.3.13 From Transmittance to Reflectance

The following pseudo-code describes this PCS conversion:

```
    transmittance = Convert_from_PCS_Transmittance();
    If source relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
    Endif
    If spectralRange(transmittance) != spectralRange(destination_PCS)
        transmittance = Adjust_range(transmittance, spectralRange(destination_PCS));
```



```
    Endif
    If destination relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
    Endif
Convert_to_PCS_Reflectance(transmittance);
```

A.3.14 From Transmittance to Transmittance

The following pseudo-code describes this PCS conversion:

```
    transmittance = Convert_from_PCS_Transmittance();
    If source relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
    Endif
    If spectralRange(transmittance) != spectralRange(destination_PCS)
        transmittance = Adjust_range(transmittance, spectralRange(destination_PCS));
    Endif
    If destination relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
    Endif
    Convert_to_PCS_Transmittance(transmittance);
```

A.3.15 From Transmittance to Radiance

The following pseudo-code describes this PCS conversion:

```
    transmittance = Convert_from_PCS_Reflectance();
    If source relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
    Endif
    If spectralRange(transmittance) != spectralRange(source_PCC_illuminant)
        transmittance = Adjust_range(transmittance, spectralRange(source_PCC_illuminant));
    Endif
    radiance = ApplyIlluminant(transmittance, source_PCC_illuminant);
    If spectralRange(radiance) != spectralRange(destination_PCS)
        radiance = Adjust_range(radiance, spectralRange(destination_PCS));
    Endif
    If destination relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
```



```
Endif
Convert_to_PCS_Radiance(radiance);
```

A.3.16 From Radiance to Lab

The following pseudo-code describes this PCS conversion:

```
radiance = Convert_from_PCS_Radiance();
If source relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
Endif
If spectralRange(radiance) != spectralRange(source_PCC_observer)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_observer));
Endif
XYZ = Apply_intensity_observer(radiance, source_PCC_Observer);
If black point compensation adjustment is needed
        XYZ=Scale_and_shift(XYZ, ...);
Endif
If sourcePCC_PCS != destinationPCS
        If sourcePCS != standardPCS
                XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
                XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSLAB(XYZ);
```

A.3.17 From Radiance to XYZ

The following pseudo-code describes this PCS conversion:

```
radiance = Convert_from_PCS_Radiance();
```



```
If source relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
Endif
If spectralRange(radiance) != spectralRange(source_PCC_observer)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_observer));
Endif
XYZ = Apply_intensity_observer(radiance, source_PCC_Observer);
If black point compensation adjustment is needed
        XYZ=Scale_and_shift(XYZ, ...);
Endif
If sourcePCC_PCS != destinationPCS
        If sourcePCS != standardPCS
        XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
        XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSXYZ(XYZ);
```

A.3.18From Radiance to Reflectance

The following pseudo-code describes this PCS conversion:

```
radiance = Convert_from_PCS_Radiance();
If source relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
Endif
If spectralRange(radiance) != spectralRange(source_PCC_illuminant)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_illuminant));
```



```
Endif
reflectance = FactorOutIlluminant(radiance, source_PCC_illuminant);
If spectralRange(reflectance) != spectralRange(destination_PCS)
        reflectance = Adjust_range(reflectance, spectralRange(destination_PCS));
Endif
If destination relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
Convert_to_PCS_Reflectance(reflectance);
```

A.3.19 From Radiance to Transmittance

The following pseudo-code describes this PCS conversion:

```
radiance = Convert_from_PCS_Radiance();
If source relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
Endif
If spectralRange(radiance) != spectralRange(source_PCC_illuminant)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_illuminant));
Endif
transmittance = FactorOutIlluminant(radiance, source_PCC_illuminant);
If spectralRange(transmittance) != spectralRange(destination_PCS)
        transmittance = Adjust_range(transmittance, spectralRange(destination_PCS));
Endif
If destination relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
Endif
Convert_to_PCS_Transmittance(transmittance);
```

A.3.20 From Radiance to Radiance

The following pseudo-code describes this PCS conversion:

```
radiance = Convert_from_PCS_Radiance();
If source relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
Endif
If spectralRange(radiance) != spectralRange(destination_PCS)
```

```
        radiance = Adjust_range(radiance, spectralRange(destination_PCS));
    Endif
    If destination relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
    Endif
    Convert_to_PCS_Radiance(radiance);
```

A.3.21 From Fluorescence to Lab

The following pseudo-code describes this PCS conversion:

```
    fluorescence = Convert_from_PCS_Fluorescence();
    illuminant = source_PCC_illuminant
    If spectralInputRange(fluorescence) != spectralRange(source_PCC_illuminant)
        illuminant = Adjust_range(source_PCC_illuminant, spectralInputRange(fluorescence));
    Endif
    radiance = Apply_Donaldson_matrix(fluorescence, illuminant);
    If spectralRange(radiance) != spectralRange(source_PCC_observer)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_observer));
    Endif
    illuminant2 = Adjust_range(illuminant, spectralOutputRange(fluorescence));
    If (spectralRange(illuminant2) != spectralRange(source_PCC_observer)
        illuminant2 = Adjust_range(illuminant2, spectralRange(source_PCC_observer);
    Endif
    XYZ = Apply_relative_observer(radiance, source_PCC_Observer, illuminant2);
    If source relative/absolute intent and/or black point compensation adjustment is needed
        XYZ=Scale_and_shift(XYZ, ...);
    Endif
    If sourcePCC_PCS != destinationPCS
        If sourcePCS != standardPCS
            XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
            XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
    Endif
```

```
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSLab(XYZ);
```

A.3.22 From Fluorescence to XYZ

The following pseudo-code describes this PCS conversion:

```
fluorescence = Convert_from_PCS_Fluorescence();
illuminant = source_PCC_illuminant
If spectralInputRange(fluorescence) != spectralRange(source_PCC_illuminant)
        illuminant = Adjust_range(source_PCC_illuminant, spectralInputRange(fluorescence));
Endif
radiance = Apply_Donaldson_matrix(fluorescence, illuminant);
If spectralRange(radiance) != spectralRange(source_PCC_observer)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_observer));
Endif
illuminant2 = Adjust_range(illuminant, spectralOutputRange(fluorescence));
If (spectralRange(illuminant2) != spectralRange(source_PCC_observer)
        illuminant2 = Adjust_range(illuminant2, spectralRange(source_PCC_observer);
Endif
XYZ = Apply_relative_observer(radiance, source_PCC_Observer, illuminant2);
If source relative/absolute intent and/or black point compensation adjustment is needed
        XYZ=Scale_and_shift(XYZ, ...);
Endif
If sourcePCC_PCS != destinationPCS
        If sourcePCS != standardPCS
                XYZ = Apply_Tag(XYZ, source_PCC_customToStandardPCSTag);
        Endif
        If destinationPCS != standardPCS
                XYZ = Apply_Tag(XYZ, destination_PCC_standardToCustomPCSTag);
        Endif
```

```
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If destination relative/absolute intent and/or black point compensation adjustment is needed
        XYZ = Scale_and_shift(XYZ, …);
Endif
Convert_to_PCSXYZ(XYZ);
```

A.3.23 From Fluorescence to Reflectance

The following pseudo-code describes this PCS conversion:

```
fluorescence = Convert_from_PCS_Fluorescence();
illuminant = source_PCC_illuminant
If spectralInputRange(fluorescence) != spectralRange(source_PCC_illuminant)
        illuminant = Adjust_range(source_PCC_illuminant, spectralInputRange(fluorescence));
Endif
radiance = Apply_Donaldson_matrix(fluorescence, illuminant);
If spectralRange(radiance) != spectralRange(source_PCC_illuminant)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_illuminant));
Endif
reflectance = FactorOutIlluminant(radiance, source_PCC_illuminant);
If spectralRange(reflectance) != spectralRange(destination_PCS)
        reflectance = Adjust_range(reflectance, spectralRange(destination_PCS));
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If destination relative/absolute adjustment is needed
        reflectance = Scale(reflectance, …);
Endif
Convert_to_PCS_Reflectance(reflectance);
```

A.3.24 From Fluorescence tor Transmitance

The following pseudo-code describes this PCS conversion:

```
fluorescence = Convert_from_PCS_Fluorescence();
```



```
illuminant = source_PCC_illuminant
If spectralInputRange(fluorescence) != spectralRange(source_PCC_illuminant)
        illuminant = Adjust_range(source_PCC_illuminant, spectralInputRange(fluorescence));
Endif
radiance = Apply_Donaldson_matrix(fluorescence, illuminant);
If spectralRange(radiance) != spectralRange(source_PCC_illuminant)
        radiance = Adjust_range(radiance, spectralRange(source_PCC_illuminant));
Endif
transmittance = FactorOutIlluminant(radiance, source_PCC_illuminant);
If spectralRange(transmittance) != spectralRange(destination_PCS)
        transmittance = Adjust_range(transmittance, spectralRange(destination_PCS));
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
If destination relative/absolute adjustment is needed
        transmittance = Scale(transmittance, …);
Endif
Convert_to_PCS_Transmittance(transmittance);
```

A.3.25 From Fluorescence to Radiance

The following pseudo-code describes this PCS conversion:

```
fluorescence = Convert_from_PCS_Fluorescence();
illuminant = source_PCC_illuminant
If spectralInputRange(fluorescence) != spectralRange(source_PCC_illuminant)
        illuminant = Adjust_range(source_PCC_illuminant, spectralInputRange(fluorescence));
Endif
radiance = Apply_Donaldson_matrix(fluorescence, illuminant);
If spectralRange(radiance) != spectralRange(destination_PCS)
        radiance = Adjust_range(radiance, spectralRange(destination_PCS));
Endif
If luminance matching is needed
        XYZ = Scale(XYZ, source_luminance / destination_luminance)
Endif
```

```
If destination relative/absolute adjustment is needed
        radiance = Scale(radiance, …);
Endif
Convert_to_PCS_Radiance(radiance);
```

A.3.26 From Fluorescence to Fluorescence

No conversion is made. The connection is only valid if spectralFluorescentRange(source_PCS) is the same as the spectralFluorescentRange(destination_PCS).

Annex B
(informative)

Gamut Boundary Description

B.1 Introduction

The Gamut Boundary Description tag introduces a precise method of describing a gamut boundary. The Gamut Boundary description describes the gamut boundary as a collection of 3D vertices and faces.

The Gamut Boundary Description tag can be used to individually describe the gamut boundaries for the different rendering intents. For the perceptual intent, the actual reference medium gamut that is used can be encoded. Separate gamut boundaries for relative and absolute intents are permitted because the floating-point tags permit separate transforms.

The gamut boundary description is composed of a set of vertices and faces. The vertices contain PCS values and optional device values. The faces are described by a set of vertex IDs. Useful gamut boundary attributes such as edges and surface normals can be computed from the vertices and faces.

If the gamut boundary description includes device values, then the gamut boundary description can be used to transform values from PCS to device values.

B.2 Computing the entries in a gamutBoundaryDescriptionType tag:

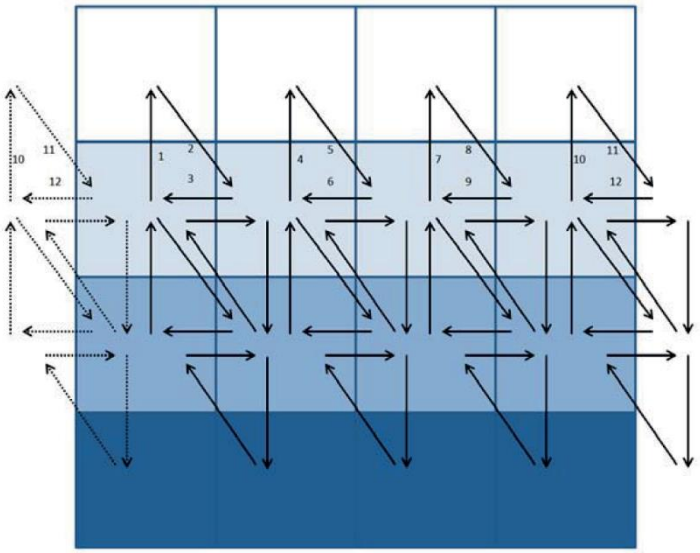


Figure B.1 — Gamut boundary face selection from device coordinates image

To compute the faces and vertices for the gbd tag for a given rendering intent the following procedure can be used:

1) Generate a set of device coordinates on the gamut boundary of the encoding. These coordinates are arranged so that the ratio of the relative colorant amounts varies in the horizontal direction, and the total colorant amount varies in the vertical direction. (An example image is given by Green in [10]). The white point and black point shall be repeated across the first and last rows in the coordinate array.

2) Using the AToBx LUT for the rendering intent, convert the device coordinates in step 1 to the PCS.

3) If the AToBx LUT contains more than three input channels, convert the PCS coordinates back to device coordinates using the BToAx LUT and then back to PCS coordinates, in both cases using the selected rendering intent.

4) The data from step 2 (or step 3) is read row-wise and arranged as a n*m x3 array to form the vertex array.

5) To construct the face array for this data, start with the upper left device coordinate and move clockwise to the two coordinates in the next row, as shown in Figure B.1. The first row of the faces list is therefore [1, m+2, m+1] where m is the number of coordinates in the mxn set of device coordinates. The next row in the faces list is [1, 2, m+1]. Continue to move through the device coordinates until the face list is fully populated with one row per face.

The following Matlab code can be used to generate a face array for an nxm array of device coordinates:

```
% facemat array offsets
q=zeros(m,3);
t=(0:n-1)*m;
H=zeros(m*(n-1),3);
for i=1:n-1
H((i-1)*m+1 :m*i,:)=q+t(i);
end
% left triangles facemat array
L=(1:m)';M=[(m+2):(m+m),m+1]';N=((m+1):m+m)';
LMN=[L,M,N];
R=repmat(LMN,n-1,1);
facematL=R+H;
% right triangles facemat array
O=(1:m)';Q=[2:m,1]';S=[(m+2):(m+m),m+1]';
OQS=[O,Q,S];
T=repmat(OQS,n-1,1);
facematR=T+H;
faces=[facematL;facematR];
```

B.3 Gamut mapping

B.3.1 General

Gamut mapping can be performed using the data defined in a gamut boundary description tag. For Colorimetric rendering intents, the mapping is essentially a clipping of all out-of-gamut coordinates to the boundary of the destination gamut. For Perceptual and Saturation intents, the mapping can involve a compression from source to destination gamut so that clipping artefacts are avoided. A number of algorithms have been defined for both clipping and compression. Below some operations using a gamut boundary descriptor based on a list of vertices and faces is described.

To compress a coordinate to the surface of a gamut, it is moved towards a point on the achromatic axis by an amount determined by the distance between the coordinate, the gamut surface and the achromatic axis, and the selected compression function.

In order to find the point on the gamut surface that a given coordinate maps to, it is necessary to find the point of intersection between the plane on which a face lies and a vector representing the line of clipping or compression from the point being mapped. It is then necessary to determine whether the intersection lies inside the face. This process is iterated for all the faces in the face list, and the face intersection closest to the point being mapped is selected. Finally, the compression function is applied.

B.3.2 Intersection between vector and plane

Given the three vertices of a face, the plane formula, Formula (B.1), is:

$$[x, y, z] \cdot [n_x, n_y, n_z] = k \tag{B.1}$$

where N is the normal to the plane [Formulae (B.2) and (B.3)]:

$$N = \left(P_1 - P_2\right) \times \left(P_3 - P_2\right) \tag{B.2}$$

and

$$k = NP_1 \tag{B.3}$$

The mapping vector L intersects the plane P at point S in 3D space, as shown in Formulae (B.4) and (B.5).

$$t = \frac{N \cdot \left(P_1 - L_1\right)}{N \cdot \left(L_2 - L_1\right)} \tag{B.4}$$

$$S = L_1 + t\left(L_2 - L_1\right) \tag{B.5}$$

B.3.3 Determine if an intersection between line and face lies inside a face.

The intersection point between a mapping line and the plane in which a given face lies can be defined by Barycentric coefficients with respect to the coordinates of the face. If the three coefficients lie in the range [0,1] and sum to unity (within a small tolerance) the point lies inside the face.

Annex C
(informative)

ICC colour appearance model transformations

C.1 Introduction

The ICC colour appearance model (iccCAM) is based on the CIECAT02 transform as published by the CIE[4]. It is a simple modification of CIECAM02 to obtain a stable transform from measured XYZ to perceived opponent-colour Jab and back. All parameters specifying the viewing conditions and visual measures as specified by the original model are also supported by the iccCAM.

The top-level structure of CIECAM02 from XYZ to Jab is as follows:

a) The XYZ values are linearly transformed to a chromatic-adaptation basis and independently scaled to the input white to produce Von-Kries-adapted values. The chromatic-adaptation basis that is used in CIECAM02 is called CIECAT02.

b) The adapted XYZ values are added to non-adapted values in a pre-selected proportion D called the degree of adaptation.

c) The resulting RGB values are converted to the Hunt-Pointer-Estevez (HPE) R'G'B' space, and then subjected to a power-function nonlinearity (hyperbolic function) to become $R'_aG'_aB'_a$.

d) The opponent-colour values a, b are computed as linear combinations of $R'_aG'_aB'_a$.

e) The lightness J is computed starting from another linear combination A of $R'_aG'_aB'_a$. This involves dividing A by the value A_w for white, and evaluating a non-integer power of the quotient.

The basic problem with CIECAM02 is due to the choice of "sharpened" CIECAT02 primaries. That means the chromaticity triangle of the CIECAT02 primaries intersects the spectrum locus several times. In turn, this implies there are illuminants and colours that are physically producible but lie outside the triangle. Three kinds of problem can thereby arise in CIECAT02, quite apart from its context in CIECAM02:

1) An illuminant whose chromaticity lies on an edge of the triangle causes a zero Von-Kries denominator, and hence a division by zero in creating the Von-Kries ratios.

2) Because the CIECAT02 triangle falls slightly outside the HPE triangle, adaptation to a bluish illuminant moves a colour from inside the HPE triangle to outside the HPE triangle, resulting in negative R'G'B' values and hence a negative A value. The value A_w on the other hand is always positive as shown in by Luo[5]. This issue, also referred to as the yellow-blue problem, is solved in the CIE reportership R8-07[6].

3) A colour that lies outside the CIECAT02 triangle has at least one negative coordinate, so when chromatic adaptation happens, that colour moves in the wrong direction in the chromaticity space. For example, for CIECAT02 there is a band of purple colours within the spectrum locus that lies outside the primary triangle for CIECAT02 RGB. If the starting point a white light and adaptation to a more purple light occurs, all the purple colours within the CIECAT02 triangle shift toward green (away from purple), but colours that are outside the CIECAT02 triangle becomes more purple, contrary to actual experience.

Apart from the above mentioned problems, there are additional hazards because CIECAM02 applies a power function (power < 1) to the R'G'B' values. This results in an infinite slope of the power function at zero, destabilizing colour management algorithms for colours going to black [XYZ = (0,0,0)].

C.2 The ICC colour appearance model

A simple way to resolve the previously mentioned problems is as follows:

First, negative R'G'B' values need to be avoided. As a consequence, the HPE triangle will lie outside or be coincident with any candidate CIECAT02 triangle. On the other hand, to assure internal consistency, the candidate Von-Kries primaries make a triangle that circumscribes the spectrum locus. As mentioned by Süsstrunk and Brill[4,5], "Three chromaticity sets are nested: The HPE triangle encompasses the modified CIECAT02 triangle, which in turn encompasses the spectrum locus".

A simple and stable solution is obtained by replacing CIECAT02 with the HPE triangle. Even though HPE is not the best set of fundamental primaries, it is a significant improvement over a Von-Kries adaptation in XYZ as implemented in the CIELAB model, which is commonly used in colour management.

As a consequence, the range of colours and illuminants for which the model is invertible is given by the range of positive RGB values defined by HPE triangle for the colours, and strict positive RGB values as defined by the HPE triangle for the illuminants. If colours outside this region are used, they have to be mapped onto the boundary of the RGB space defined by the HPE triangle. Illuminants are assumed to be always real hence no mapping strategy is needed. In the iccCAM, colours are mapped in the HPE RGB space by clipping component by component the values to the nearest value.

The instability near the black point is due to the hyperbolic function in Formula (C.1).

$$f(x)=\frac{400(F_L\,x)^{0,42}}{27,13+(F_L\,x)^{0,42}} \tag{C.1}$$

where x ranges from 0 to 1 (in CIECAM02 x represents R'/100, G'/100 and B'/100).

For x going to zero, the derivative of the function $f(x)$ goes to infinity, resulting in unstable behaviour. This can be solved by replacing $f(x)$ with the function $g(x)$ defined in Formulae (C.2) and (C.3):

$$g_1(x)=\left[\frac{(1,2307)}{h(1)}h(x)-0,2307\right]f(1) \tag{C.2}$$

$$\text{for } x>\frac{4}{255}$$

$$g_2(x)=\frac{g_1\left(\frac{4}{255}\right)}{\frac{4}{255}}x \tag{C.3}$$

$$\text{for } x<=\frac{4}{255}$$

where $h(x)=\frac{400(F_L\,x)^{0,3169}}{27,13+(F_L\,x)^{0,3169}}$.

Annex D
(informative)

Named colour profiles

D.1 Introduction

In addition to the information specified in ISO 15076-1, the namedColor profile can also specify, for example, tint values, spectral PCS and spectral over black in the namedColorTag. An example of the namedColorTag is given in Figure D.1.

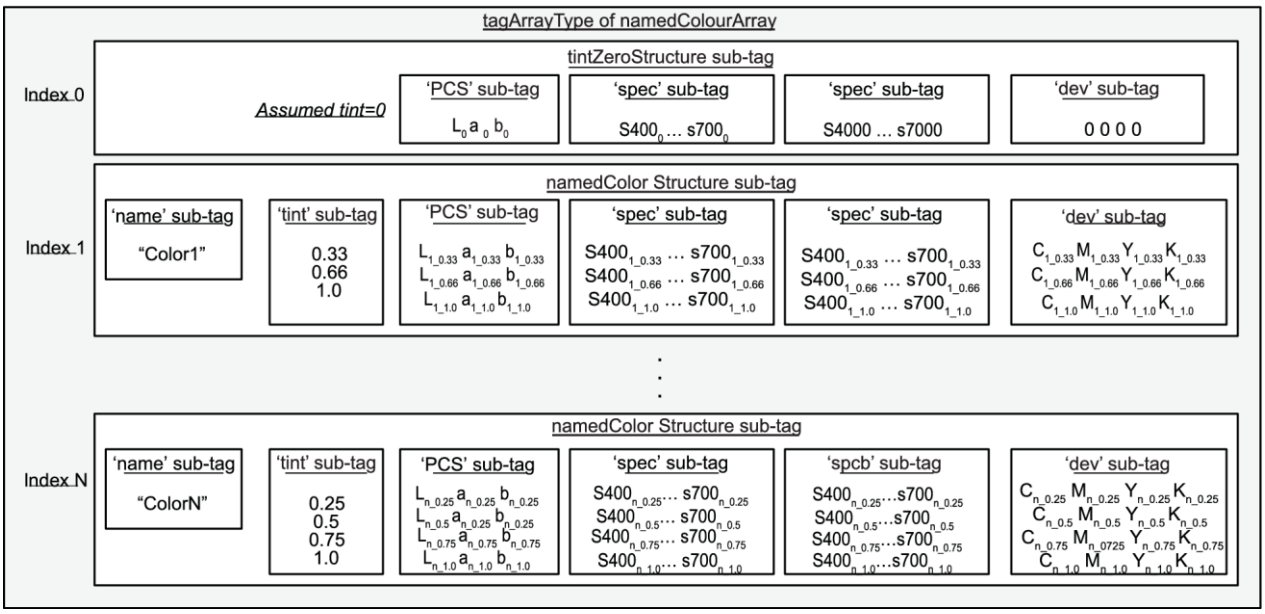


Figure D.1 — namedColorTag example

Where the deviceData and the tintValues are optional, the pcsData is required if the PCS is non-zero in profile header and the spectralData is required if the spectralPCS is non-zero in profile header.

D.2 Rendering intent of a named colour

The tintZeroStructure is used to specify the information on a substrate as well as over a black substrate. The PCS or spectral PCS of the substrate enables the value of a named colour represented with either the ICC-absolute colorimetric or the relative colorimetric rendering intent. The rendering intent field in the header is used to indicate which one is to be used.

D.3 Spectral calculation for a tint value

For a named colour with a tint value in between two tint values, a linear interpolation is assumed.

Example 1:

A colour with name of 'Color1' and tint value of 20 %.

The tint value of 20% is in between the zero tint value and the tint value of 33 %. The estimated CIELab value is shown in Formulae (D.1) to (D.3):

$$L^*_{1\_0,2}=20/33*(L^*_{1\_0,33}-L0)+L0 \quad \text{(D.1)}$$

$$a^*_{1\_0,2}=20/33*(a^*_{1\_0,33}-L0)+L0 \quad \text{(D.2)}$$

$$b^*_{1\_0,2}=20/33*(L^*_{1\_0,33}-L0)+L0 \quad \text{(D.3)}$$

Example 2:

A colour with name of 'ColorN' and tint value of 40 %.

The tint value of 40 % is in between the tint value of 25 % and the tint value of 50 %.

The estimated spectral values of this colour are shown in Formulae (D.4) and (D.5):

$$s400\_0,5=0,6*(\ s400\_0,50-s400\_0,25)+\ s400\_0,25 \quad \text{(D.4)}$$

…

$$s700\_0,5=0,6*(\ s700\_0,5-s700\_0,25)+\ s700\_0,25 \quad \text{(D.5)}$$

Similarly, the PCS values and the spectral values over black can also be obtained.

D.4 Overprint calculation

One way to calculate a spot colour overprint in between two tint values is to apply an overprint model on the tristimulus values obtained from the corresponding spectral values on the substrate and over the substrate. One bibliography reference outlines a few possible methods[9].

D.5 Example of a namedColor profile in a colour management workflow

The PCS or spectral PCS representations enable CMM to process a named colour in conjunction with a namedColor profile, a device destination profile, or/and a source profile. For example, a spot colour viewed on a sRGB display with the perceptual rendering intent is simulated on a glossy paper under a custom viewing condition with the relative colorimetric rendering intent, where the viewing conditions are specified in the PCC. The iccCAM transformation can be used to convert this colour from the D50 to the custom viewing condition. The colour is then interpolated with the associated device output profile to obtain its device value.

Annex E
(informative)

Sparse matrix operations

The following discussion provides pseudo code to describe the addition of two sparse matrices as well as the multiplication of a vector by a sparse matrix. In each case a sparse matrix is represented using the following general structure:

```
Structure SparseMatrix {
  rows
  columns
  entries
  offset[rows+1]
  index[entries]
  data[entries]
}
```

NOTE 1 The first element in each array in the following pseudo code is assumed to be zero (0).

NOTE 2 In the above structure offset[rows] have the same value as entries

NOTE 3 The max_entries value used below is determined using the formula to determine the max number of entries given the number of bytes available to encode the sparse matrix (S), the number of rows in the sparse matrix (R), and the number of bytes used to encode each matrix entry (see Sparse Matrix Encoding).

Addition of sparse matrices

The following pseudo code performs the addition of the matrix operation in Formula (E.1).

$$C = a\,A + b\,B \tag{E.1}$$

where

A, B, C are sparse matrices;

a, b are scalar values.

```
if (not ( (A.rows equals B.rows) and (A.columns equals B.columns)))
 Return Failure;
endif

C.rows = A.rows
C.columns = A.columns
```

```
pos=0;
row = 0;
while (row < A.rows)
 nA = A.offset[row+1] – A.offset[row]
 nB = B.offset[row+1] – B.offset[row]

 C.index[row] = pos;
 if (nA!=0 and nB != 0)
  i=0;
  j=0;
  while (i<nA or j<nB)
   if (pos >= max_entry)
    return failure;
   endif

   if (i<nA and j<nB)
    if (A.index[A.offset[row]+i] < B.index[B.offset[row]+j])
     C.index[pos] = A.index[A.offset[row]+i];
     C.data[pos] = a * A.data[A.offset[row]+i];
     pos = pos +1;
     i = i+1;
    elseif (B.index[B.offset[row]+ j] < A.index[A.offset[row] + i])
     C.index[pos] = B.index[B.offset[row] + j];
     C.data[pos] = b * B.data[B.offset[row] + j];
     pos = pos +1;
     j = j+1;
    else
     C.index[pos] = A.index[A.offset[row]+i];
     C.data[pos] = a * A.data[A.offset[row] + i] +
              b * B.data[B.offset[row] + j];
     pos = pos + 1;
     i = i + 1;
     j = j + 1;
```

```
      endif
    elseif (i<nA)
      C.index[pos] = A.index[A.offset[row] + i];
      C.data[pos] = a * A.data[A.offset[row] + i];
      pos = pos + 1;
      i = i + 1;
    else
      C.index[pos] = B.index[B.offset[row] + j];
      C.data[pos] = b * B.data[B.offset[row] + j];
      oos = pos + 1;
      j = j + 1;
    endif
  endwhile
 elseif (nB equals 0)
  if (pos+nA >= max_entry)
    return failure;
  endif

  for i=0 to nA-1
    C.index[pos] = A.index[A.offset[row]+i];
    C.data[pos] = a * A.index[A.offset[row]+i];
    pos = pos + 1
  endfor
 elseif (nA equals 0)
  if (pos+nB >= max_entry)
    return failure;
  endif

  for i=0 to nB-1
    C.index[pos] = B.index[B.offset[row]+i];
    C.data[pos] = b * B.index[B.offset[row]+i];
  endfor
 endif
```

```
 row = row + 1;
endwhile

C.entries = pos;
C.index[C.rows] = pos;

Return C;
```

Multiplication of vector by sparse matrix

The pseudo code performs a multiplication of a vector *x* by a sparse matrix *M* [Formula (E.2)].

$$y = M\,x \tag{E.2}$$

Where *M* is a sparse matrix, *x* and *y* are vectors and the length of the vectors is the same as the number of columns in *M*.

```
j=0;
while (j < M.rows)
 pos = M.offset[j];
 nC = M.offset[j+1] – pos;

 y[j] = 0
 i = 0;
 while (i<nC)
  y[i] = y[i] + x[i] * M.data[M.index[pos + i]];
  i = i +1;
 endwhile

 j = j + 1;
endwhile
```


Annex F
(informative)

calculatorElement text representation and examples

F.1 Textual representation of calculator processing elements

The actual binary encoding of contents of operations in a calculatorElement is specified in Clause 11. However, when discussing the contents of a calculatorElement's main function, it is useful to use a text nomenclature for expressing the sequences of operations to perform. The following guidelines can be used for describing calculator element main function contents:

A sequence of operations is delineated by curly parenthesis { }.

EXAMPLE { … }

Numbers correspond to putting values on stack.

EXAMPLE 95 or pi

Operations are simply the text-based (without padded spaces) names of the operation.

EXAMPLE pow

Conditional operations with associated streams are represented using sequences of operations delineated by curly parenthesis { }.

EXAMPLE if { …} else { … }

CMM environment variables are identified by placing the 32-bit four-character environment variable signature (or 8-digit hexadecimal value) in parentheses after the env operator. CMM environment variable signatures with less than four characters are space padded.

EXAMPLE env(true) *is the same as* env(74727565)

EXAMPLE env(xx) *is the same as* env(78782020)

Stack operations, if not followed by a size specification (n) or by a size and repeat specification (n,r), assume only one stack element is involved. If a size specification is provided, at least two operations are needed.

EXAMPLE copy *or* copy(5) *or* copy(3,2)

Variable length operations if not followed by a size specification [n] assume only two operations. If specified, at least two operations are needed.

EXAMPLE and *or* and[5]

Requirements of channel vector operations are described in 11.2.1.

EXAMPLE tget[0] *or* tget[0,3]

F.2 Examples

The following are examples of defining calculator elements within a multiProcessElementTag.

F.2.1 Polynomial device modelling

The following discussion shows a basic example of device modelling, which can definitely be improved upon. The exact details of the model are not as important as understanding the processing element interaction in the approach.

Suppose that the colorimetric measurement of CMYK device output can be modelled using Formula (F.1) (note that more complicated functions are likely to be needed).

$$\begin{aligned} L^* &= a_{1,1} + a_{2,1}C + a_{3,1}M + a_{4,1}Y + a_{5,1}K + a_{6,1}CM + a_{7,1}CY + a_{8,1}CK + a_{9,1}MY + a_{10,1}MK + a_{11,1}YK \\ a^* &= a_{1,2} + a_{2,2}C + a_{3,2}M + a_{4,2}Y + a_{5,2}K + a_{6,2}CM + a_{7,2}CY + a_{8,2}CK + a_{9,2}MY + a_{10,2}MK + a_{11,2}YK \\ b^* &= a_{1,3} + a_{2,3}C + a_{3,3}M + a_{4,3}Y + a_{5,3}K + a_{6,3}CM + a_{7,3}CY + a_{8,3}CK + a_{9,3}MY + a_{10,3}MK + a_{11,3}YK \end{aligned} \tag{F.1}$$

This can be expressed as a matrix/vector equation as shown in Formula (F.2):

$$\begin{bmatrix} L^* \\ a^* \\ b^* \end{bmatrix} = \begin{bmatrix} a_{1,1} & \dots & a_{1,11} \\ a_{2,1} & & a_{2,11} \\ a_{3,1} & & a_{3,11} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ C \\ M \\ Y \\ K \\ CM \\ CY \\ CK \\ MY \\ MK \\ YK \end{bmatrix} \tag{F.2}$$

Expressing this as a calculator element can be done using matrix and vector operations by first placing the coefficients of the matrix followed by calculations of the vector on the stack as follows:

Assume C=in[0], M=in[1], Y=in[2], K=i[3], mat(0) = matrix as defined above

{

1,0 in[0,4]

in[0,2] mul

in[0] in[2] mul

in[0] in[3] mul

in[1,2] mul

in[1] in[3] mul

in[2,2] mul

mat(0) out[0,3]

}

Assume L=out[0], a=out[1], b=out[2]

F.3 RGBW display projector inverse model

The following example shows how a single Calculator Element can be used to encode the entire transformation from XYZ to RGB input values of the inverse model for an RGBW projector.

NOTE This is an implementation of the algorithm proposed by David Wyble and Mitchel Rosen in "Colorimetric Management of DLP™ Projectors", IS&T/SID Eleventh Color Imaging Conference, 228-232, (2004).

Assume X=in[0], Y=in[1], Z=in[2], mtx(0) = matrix used to estimate theoretical RGB values, mtx(1) = matrix used to estimate pre-LUT RGB values, curv(2)=white LUT with max red, curv(3)=white LUT with max green, curv(4)=white LUT with max blue, curv(5)=inverse RGB transfer curves. In the following example values in italics represent constants that would actually be used.

{
in[0,3]
$X_{DisplayWhite}$ $Y_{DisplayWhite}$ $Z_{DisplayWhite}$ mul[3]
0.9642 1.000 0.8249 div[3]

$X_{DisplayBlack}$ $Y_{DisplayBlack}$ $Z_{DisplayBlack}$ sub[3]
copy(3) tput[3,3] %requested normalized XYZ

mtx(0) copy(3) tput[0,3] %theoretical RGB values

max(3) 1.0 gt if {
tget[0,3] min(3)
copy tget[0] eq if {
curv(2)
} else {
copy tget[1] eq if {
curv(3)
} else {
curv(4)
}
}
neg copy(1,2)
($X_{DisplayWhite}$ - $X_{DisplayBlack}$)
($Y_{DisplayWhite}$ - $Y_{DisplayBlack}$)
($Z_{DisplayWhite}$ - $Z_{DisplayBlack}$)
mul[3]

tget[3,3] add[3]


```
   mtx(1)
  }
  curv(5) out[0,3]
}
```

F.4 CLUT interpolation using Lch addressing from an XYZ PCS example

The following example shows how a single Calculator Element can be used to encode the entire transformation from XYZ to Lab, and then to Lch and then through a sub-CLUT.

Assume X=in[0], Y=in[1], Z=in[2], clut[0] = colour lookup table to convert to CMYK output

{ in[0,3] 0.9642 1.000 0.8249 div[3] tLab ctop 100 182 360 div[3] clut[0] out[0,4] }

Assume C=out[0], M=out[1], Y=out[2], K=out[3]

NOTE This is only a textual representation of how the data are stored in the calculator element. The actual function operations are stored as an array of 8-byte operation definitions.

Annex G
(informative)

BRDF overview and description

G.1 Introduction

The light that comes from a surface to a viewer of the surface is defined by a complex function that is controlled by many factors. As a surface is viewed from different angles, or the angle of the light shining on the surface changes, the appearance of the object changes. Specular highlights can appear, and colours can change.

A bidirectional reflection distribution function (BRDF) describes the light that is reflected from an opaque surface. The inputs to the function are the direction of the incoming light relative to the surface normal and the direction of the viewer relative to the surface normal.

The output of the function is the ratio of reflected radiance to the irradiance incident on the surface. The function has the form shown in Formula (G.1):

$$f_{\mathrm{r}}\left(\omega_{\mathrm{i}},\omega_{\mathrm{r}}\right)=\frac{dL_{\mathrm{r}}\left(\omega_{\mathrm{r}}\right)}{L_{\mathrm{i}}\left(\omega_{\mathrm{i}}\right)\cos\theta_{\mathrm{i}}d\omega_{\mathrm{i}}} \tag{G.1}$$

where

L_{i} is incoming radiance;

L_{r} is reflected radiance;

ω_{i} is the unit vector that points to the position of the light;

ω_{r} is the unit vector that points to the position of the observer;

θ_{i} is the angle between ω_{i} and n.

The function can be visualized with the diagram shown in Figure G.1.

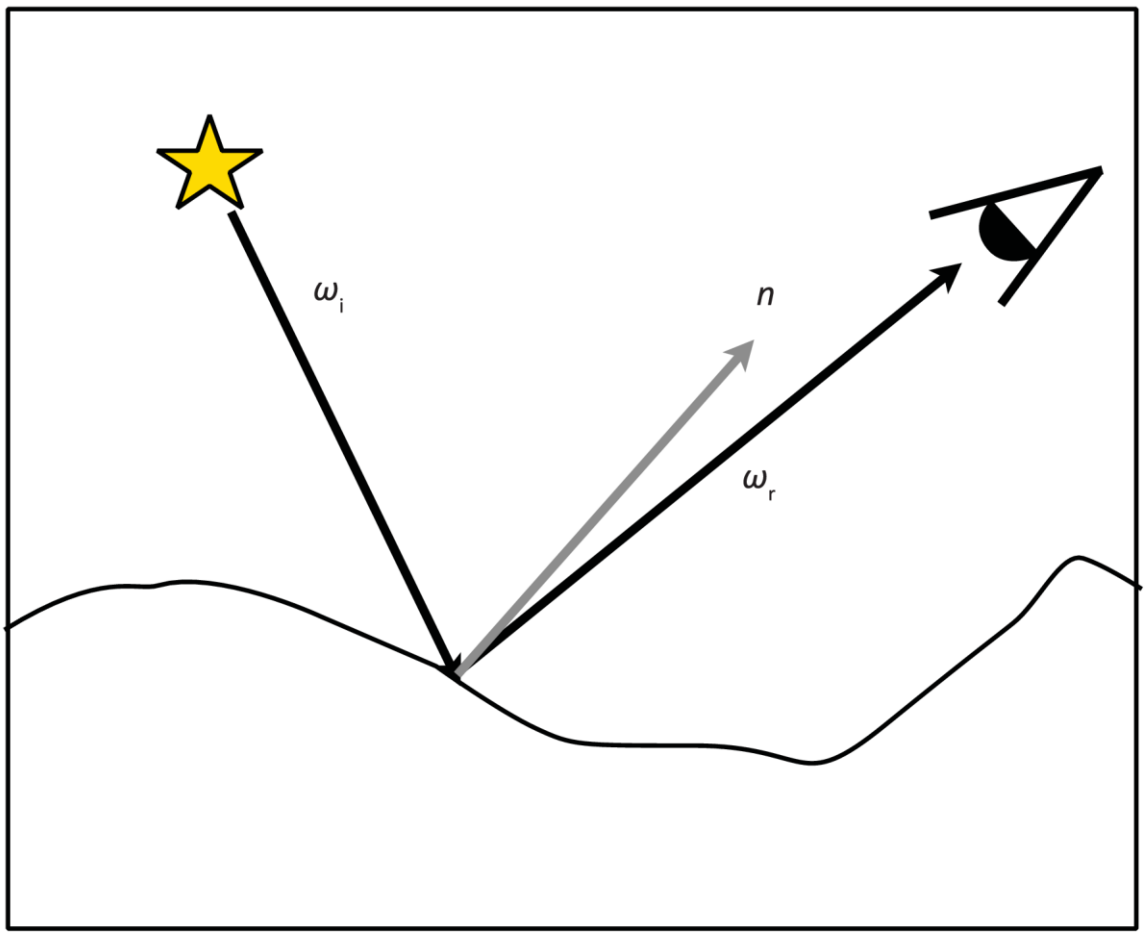


Figure G.1 — BRDF function

Both ω_i and ω_r are defined relative to the surface normal. Since both vectors are unit length, the BRDF function is four-dimensional.

Functions related to the BRDF function can also describe sub-surface scattering, shadowing, masking, and inter-reflections. These types of functions could be supported by the addition of new function types.

G.2 Purpose

The initial purpose for including BRDF support into ICC profiles is to allow a profile to provide the necessary information for defining the appearance of a surface for arbitrary lighting conditions and viewing positions.

For the most part ICC colour management assumes 0:45 geometry, and conversion between geometries is not defined. However, if a three-dimensional object within the context of a three-dimensional rendering system uses a BRDF colour defined by an ICC profile, the ICC profile can provide all the BRDF information for the rendering system to generate an appearance simulation of the object.

A baseline case allows a CMM to use 0:45 geometry with BRDF information to derive colour appearance information that is appropriate for connecting to an output profile.

G.3 The BRDFStruct element and the BRDFFunction element

Two types of BRDF tags are supported. The two types of tags supply data that neither alone can provide.

The BRDFStruct type tags (see brdfColorimetricParameter0Tag, brdfColorimetricParameter1Tag, brdfColorimetricParameter2Tag, brdfColorimetricParameter3Tag, brdfSpectralParameter0Tag, brdfSpectralParameter1Tag, brdfSpectralParameter2Tag, brdfSpectralParameter3Tag, brdfMToB0, brdfMToB1, brdfMToB2, brdfMToB3, brdfMToS0, brdfMToS1, brdfMToS2, and brdfMToS3) supply parameters for a BRDF model that can be used by 3D rendering software.

The BRDF Function tags (see brdfAToB0Tag, brdfAToB1Tag, brdfAToB2Tag, brdfAToB3, brdfDToB0Tag, brdfDToB1Tag, brdfDToB2Tag, and brdfDToB3Tag) provide PCS or spectralPCS values when given lighting direction, viewing angle, and device values. They can be implemented as a function or lookup tables within the multiProcessElementType. The BRDF Function tags can be more accurate than the results from using parameters with a BRDF model.

The reverse BRDF Function tags (see brdfBToA0Tag, brdfBToA1Tag, brdfBToA2Tag, brdfBToA3, brdfBToD0Tag, brdfBToD1Tag, brdfBToD2Tag, and brdfBToD3Tag) provide device values when given lighting direction, viewing angle, and PCS or spectralPCS values. They can be implemented as a function or lookup tables within the multiProcessElementType.

NOTE Obtaining BRDF model parameters for the purpose of 3D rendering from the BRDF Function type would require fitting the data from the transform to a BRDF model. The cost of performing this fitting would most likely be impractical for many use cases.

G.4 BRDF model support in ICC profiles with BRDFStruct

BRDF information can be expressed as a single function (monochrome BRDF) that is applied uniformly to all channels of the resulting colour, or separate BRDF information can be defined for each individual channel (chromatic BRDF) of the resulting colour. The first case (monochrome BRDF) allows for a simple and compact representation of surface appearance that in many cases will be sufficient. With the second case (chromatic BRDF) the characteristics of special effects pigments can be expressed.

G.5 Workflows

The following workflows are envisioned:

G.5.1 Normal non-BRDF

For a non-BRDF workflow a profile with BRDF tags functions is used just like a profile without BRFD tags, as illustrated in Figure G.2.

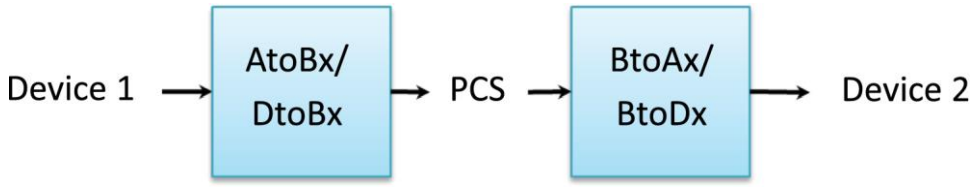


Figure G.2 — Non-BRDF workflow

G.5.2 Getting BRDF parameters from a profile with monochrome BRDF tags

In this workflow, illustrated in Figures G.3 and G.4, rendering parameters are provided by a profile with monochrome BRDFStruct tags. The full set of parameters is calculated by combining the output of the AToBxTag/DToBxTag/MToBxTag with the output of a brdfColorimetricParameterX (hereafter BCPx), brdfSpectralParameterX (BSPx) tag, brdfMToBx, or brdfMToSx tag (hereafter referred to collectively as BRDx tags). The parameters are combined as described in 12.2.1.

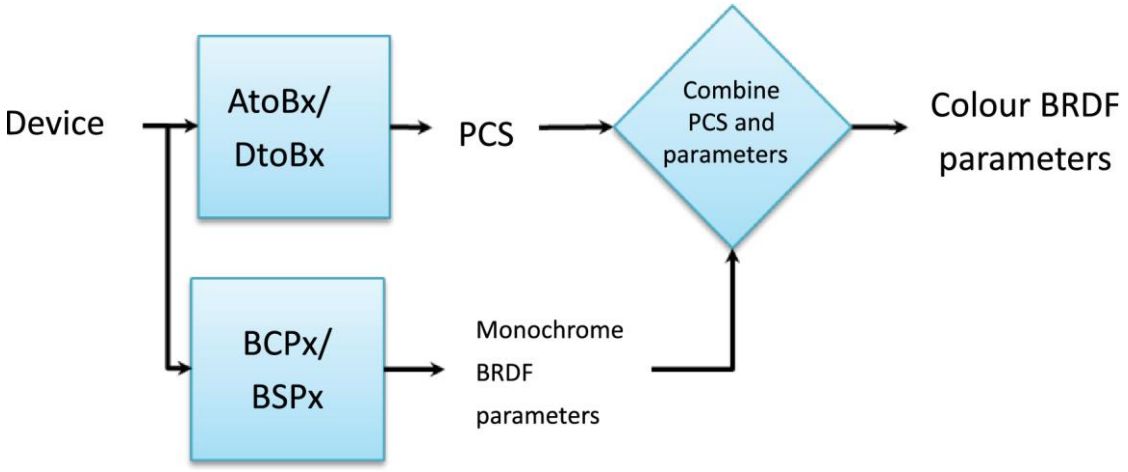


Figure G.3 — BRDF parameters from profile with monochrome BRDF tags

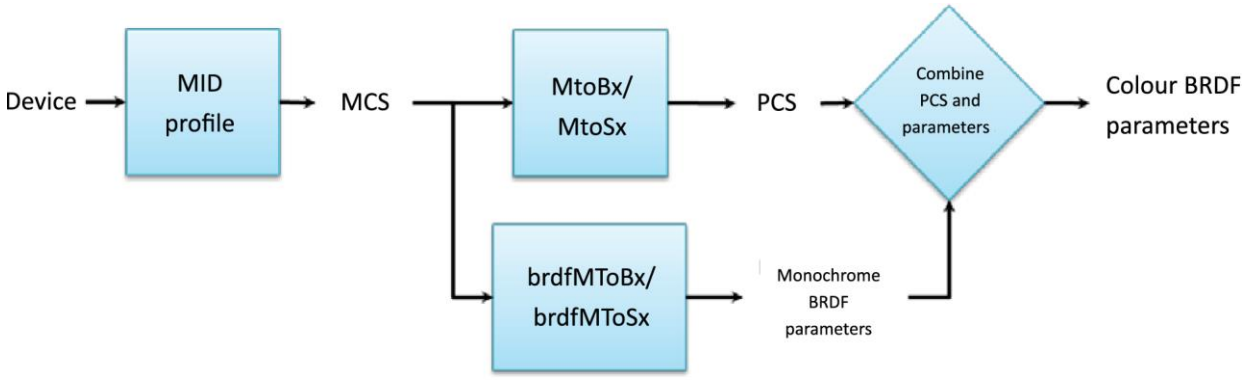


Figure G.4 — BRDF parameters from profile with monochrome BRDF tags

The following is an example of how to get colour BRDF parameters from a profile with monochrome Blinn-Phong BRDx tags:

Get colour BRDF parameters for device value of (1,0, 0,2, 0,7) from a colorimetric profile with a Blinn-Phong BRDFx Tag.

1) Pass device values through AToBx/DToBx to get PCS.

   a) the Tag returns an XYZ of (0,35,0,25,0,2);

   b) B = 0,35,0,25,0,2.

2) Get the monochrome BRDF parameters passing the device or multiplex channel values through the BRDx tag.

   a) the BRDx tag returns (0,5,0,51,0,49, 0,01,0,02,0,03, 0,3,0,31,0,29, 5,0,5,0,5,0);

   b) these returned values translate to:

      i) l_d = 0,5,0,51,0,49;

      ii) l_s = 0,01,0,02,0,03;

      iii) l_{gs} = 0,3,0,31,0,29;

iv) n = 5,0,5,0,5,0.

3) Combine the PCS values and the monochrome BRDF parameters to get k_d, k_s, and n.

a) $k_d = l_d B$;

b) k_d = 0,5*0,35,0,51*0,25,0,49*0,2;

c) k_d = 0,175,0,1275,0,098;

d) $k_s = l_s B + l_{gs}$;

e) k_s = 0,01*0,35+0,3,0,02*0,25+0,31,0,03*0,2+0,29;

f) k_s = 0,3035,0,315,0,302;

g) n = 5,0,5,0,5,0.

G.5.3 Getting BRDF parameters from a profile with chromatic BRDF tags

In this workflow, illustrated in Figure G.5, rendering parameters are provided by a profile with chromatic BRDFStruct tags. The full set of parameters is provided by the brdfColorimetricParameter0Tag, brdfColorimetricParameter1Tag, brdfColorimetricParameter2Tag, or brdfColorimetricParameter3Tag, (hereafter BCPx tags); or brdfSpectralParameter0Tag, brdfSpectralParameter1Tag, brdfSpectralParameter2Tag, or brdfSpectralParameter3Tag (hereafter BSPx tags).

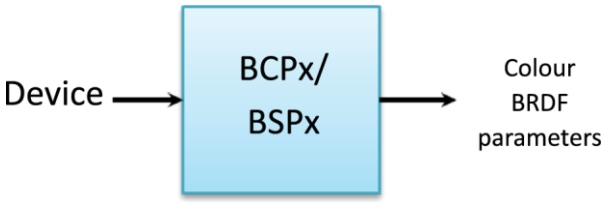


Figure G.5 — BRDF parameters from profile with chromatic BRDF tags

The following is an example of how to get colour BRDF parameters from a profile with chromatic Blinn-Phong BCPx/BSPx tags:

Get colour BRDF parameters for device value of (1,0, 0,2, 0,7) from a colorimetric profile with a Blinn-Phong BRDF Tag.

1) Pass device values through BCPx to get colour BDF parameters.

a) the BCPx tag returns (0,175,0,1275,0,098, 0,3035,0,315,0,302, 5,0,5,0,5,0);

b) k_d = 0,175,0,1275,0,098;

c) k_s = 0,3035,0,315,0,302;

d) n = 5,0,5,0,5,0.

G.5.4 Getting PCS values for a lighting position, viewing position, and device values from a BRDF Function element

The brdfAToBx/brdfDToBx BRDF Function tags allows PCS values to be obtained for a given illumination angle, viewing angle, and set of device values for the workflow illustrated in Figure G.6.

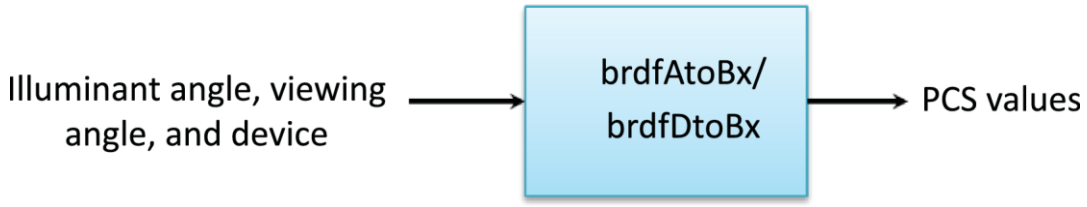


Figure G.6 — Obtaining PCS values from a BRDF function element

G.5.5 Getting device values for a lighting position, viewing position, and PCS values from a BRDF Function element

The brdfBToAx/brdfBToDx BRDF Function tags allow device values to be obtained for a given illumination angle, viewing angle, and set of device values for the workflow illustrated in Figure G.7.

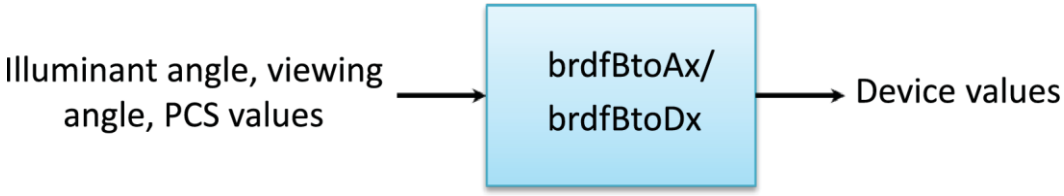


Figure G.7 — Obtaining PCS values from a BRDF function element

G.5.6 Getting PCS values for a lighting position, viewing position, and device values from a BRDF Structure element

The BRDF Structure element provides BRDF parameters for a specified BRDF model, as shown in Figure G.8. Acquiring PCS values from these parameters requires the implementation of the BRDF model using the the brdfColorimetricParameterX (BCPx) or brdfSpectralParameterX (BSPx) tags.

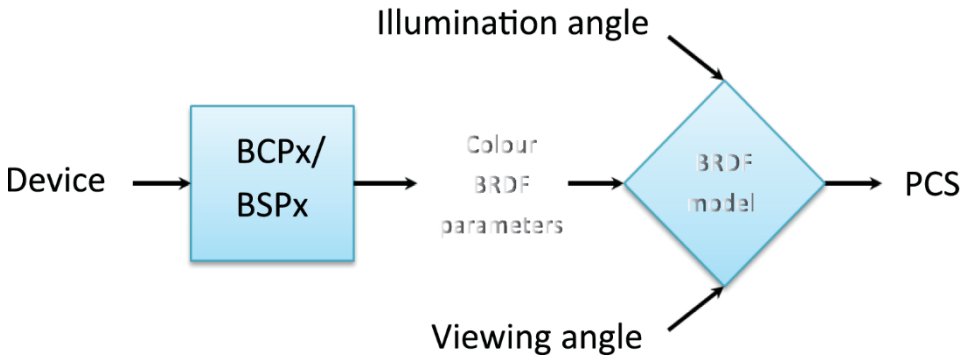


Figure G.8 — Obtaining PCS values from a BRDF Structure element

An example of the implementation of a BRDF model is given in G.5.7.

G.5.7 Obtain 45/0 PCS from profile that doesn't use 0:45 geometry and has BRDFStructure tags

For a profile that doesn't use 0:45 geometry but has BRDF tags, it is possible to get 0:45 PCS values, as illustrated in Figure G.9. If the profile supplies BRDF parameters for a BRDF model with a BRDFStructure tag, the PCS values are calculated by using the BRDF model with the brdfColorimetricParameterX (BCPx) or brdfSpectralParameterX (BSPx) tags.

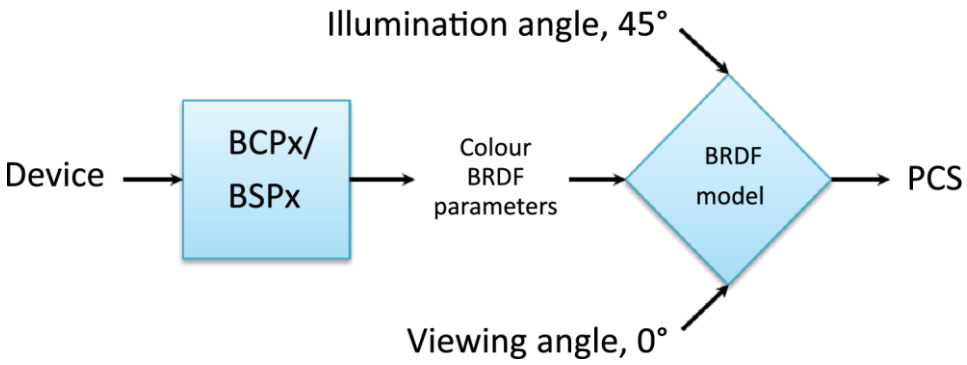


Figure G.9 — Obtaining 0:45 PCS values from a BRDF Structure tag

The following example shows how 0:45 colorimetric PCS values can be calculated from a profile with brdfColorimetricParameterX tags:

Get colour BRDF parameters for device value of (1,0, 0,2, 0,7) from a colorimetric profile with a Blinn-Phong BRDF Tag.

1) Pass device values through brdfColorimetricParameterX to get colour BDF parameters.

   a) the brdfColorimetricParameterX tag returns (0,175,0,1275,0,098, 0,3035,0,315,0,302, 5,0,5,0,5,0);

   b) k_d = 0,175,0,1275,0,098;

   c) k_s = 0,3035,0,315,0,302;

   d) n = 5,0,5,0,5,0.

2) Compute PCS values.

   a) light is at 45° and viewer is at 0°;

   b) L_m can be defined as (0,707,0,707,0);

   c) N (and the view direction) is (1,0,0);

   d) H_m is the half angle between Lm and the viewing direction. Hm is (0,9239,0,3827,0);

   e) the Blinn-Phong equation is $I_p = \sum_{m \in \text{lights}} \left(k_d \left(\hat{L}_m \cdot \hat{N} \right) i_{m,d} + k_s \left(\hat{N} \cdot \hat{H}_m \right)^n i_{m,s} \right)$;

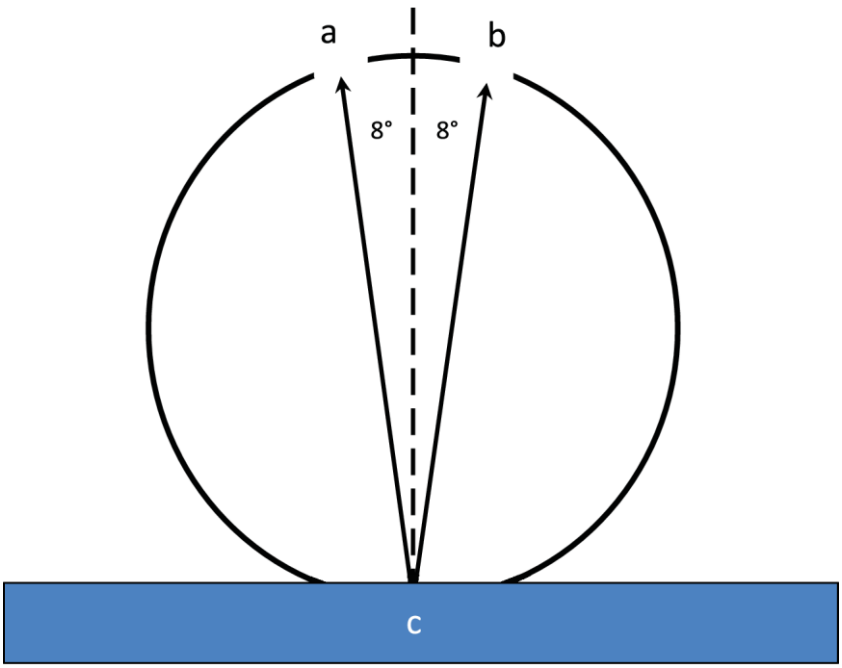
   f) the dot product $L_m \cdot N$ is 0,707;

   g) the dot product $N \cdot H_m$ is 0,9239;

   h) with $i_{m,d}$ and $i_{m,s}$ equal to 1,0, I_p = 0,3280,0,3022,0,2726.

G.5.8 Obtain Spherical PCS from profile that uses 0:45 geometry and has BRDFStructure tags

An instrument with spherical geometry uses a sphere with highly reflective white coating and a baffled light source located near the rear of the sphere. The measurement geometry of a typical 8° sphere instrument is illustrated in the simplified diagram shown in Figure G.10.



Key

a viewing port

b specular port

c sample

Figure G.10 — d:8 measurement using integrated sphere

The sample is viewed 8° from perpendicular and the associated specular is also at 8°. The specular port is located where the specular component impinges on the sphere. Opening the port allows the specular component to exit without being detected.

Acquiring spherical measurements by using the brdfColorimetricParameterX (BCPx) or brdfSpectralParameterX (BSPx) tags in a profile requires that an illumination map be applied to the BRDF model, as illustrated in Figure G.11.

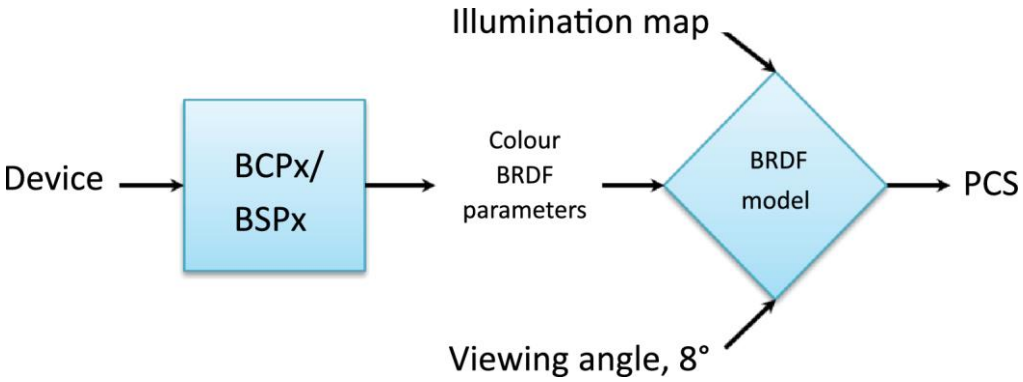


Figure G.11 — Obtaining PCS values using an illumination map

The illumination map is a fully illuminated sphere with a hole at the specular if the specular is not being included.

G.6 Rendering intent usage with BRDF data

The usage of rendering intents with BRDF data mirrors the usage of rendering intents without BRDF data. When the illumination and viewing angles match the profile's default (typically 0:45), the colour from the BRDF closely matches the colour of the non-BRDF transform for the corresponding rendering intent. For differing angles the colours are transformed in a similar manner, as illustrated in Figure G.12.

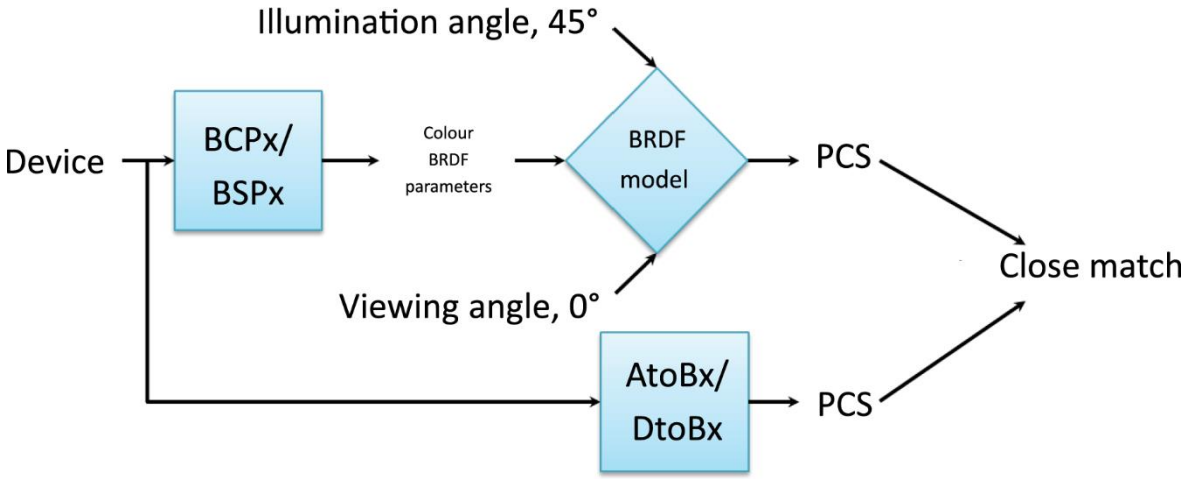


Figure G.12 — Matching PCS values using BRDF and non-BRDF intents

G.7 Normal map and height map usage with BRDF data

A BRDF doesn't provide spatial information about a surface. Information about the texture of surface can be included in a profile through the inclusion of a height map or a normal map. Both types of maps can be used to specify the small variations in surface height that are typical with many types of substrates.

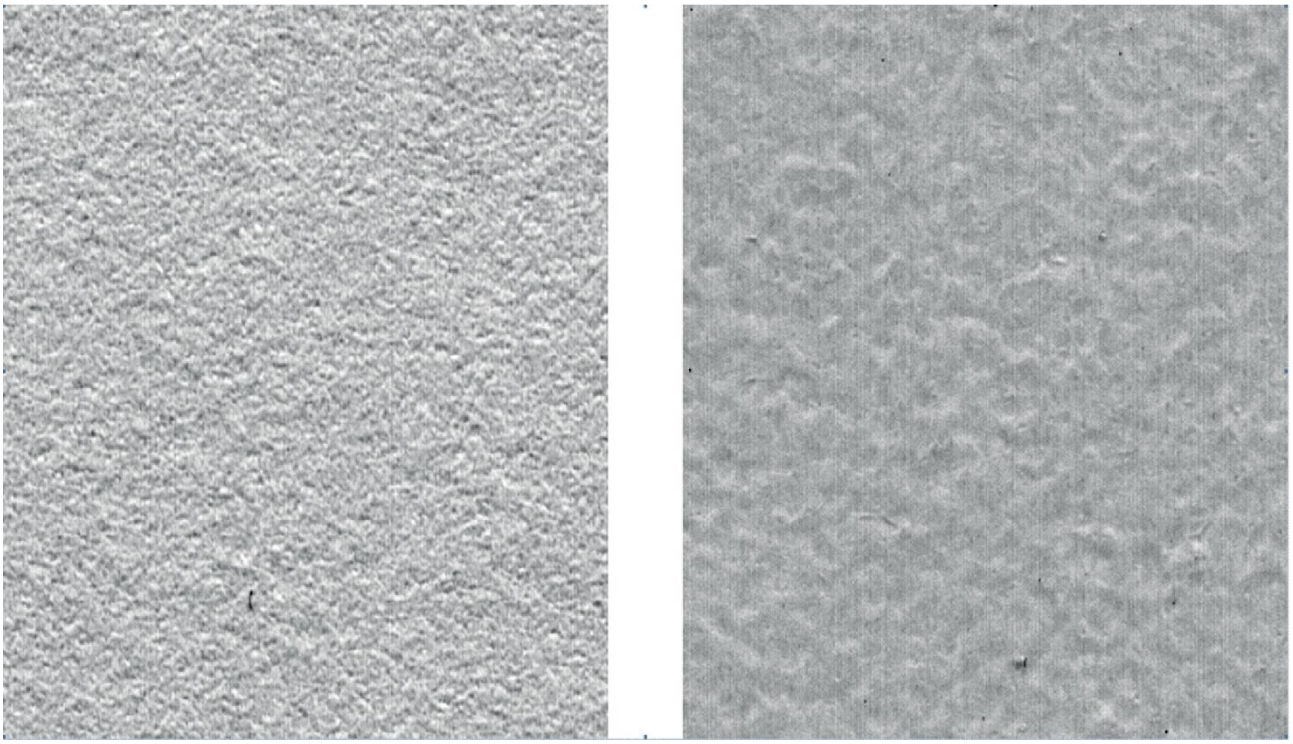


Figure G.13 — Magnified images of common papers with different surface characteristics.

Two examples of the surface variations that could be characterized by these maps are illustrated in the images shown in Figure G.13, where the left image is matte paper and the right image is a high quality semi-glossy paper.

A normal map is suitable for use in the 3D graphics technique of normal mapping. This technique enhances the appearance of detail without increasing the number of polygons. This technique doesn't alter the geometry, but provides a shader with a high-res map of surface orientation. This surface orientation is then used when the BRDF is applied. Normal mapping can have unrealistic results when the variations in surface height are large. For typical substrates the variation in height is small enough for this technique to appear realistic.

A height map can be used with the technique of displacement mapping. Displacement mapping alters the geometry in the scene and leads to correct outlines and shadows.

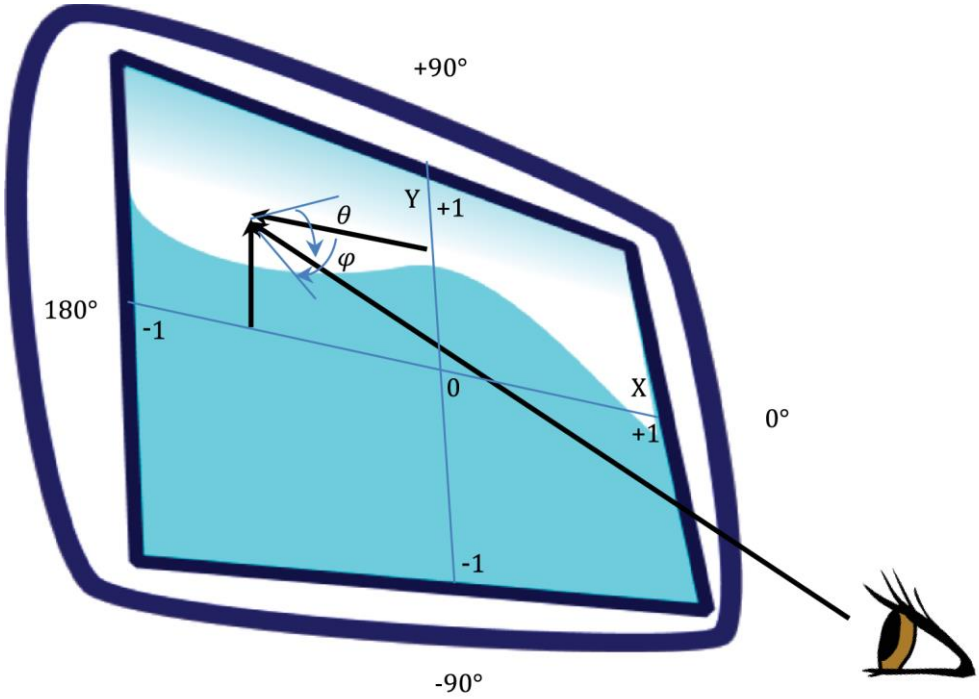
When a texture is used for 3D rendering it is typically not of high enough resolution to be applied across the entire surface. The texture is typically repeated across the surface so that the entire surface can be covered and the correct resolution can be maintained. When the texture is repeated across the surface the boundaries between the edges of the texture need to be seamless so that the edges between the borders of the texture are not visible.

It is possible to compute a height map from a normal map and a normal map from a height map using numerical techniques.

Annex H
(informative)

Directional emissive colour

The light emission experienced by an observer often changes as an observer views an emissive surface (display) viewed from different angles as well as focusing upon different positions in the field of view. An example is found in Figure H.1. The directional tags are implemented as multiProcessElelement tags that take viewing angle and relative position with either device or PCS values and return PCS or Device values associated with the expected observer experience.



Key

X horizontal position (-0.7)

Y vertical position (+0.6)

Φ azimuth angle (-70°)

θ zenith angle (50 °)

Figure H.1 — Viewing angles relative to display

The forward directional tags (directionalAToB0Tag, directionalAToB1Tag, directionalAToB2Tag, directionalAToB3Tag, directionalBToA0Tag, directionaDToB0Tag, directionaDToB1Tag, directionaDToB2Tag, directionaDToB3Tag) provide a means of describing the observed colour or spectral emission based upon viewing angle and relative position on the display.

The reverse directional tags (directionalBToA1Tag, directionalBToA2Tag, directionalBToA3Tag, directionalBToD0Tag, directionalBToD1Tag, directionalBToD2Tag, directionalBToD3Tag) provide a means of determining the device values needed to obtained an observed colour or spectral emission based upon viewing angle and relative position on the display.

Annex I
(informative)

Multiplex connection spaces

I.1 Introduction

A foundational principle of ICC-based colour management has been the use of a PCS to connect profiles. A PCS provides the answer to the question "What does the colour look like?" when defined colorimetrically, or "What relationship does the colour have to light?" when defined spectrally.

An alternative method of connecting profiles can be said to have always existed, but has seldom been concretely identified. When two profiles have the same signature in the device colour space field of their profile headers and their assumptions about device channel encoding are the same, then such profiles can be connected together by passing the device channel results from one profile as input data to the next profile. This can potentially be done by either the CMM directly or by an application making successive calls to a CMM that does not support device channel connection. When profiles are connected in this manner a kind of "Device Connection Space" is formed that answers the questions "What is the recipe for the colour?" or just simply "What is it?" with the implication that the meaning of the colour encoding is only defined by the device channel values provided and used by the profiles involved.

One limitation of using such a "Device Connection Space" is that both the number of channels and order of the channels need to be the same for both profiles. This is because there is no implicit processing associated with "Device Connection Space" channel connection. For each and every channel in the connection space, data from the i^{th} channel of one profile is passed directly to the i^{th} channel of the next profile without any modification of the channel data value.

The concept of a "Multiplex Connection Space" (MCS) as defined by this document-1 essentially extends the concept of a "Device Connection Space" to allow for flexibility in both the number and order of the channels while maintaining the same "What is it?" concept of the channel data encoding.

MCS channel connection flexibility enables various ICC based workflows that would be difficult if not impossible to otherwise implement.

No processing of the actual data is performed when MCS channel data are passed from one profile to the next profile across the MCS connection. The only processing that is performed is routing and initialization of "connected" Multiplex channels. MCS channels have no processing relationship to PCS channels as far as a CMM is concerned (in the same manner that there is no processing relationship between device channels and PCS channels). Any processing that is done between MCS and PCS connections is provided by transformation tags within the ICC profiles being used.

I.2 MCS connection basics

MCS usage is identified by the MCS field of the profile header, which contains a signature indicating the number of channels participating in the MCS.

The flags field in the profile header has also been extended to indicate that the MCS channels defined by the profile need to form a proper subset of the MCS channels defined in the profile that is being connected to. This logic is better understood relative to the example of connecting a MultiplexIdentification (MID) profile to a MultiplexVisualization (MVIS) profile using MCS connection. Four possible use cases of MCS subset flag combinations exist.

i) If the neither the MID profile nor the MVIS profile have the MCS subset flag set then there are no subset requirements.

ii) If only the MID profile has the MCS subset flag set then all of the MCS channels in the MID profile need to be present in the MVIS profile (i.e. the MID MCS channels need to be a subset of the MVIS MCS channels).

iii) If only the MVIS profile has the MCS subset flag set then all of the MCS channels in the MVIS profile need to be present in the MID profile (i.e. the MVIS MCS channels need to be a subset of the MID MCS channels).

iv) If both the MID and MVIS profiles have the MCS subset flag then both profiles need to contain the same set of MCS channels (though order of the MCS channels can vary).

MCS connection requires that these flags, along with the multiplexTypeArrayTags, be used to determine whether the MCS channels meet the subset requirements of both the profiles.

The multiplexTypeArrayTag defines names for each of the channels in the MCS. MCS channel data routing is based upon matching channel names in the profiles being connected.

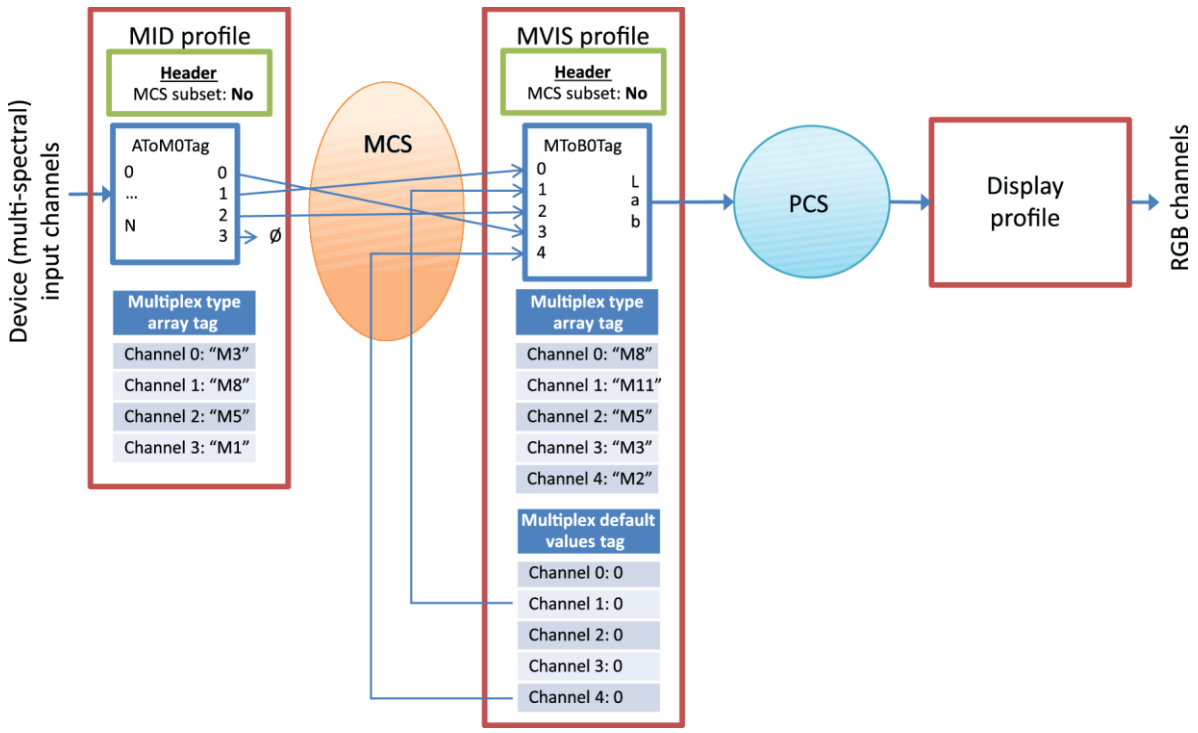
The optional multiplexDefaultValuesTag provides default values for channels that are not present in a source profile.

Four classes of profile are able to participate in MCS connection. Two can be used as input and two can be used as output. Input class profiles and MultiplexIdentification class profiles can provide AToM0 tags that transform device channel data to MCS channel data; MultiplexLink class profiles provide MToA0 tags that transform MCS channel data to device channel data; and MultiplexVisualization class profiles provide either MToS0 (spectral) or MToB0 (colorimetric) tags to provide transforms that go from MCS channel data to PCS channel data (spectral or colorimetric).

When MCS connection routing is determined, channels with identical names are “connected”. Thus, for each MCS channel of the source profile data are passed directly to an MCS channel of the destination profile that has an identical name. Source MCS Channel data are not used when the destination profile does not have an MCS channel with an identical name. Destination MCS channels are passed default values (either zero or from the multiplexDefaultValuesTag) when the source MCS does not have an MCS channel with an identical name.

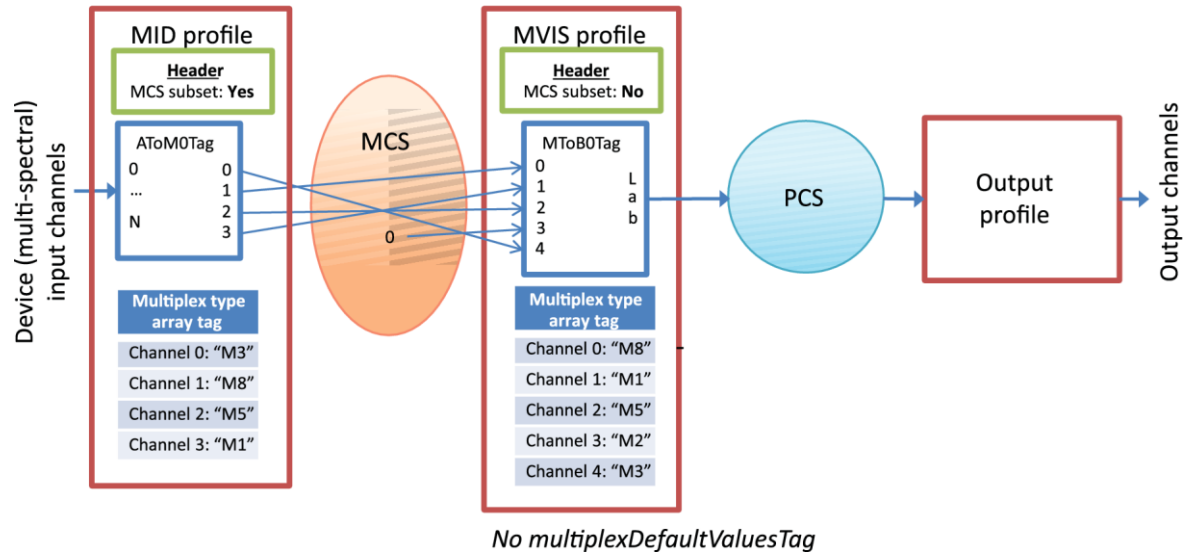
I.3 MCS connection examples

Examples of a few possible MCS workflow connections are depicted in Figures I.1 to I.5 (not to be considered an exhaustive set).



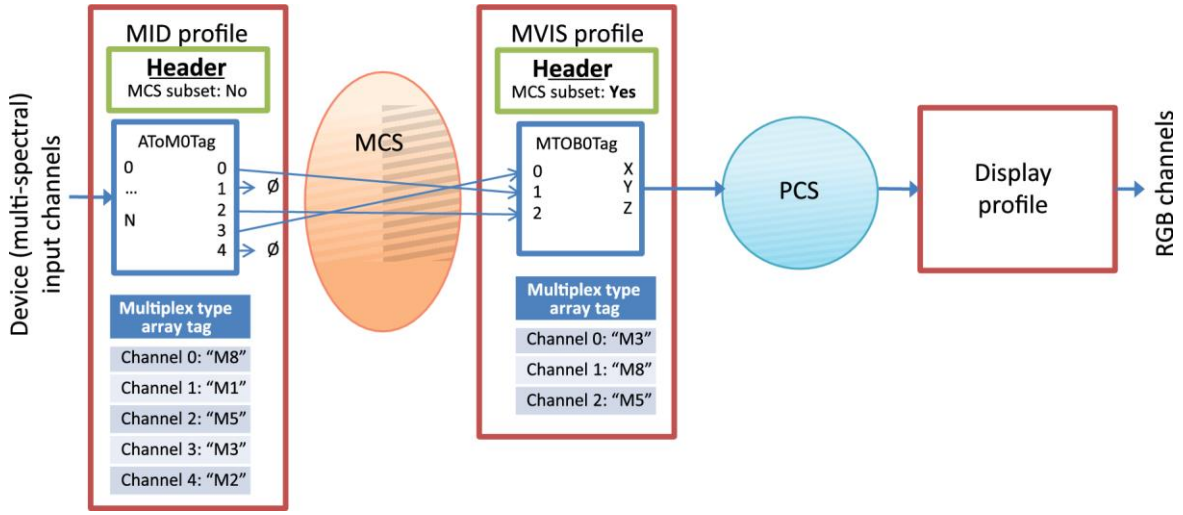
NOTE This connection would be expected to fail if the MCS subset requirement was enabled for either the MID or MVIS profile due to the fact that their multiplex channels are not subsets of one another.

Figure I.1 — Workflow with NO multiplex channel subset requirements



NOTE Value of zero is passed into MVIS channel index 3 because no multiplexDefaultValuesTag is defined in the MVIS profile.

Figure I.2 — Workflow with requirement that multiplex channels of MID profile are a subset of the multiplex channels of the MVIS profile



NOTE Data from MID profile's channels indexed 1 and 4 are ignored in the connection.

Figure I.3 — Workflow with requirement that multiplex channels of the MVIS profile are a subset of the multiplex channels of the MID profile

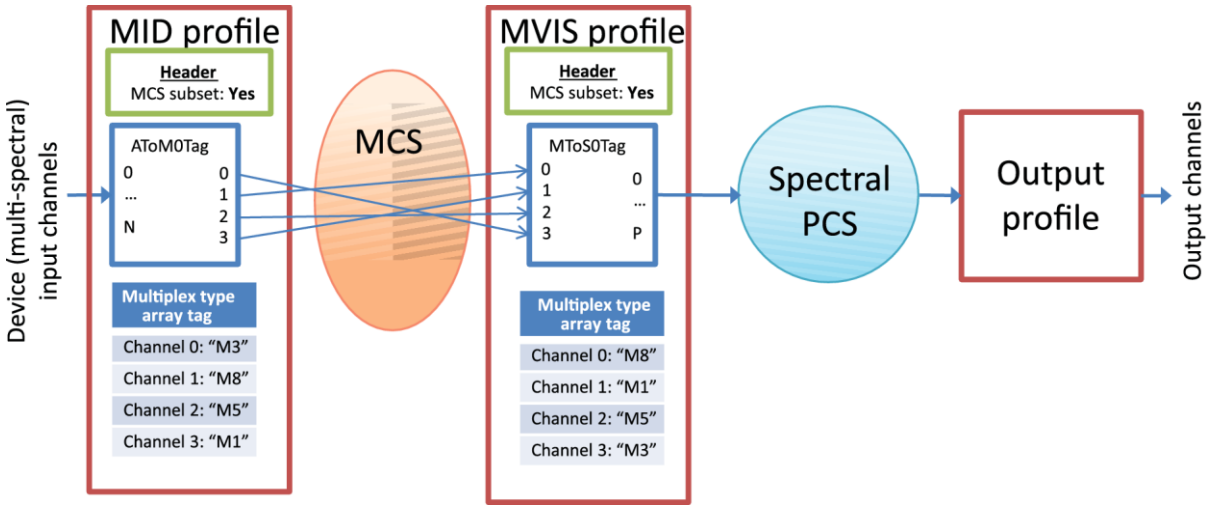


Figure I.4 — Workflow where MID and MVIS profiles have equivalent multiplex channels

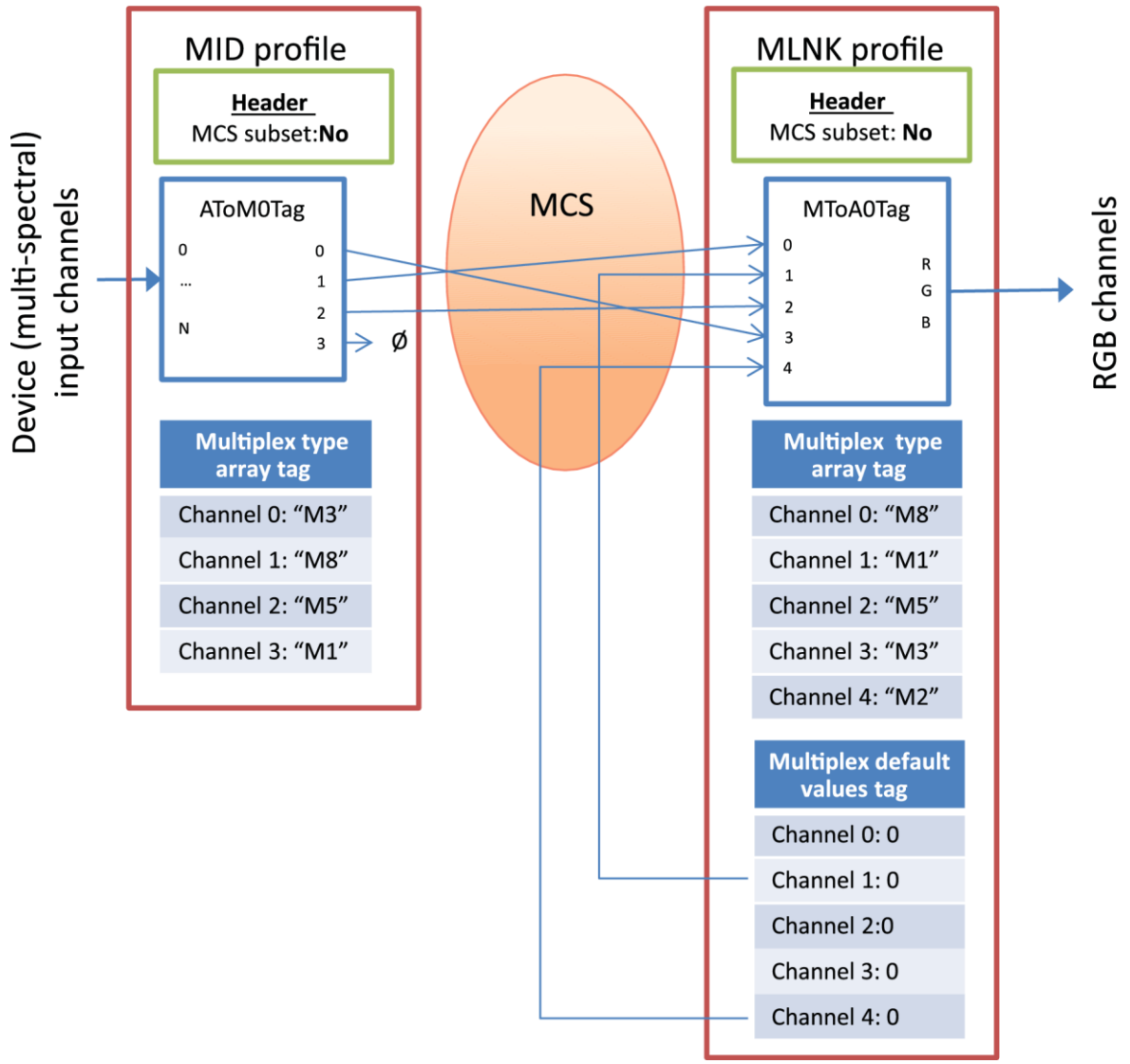


Figure I.5 — Workflow with NO multiplex channel subset requirements using a MultiplexLink profile

Annex J
(informative)

ColorEncodingSpace profiles

ColorEncodingSpace profiles allow for profile files that have minimum data structure which can be embedded in images with clear, concise, and non-redundant (canonical) information relative to a "named" reference that is provided to the CMM for determining the actual transforms to apply. Because the actual transformation is not defined by the profile, the CMM is responsible for determining/defining the transform to use.

Various methods can be employed by a CMM to provide the appropriate transform.

One method of implementing support for ColorEncodingSpace profiles within a CMM is to utilize a repository of either fully specified ColorEncodingSpace profiles (described below) that provide the information to establish the needed transforms, or profiles of other classes (e.g. display, colorspace) that also contain a referenceNameTag that associates the profile with a named colour encoding space.

When a ColorEncodingSpace profile is presented to the CMM, the CMM could look for a profile in its repository that has a matching colour encoding space name in the referenceNameTag of the matching profile (or colorSpaceNameTag if the referenceNameTag is set to "ISO 22028-1"). If the matching profile is not a ColorEncodingSpace profile then the profile can directly be used in place of the presented profile, and any optional parameters in the presented profile will be ignored.

If the matching repository profile is a fully specified ColorEncodingSpace profile then the colorSpaceNameTag of the matching repository profile will have the same text as the profile presented to the CMM, and the colorEncodingParamsTag of the matching repository profile will define all of the parameters for the colour endocing space (as defined by ISO 22028-1)[2]. These parameters can then be used (along with any overrides provided by the presented profile's colorEncodingParamsTag) to dynamicly create a temporary profile to be used to perform colour transforms in place of the presented profile.

The information in the colorEncodingParamsTag represents (to some degree) a logical replacement of the Matrix/TRC architecture defined in ISO 15061-1 with the addition of viewing condition information that can be used along with a colour appearance model to correct for differences in viewing conditions.

Therefore, a temporary profile can be created using a transform multiProcessElementsTag containg a curve set element and matrix element populated with information from the colorEncodingsParamsTag data with appropriate PCC tags that utilize a matrix element and/or colour appearance elements to adjust for differences in viewing conditions.

The temporary profile might be deleted when it is no longer needed to perform colour transformations.

Annex K
(informative)

Workflow scenarios and CMM processing control options

K.1 Introduction

Multiple workflow scenarios are enabled when profiles are used that contain more than one tag with encoded transforms. Additionally, multiple workflow scenarios are possible when a colour management module (CMM) implements alternative processing operations with existing transforms in profiles. Unambiguous operation is accomplished through CMM processing controls selected using Application Programing Interfaces (APIs) that uniquely select and configure particular workflow scenarios.

Specific workflow scenarios are not defined by this document. Rather, they are defined in ICS documents (separate from this document) which outline explicit profile encoding requirements as well as CMM processing controls related to each workflow scenario that are specified by the ICS.

Some processing control options (and associated workflow scenarios) are implied when using a profile that contains a single transform or transform type. Other processing control options are implicitly selected when a CMM has limited implementation support for transform types or processing options. Otherwise explicit CMM processing control options are used to select and achieve unambiguous operation. For some workflow scenarios multiple processing control options are combined to uniquely select and control the operation of the workflow scenario.

Several example CMM processing control options are described in K.2.

K.2 Example CMM processing control options

K.2.1 Rendering intent selection and processing

Rendering intent selection is a basic CMM processing control option associated with both this document and ISO 15067. Rendering intent selection is associated with both tag selection (see 8.14) and PCS processing (see Annex A). Rendering intent selection is used in most cases in conjunction with transform PCS selection (see K.2.4) and transform type selection (see K.2.5).

K.2.2 Application of Black Point Compensation (BPC)

Black Point Compensation (BPC) involves an additional PCS processing step in conjunction with colorimetric PCS processing (see K.2.4). The operations involved with BPC processing are specified by ISO 18619.

K.2.3 Forward/Reverse transform selection

A forward transform converts from a device, MCS, or colour space encoding to a PCS. A reverse transform converts from a PCS to a device, MCS or colour space encoding. Forward/Reverse transform selection is often implicitly determined by the position of the profile in the sequence of profiles being applied by the CMM (possibly in conjunction with the colour spaces being used). In other cases there may not be a forward/reverse transform associated with the transform type in which case forward/reverse transform selection is implicitly determined.

K.2.4 Transform PCS selection

Transform PCS selection involves determining the set of tags associated with a transform type that work with either a colormetric-based PCS or a spectrally based PCS. Transform PCS selection is generally also associated with rendering intent selection (see K.2.1) in addition to transform type selection (see K.2.4).

Colorimetric PCS-based transforms use the colorimetric PCS defined by the PCS field in the profile header along with other PCC metadata. Spectral PCS-based transforms use the the spectralPCS, spectralRange, and biSpectralRange fields in the profile header along with other PCC metadata.

NOTE This CMM processing control selection will be implicitly determined when connecting with a profile where only one PCS type is present.

Annex A provides information about PCS transformations that are performed when connecting forward transforms of one PCS type to reverse transforms of a different PCS type.

K.2.5 Transform type selection

K.2.5.1 General

Transform type selection is used to select the set of transforms that have a common processing purpose. K.2.5.2 provides a list of transform processing types associated with tags in this document.

In some cases transform type selection only provides a hint as the actual transform to use is directly implied by the fact that only one transform type is present in a profile.

K.2.5.2 Specic transform processing types

K.2.5.2.1 Colour transform processing

The most common form of transform used in ICC profiles is the colour transform. Colour transform processing usually involves transforming between a device or colour space encoding and a colorimetric or spectrally based PCS (in either direction) with transform PCS selection (see K.2.4) providing the means of selecting which type of PCS is involved and forward/reverse transform selection (see K.2.3) used to select which direction.

Two exceptions to this basic use of colour transform are used by Abstract and DeviceLink profiles. Abstract profiles use colour transforms that convert between a common PCS type – either colorimetric or spectral depending on the transform PCS selection (see K.2.4). DeviceLink profiles use a special form of colour transform that substitutes the use of the destination device colour space signature for the colorimetric PCS (with no PCS processing being performed).

Relevant tags for forward colour transform processing using a colorimetric PCS include: AToB0Tag (see 9.2.1), AToB1Tag (see 9.2.2), AToB2Tag (see 9.2.3), and AToB3Tag (see 9.2.4).

Relevant tags for reverse colour transform processing using a colorimetric PCS include: BToA0Tag (see 9.2.38), BToA1Tag (see 9.2.39), BToA2Tag (see 9.2.40), and BToA3Tag (see 9.2.41).

Relavant tags for forward colour transform processing using the spectral PCS include: DToB0Tag (see 9.2.76), DToB1Tag (see 9.2.77), DToB2Tag (see 9.2.78), and DToB3Tag (see 9.2.79).

Relavant tags for reverse colour transform processing using the spectral PCS include: BToD0Tag (see 9.2.42), BToD1Tag (see 9.2.43), BToD2Tag (see 9.2.44), and BToD3Tag (see 9.2.45).

K.2.5.2.2 Direct BRDF processing

With direct BRDF processing the bidirectional reflectance distribution function (BRDF) is implemented directly in the profile transform and the CMM is used to to determine results for rendering purposes. The application of a BRDF involves providing the viewing and lighting angles in addition to device/colour space/PCS encoding. Transform PCS selection (see K.2.4) provides the means of selecting which type of

PCS is involved, and forward/reverse transform selection (see K.2.3) is used to select which direction. Forward direct BRDF processing transforms convert the combination of viewing angles, lighting angles, and device/colour space encoding to PCS values. Reverse direct BRDF processing transforms convert from viewing angles, lighting angles and PCS values to device/colour space encoding. Further discussion of BRDF processing can be found in Annex G.

Relevant tags for forward direct BRDF processing using a colorimetric PCS include: brdfAToB0Tag (see 9.2.14), brdfAToB1Tag (see 9.2.15), brdfAToB2Tag (see 9.2.16), and brdfAToB3Tag (see 9.2.17).

Relevant tags for forward direct BRDF processing using a spectral PCS include: brdfDToB0Tag (see 9.2.26), brdfDToB1Tag (see 9.2.27), brdfDToB2Tag (see 9.2.28), and brdfDToB3Tag (see 9.2.29).

Relevant tags for reverse direct BRDF processing using a colorimetric PCS include: brdfBToA0Tag (see 9.2.18), brdfBToA1Tag (see 9.2.19), brdfBToA2Tag (see 9.2.20), and brdfBToA3Tag (see 9.2.21).

Relevant tags for reverse direct BRDF processing using a spectral PCS include: brdfBToD0Tag (see 9.2.22), brdfBToD1Tag (see 9.2.23), brdfBToD2Tag (see 9.2.24), and brdfBToD3Tag (see 9.2.24).

K.2.5.2.3 Parametric-based BRDF processing

With parametric BRDF processing the BRDF the profile only provided paramaters to a function that is typically applied by separate 3D rendering system. A single set of BRDF parameters can be used to describe appearance for any viewing/illuminanting angle. Transform PCS selection (see K.2.4) provides the means of selecting which type of PCS is involved. Parameters can be applied to separate device/PCS values or paramaters can be defined for each PCS channel. Parametric-based BRDF processing transforms convert from device/colour space encoding to PCS values (a forward transform). Further discussion of BRDF processing can be found in Annex G.

Relevant tags for parametric BRDF processing using a colorimetric PCS include: brdfColorimetricParameter0Tag (see 9.2.6), brdfColorimetricParameter1Tag (see 9.2.7), brdfColorimetricParameter2Tag (see 9.2.8) and brdfColorimetricParameter3Tag (see 9.2.9).

Relevant tags for parametric BRDF processing using a spectral PCS include: brdfSpectralParameter0Tag (see 9.2.10), brdfSpectralParameter1Tag (see 9.2.11), brdfSpectralParameter2Tag (see 9.2.12), and brdfSpectralParameter3Tag (see 9.2.13).

K.2.5.2.4 MCS to parametric BRDF processing

The operation of MCS to parametric BRDF processing is identical to parametric-based BRDF processing (see K.2.5.2.3) with the exception that MCS channel data with associated channel routing (see Annex I) is used as the input to the transform instead of device/colour space encoding. Transform PCS selection (see K.2.4) provides the means of selecting which type of PCS is involved.

Further information about MCS handling can be found in Annex I.

Relevant tags for MCS to parametric BRDF processing using a colorimetric PCS include: brdfMToB0Tag (see 9.2.30), brdfMToB1Tag (see 9.2.31), brdfMToB2Tag (see 9.2.32), and brdfMToB3Tag (see 9.2.33).

Relevant tags for MCS to parametric BRDF processing using a spectral PCS include: brdfMToS0Tag (see 9.2.34), brdfMToS1Tag (see 9.2.35), brdfMToS2Tag (see 9.2.36), and brdfMToS3Tag (see 9.2.37).

K.2.5.2.5 Device to MCS processing

Device to MCS processing is solely associated with the AToM0Tag (see 9.2.5). The device to MCS processing transform selection is implicitly used with MultiplexLink profiles (see 8.12). Explicit CMM processing control options to select MCS to device are used for specific workflow scenarios (defined by an ICS) when multiple transform types are present in a profile (for an example see 8.3).

Further information about MCS handling can be found in Annex I.

K.2.5.2.6 MCS to device processing

MCS to device processing is solely associated with the MToA0Tag (see 9.2.90). The MCS to device processing transform selection is implicitly used with MultiplexIdentification profiles (see 8.11). Explicit CMM processing control options to select device to MCS processing are used for specific workflow scenarios (defined by an ICS) when multiple transform types are present in a profile.

Further information about MCS handling can be found in Annex I.

K.2.5.2.7 MCS to PCS processing

MCS to PCS processing involves transforming from MCS channels to a colorimetric or spectrally based PCS with transform PCS selection (see K.2.4) providing the means of selecting which type of PCS is involved. Further information about MCS handling can be found in Annex I.

Relevant tags for MCS to PCS processing using a colorimetric PCS include: MToB0Tag (see 9.2.91), MToB1Tag (see 9.2.92), MToB2Tag (see 9.2.93), and MToB3Tag (see 9.2.94).

Relevant tags for MCS to PCS processing using a spectral PCS include: MToS0Tag (see 9.2.95), MToS1Tag (see 9.2.96), MToS2Tag (see 9.2.97), and MToS3Tag (see 9.2.98).

K.2.5.2.8 Named colour processing

Named colour processing is solely associated with the namedColorTag (see 9.2.99). Named colour processing transform selection is implicitly used with NamedColor profiles (see 8.10). Explicit CMM processing control options to select named colour processing are used for specific workflow scenarios (defined by an ICS) when multiple transform types are present in a profile.

CMM processing control options are used in conjunction with named colour processing transforms to select transform direction (see K.2.3) between tints of named colours and device values or PCS values, or transform PCS selection to determing whether to use colorimetic or spectral information (see K.2.4). Addition CMM processing control options might involve selection of tint-based measurement over a background.

Named colour processing may involve using alternate CMM APIs from that used by colour transforms because tints of named colours are involved and names are provided as part of the transformation.

Further information about named colours can be found in Annex D.

K.2.5.2.9 Directional emissive processing

The application directional emissive processing involves providing the relative position on the display and viewing angles in addition to device/colour space encoding. Forward directional emissive processing transforms convert the combination of relative position, viewing angles, and device/colour space encoding to PCS values. Reverse directional emissive processing transforms convert from relative position, viewing angles, and PCS values to device/colour space encoding. Further directional emissive processing can be found in Annex H.

Relevant tags for forward directional emissive processing using a colorimetric PCS include: directionalAToB0Tag (see 9.2.60), directionalAToB1Tag (see 9.2.61), directionalAToB2Tag (see 9.2.62), and directionalAToB3Tag (see 9.2.63).

Relevant tags for forward directional emissive processing using a spectral PCS include: directionalDToB0Tag (see 9.2.72), directionalDToB1Tag (see 9.2.73), directionalDToB2Tag (see 9.2.74), and directionalDToB3Tag (see 9.2.75).

Relevant tags for reverse directional emissive processing using a colorimetric PCS include: directionalBToA0Tag (see 9.2.64), directionalBToA1Tag (see 9.2.65), directionalBToA2Tag (see 9.2.66), and directionalBToA3Tag (see 9.2.67).

Relevant tags for reverse directional emissive processing using a spectral PCS include: directionalBToD0Tag (see 9.2.68), directionalBToD1Tag (see 9.2.69), directionalBToD2Tag (see 9.2.70), and directionalBToD3Tag (see 9.2.71).

K.2.6 Alternate PCC

Profile Connection Conditions (PCC) information (see 6.3.2) is present in a profile when either a non-standard PCS is used or a spectral PCS is used. PCC information is used as part of PCS processing (see Annex A) in cases where spectral viewing condition information is applied or conversions from/to custom colorimetry is performed.

Alternate PCC information is used instead of the PCC information in the profile when alternate profile connection condition information is provided to a CMM as part of the processing control options used to configure the application of a profile.

K.2.7 CMM environment variable usage

CMM environment variables are accessed by the calculator element 'env' operator (see 11.2.1.4). The values for CMM environment variables are provided using CMM processing control options to define values for environment variables with given signatures. ICSs define the CMM environment variable signatures and value encoding for specific workflow scenarios.

Calculator element scripts in profiles check the status of applying the 'env' operator to determine whether the environment value has been provided and perform appropriate operations when a variable is not available.

CMM environment variables may not be available when processing a profile because either the CMM does not provide CMM environment variable passing support (implicitly defining no environment variables as part of CMM processing control options) or the calling application has not provided values for the variables using CMM processing control options for the CMM environment variables.

K.2.8 Calculator element 'solv' operator support

The support for implementing the calculator element 'solv' operator is optional (see 11.2.1.7) for some workflow scenarios based on ICS requirements. In such cases the profile calculator script either does not use the 'solv' calculator element operator or it checks the status of the operator and performs appropriate operations. A CMM processing control option is used to control whether the 'solv' operator processing should be enabled when a CMM provides support for the 'solv' operator. A CMM that does not provide 'solv' operator support implicitly disables this CMM processing control option.

Bibliography

[1] ISO 18619, *Image technology colour management — Black point compensation*

[2] ISO 22028-1, *Photography and graphic technology — Extended colour encodings for digital image storage, manipulation and interchange — Part 1: Architecture and requirements*

[3] ANSI CGATS TR 001:1995, *Graphic Technology — Color Characterization Data for Type 1 Printing*

[4] CIE 159:2004, *A colour appearance model for colour management systems: CIECAM02*, CIE Central Bureau, Vienna, Austria

[5] Li CJ, Luo MR, Testing the robustness of CIECAM02, *Color Res Appl* (30) (2005) 99-106.

[6] Süsstrunk S, R8-07, CAT in CIECAM02

[7] Süsstrunk S, Brill MH, The nesting instinct: Repairing non-nested gamuts in CIECAM02, late breaking-news paper, 14th SID/IS&T Color Imaging Conference Scottsdale, AZ, (2006)

[8] Brill, MH, Süsstrunk S, Repairing gamut problems in CIECAM02: A progress report, Color Res Appl 33 (2008), 424-426; and Erratum, p. 493

[9] DESHPANDE K. "N-colour separation methods for accurate reproduction of spot colours," Ph.D. dissertation, University of the Arts London, May 2015

[10] Green PJ, A test target for defining media gamut boundaries, *Proc. SPIE* 4300, (2001) 105-113

[11] CIE 015:2018, *Colorimetry 4th Edition*, CIE Central Bureau, Vienna, Austria

[12] Rivest RL, *RFC1321: The MD5 Message-Digest Algorithm*, 1992, https://tools.ietf.org/html/rfc1321

[13] Unicode, Inc., *Unicode Core Specification*, http://www.unicode.org/versions/latest/

[14] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0* (Fifth Edition), 2008, http://www.w3.org/TR/REC-xml