ICC Votable Proposal Submission

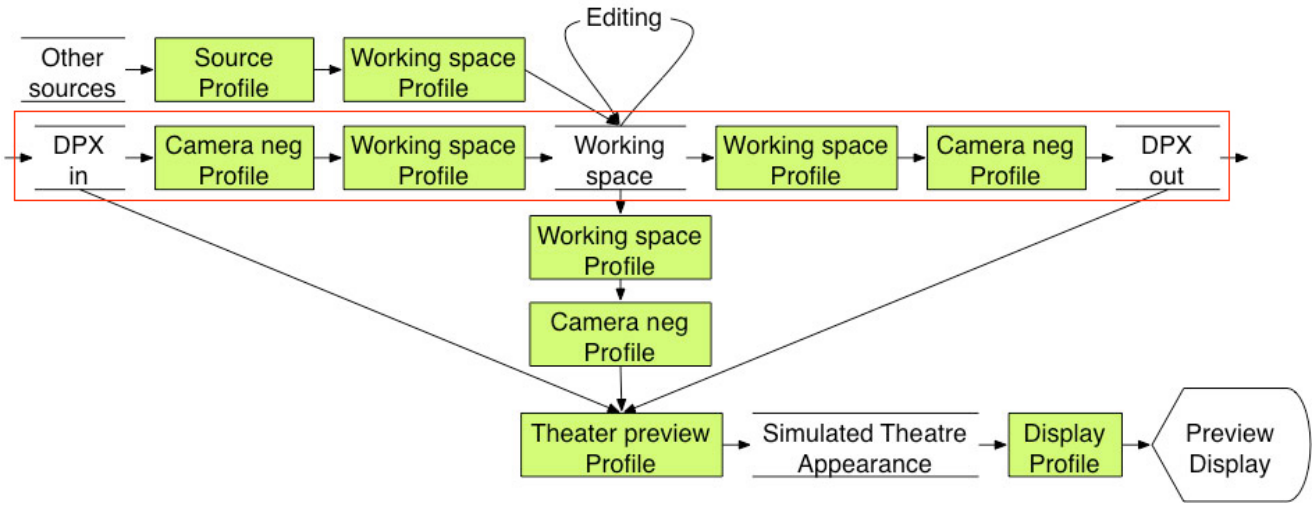
Floating-Point Device Encoding Range

Proposers: Manish Kulkarni, Adobe Systems; Max Derhak, ONYX Graphics
Date: June 16, 2006
Proposal Version: 1.1

1. Introduction

Digital Motion Picture Workflows require editing of DPX files (DPX files contain 10-bit Printing Density values obtained by scanning Original Camera Negative film). Special Effects (e.g. dinosaur breathing blue fire) are added to DPX files during the editing process. In order to make the Special Effects appear realistic, editing is performed in a Scene-Referred RGB space (strictly speaking, the space may be focal-plane-referred since flare and filtration may not be taken into account).

A typical DMP Editing workflow using ICC Profiles is as follows:



Of interest for this proposal is the path from DPX In to DPX Out, highlighted above in red.

The Camera Negative Profile enables the conversion of DPX values to Scene-Referred XYZ values. The Working Space Profile (which is RGB) enables the conversion of Scene-Referred XYZ values to Scene-Referred RGB values. Editing is performed in the Scene-Referred RGB space. The Working Space Profile and Camera Negative Profile enable the conversion of Working Space RGB values back to DPX for saving.

An example of the Camera Negative Profile is the Adobe “DPX Scene - Standard Camera Film” Profile. See http://www.color.org/membersonly/Notes_DPX_Scene-StandardCameraFilm.pdf and http://www.color.org/membersonly/DPXSceneStandard_ICC.icc.

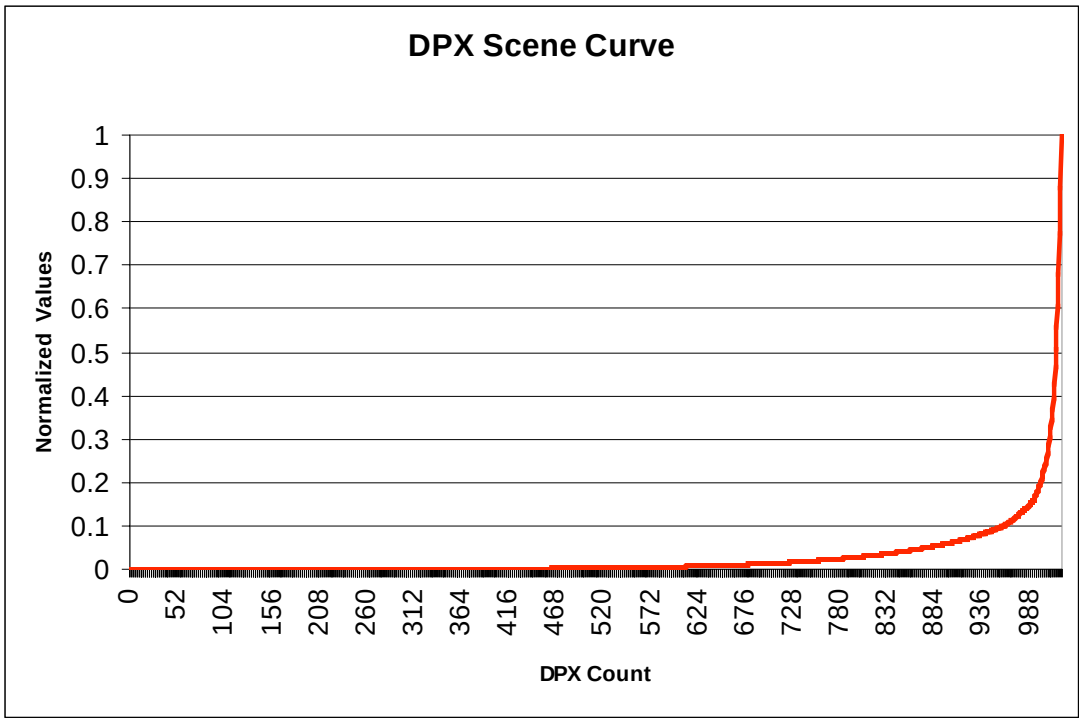
An example of the RGB Working Space is a Color Space based on the ITU-R BT.709.3 chromaticities and non-linearity.

1.1 Problem Statement

A key requirement for DPX Editing is that unedited content be unchanged in the workflow. It is not possible to satisfy this requirement with ICC v4.2.0 Profiles. The reasons are discussed below in detail.

1.1.1 Limited Precision in ICC Profiles

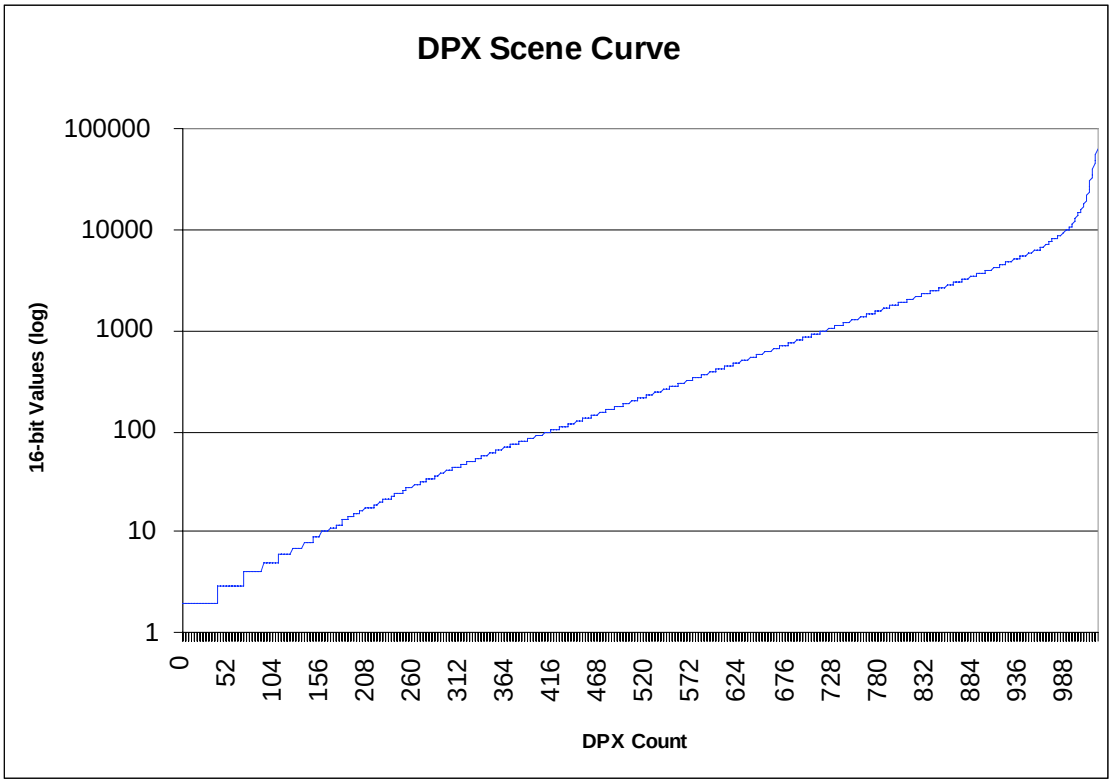
The Camera Negative Profile is used to convert DPX values to Scene XYZ values. During this conversion, a 1D Curve is applied to each input DPX value. The TRC is shown below:



Some of the values of the curve are shown in the table below:

10-bit DPX Count	Curve Values	Normalized Values	16-bit Values
0	0.001855	2.44594E-05	2
1	0.001876	2.47420E-05	2
2	0.001898	2.50279E-05	2
…	…	…	…
37	0.002837	3.74154E-05	2
38	0.002870	3.78477E-05	2
39	0.002903	3.82851E-05	3
40	0.002937	3.87274E-05	3
…	…	…	…
684	0.895955	0.0118158	774
685	0.902699	0.0119048	780
686	0.909493	0.0119944	786
…	…	…	…
1021	58.629983	0.7732119	50672
1022	66.676150	0.8793247	57627
1023	75.826544	1.0000000	65535

The curve in this example has 1024 points, one point corresponding to each input 10-bit DPX count. What must be encoded in the ICC Profile are the values in the “Normalized Values” column. Since ICC Profiles only support up to 16-bit precision, the values must be converted to 16-bit. It is obvious from the “16-bit values” column that doing so results in severe quantization. This quantization becomes obvious when the curve is graphed on a log scale, shown below:



One way to work around this problem is to apply a gamma-curve to the Normalized Values, and use two consecutive curves in the profile:

Curve 1: Sampled Curve with Applied Gamma, limited to 16-bits
Curve 2: Parametric Curve with Inverse Gamma

This method was used in the Adobe DPX Scene Profile. The model used in that case was:

Curve → Matrix

which was split up as:

Curve1 → Curve2 → Matrix

and encoded in the mAB-type A2B1 tag as:

A (Curve1) → CLUT (Identity) → M (Curve2) → Matrix → B (Identity)

A better model, however, for reconstructing scene colorimetry would be:

Matrix1 → Curve → Matrix2

This enables the modeling of film channel cross-talk.

One way to encode this in the mAB-type A2B1 tag is:

A (Identity) → 2x2x2 CLUT (Matrix1) → M (Curve) → Matrix (Matrix2) → B (Identity)

Now it is no longer possible to split up the Curve into Curve1 and Curve2 because there are insufficient processing elements.

Another problem related to precision in the ICC Profile is the precision of the Matrix element. In current ICC Profiles, the Matrix encoding is s15.16. It has been found that it is not possible to encode a Matrix and its Inverse Matrix with sufficient precision to allow DPX-count round-tripping. The inaccuracies resulting from the

encoding of the Inverse Matrix can cause round-trip errors of up to 11 DPX counts, which is clearly unacceptable.

One way to solve these problem is by having support for floating-point data in ICC Profiles.

1.1.2 Device-side encoding is bounded

In current ICC profiles the device encoding range must have definite bounds. The device encoding minimum is mapped to zero in the profile and the device encoding maximum, corresponding to the device maximum white, or in the case of a capture device the device saturation, is mapped to 1. The mapping of the measured colorimetry of the device white to 1 is accomplished using linear XYZ scaling. The media white point tag is then used to undo the scaling of the device white to produce "ICC-absolute" colorimetry values which are relative to the assumed adapted white.

The device encoding bounds do not limit the dynamic range that can be supported because there are no restrictions on the relation between the device encoding minimum and "zero photon" black, or between the device encoding maximum and the assumed adapted white. However, there is increasing interest in the use of floating-point color encodings, for exchange and as working spaces. Practically, such encodings can be thought of as unbounded, and it is therefore impossible to support them using current ICC profiles.

1.1.3 PCS encoding range is limited to [0,2)

The current ICC PCS encoding of XYZ values is limited to the range of [0,2). Since, in this encoding, the device encoding maximum is encoded as 1, it is unlikely that any device values will have corresponding D50 chromatically adapted XYZ values above 2. However, it is possible for this to occur in some unusual circumstances, for example in situations of extreme fluorescence where the media white is much darker than some saturated colors. The media white is defined to be the lightest neutral color that a capture device can capture, or an output device can produce.

It is also possible that some device values may have corresponding XYZ values that are negative. Such values can result from digital camera color analysis matrices, or chromatic adaptation transforms applied to extremely saturated blue colors. In most cases, it is acceptable to clip negative XYZ values to zero as such values do not correspond to real colors. However in some cases this may be unacceptable, for example if perfect round-tripping is desired.

1.2 Goals of the Solution

The primary goals of the solution are as follows:

- Solve the problems described in the previous section. This will enable the use of ICC Profiles in DMP workflows. This proposal introduces new features in ICC Profiles to solve the problems.
- Provide complete backwards compatibility with the existing ICC Profile Specification (Profile Version 4.2.0). This will enable us to make a Minor Revision to the current specification, and will have no impact on existing products and workflows that do not require the new features.

A secondary goal of the solution is to lay the groundwork for the support of additional workflows that are currently not possible with ICC Profiles.

1.3 Overview of the Solution

The proposal introduces a new set of optional Color Transform tags D2Bx and B2Dx. The letter “D” refers to Device Space (similar to the “A” space in the current A2Bx and B2Ax tags), and the letter “B” refers to the PCS (identical to the “B” space in the current A2Bx and B2Ax tags). The suffix “x” can be 0, 1, 2 or 3, which indicates the Rendering Intent, identical to the suffix in the current A2Bx and B2Ax tags, with the suffix ‘3’ representing the ICC-Absolute transform in the new tags. Profile Creators that include the D2Bx and B2Dx tags also need to include the A2Bx and B2Ax tags. The D2Bx and B2Dx transforms are intended to produce equivalent results to the corresponding A2Bx and B2Ax transforms, excepting the increased precision and extended range. CMMs that support floating-point encoding range can use the D2Bx and B2Dx tags instead of the A2Bx and B2Ax tags. Since the new tags are optional, CMMs are not required to support these tags.

The Device Encoding Range in the new Color Transform tags is not limited to the [0,1] range, but instead encompasses the values represented by 32-bit IEEE 754 floating-point numbers (excluding denormalised numbers, infinities and 'NaN' values). The Color Transform tags contain several processing elements which have a 32-bit IEEE 754 floating-point range and domain.

2. The acceptance of this proposal will result in a:

minor revision of Specification ICC.1:2004-10

3. Nature of the proposal:

This is a **new technical addition** to the specification.

New optional tags BToD0Tag, BToD1Tag, BToD2Tag, BToD3Tag, DToB0Tag, DToB1Tag, DToB2Tag and DToB3Tag are defined, that use a new tag type multiProcessElementsType.

4. Votable Proposal

- **Add a Normative Reference to Clause 3:**

  IEEE 754-1985, *Standard for Binary Floating-Point Arithmetic*

- **New Clause 5.x float32Number**

  Single-precision 32-bit IEEE 754 floating-point number, excluding denormalised numbers, infinities, and Not a Number 'NaN' values.

  NOTE 1: A 32-bit IEEE 754 floating-point number has an 8-bit exponent and a 23-bit mantissa.

  NOTE 2: Although denormalised numbers, infinities and NaN values are not stored in the ICC Profile, such values may occur as a result of CMM computations.

- **New Clause 5.x positionNumber**

  Positions of some data elements are indicated using a position offset with the data element's size. This data type allows this information to be stored as a single entity.

Table A – positionNumber Encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	Offset to Data Element	uInt32Number
4..7	4	Size of Data element in Bytes	uInt32Number

- **Replace Header 6 with**

  6 Profile Connection Space, Rendering Intents and Device Encoding

- **Replace Paragraph 2 of Clause 6.2.3 with the following:**

  The D2B3 and B2D3 tags contain separate transforms for the ICC-absolute colorimetric intent. This allows for an absolute expression of colorimetric data, limited only by the range of float32Number values. The mediaWhitePointTag is NOT used in processing D2B3 and B2D3 tags.

  When D2B3 and B2D3 tags are not present (or not used), profiles do not contain a separate transform for the ICC-absolute colorimetric intent. When this intent is needed, it shall be generated, as described in 6.3.2, using the mediaWhitePointTag, which specifies the CIE 1931 XYZ tristimulus values of the white point of the actual medium, as represented in the PCS. In this way, ICC-absolute colorimetric rendering may be obtained by using the media-relative colorimetric intent transformations (A2B1, B2A1) for the

source and destination profiles and scaling the PCS values by the ratio of the destination profile mediaWhitePointTag to the source profile mediaWhitePointTag (see Annex D for more information).

- **Replace Paragraph 1 of Clause 6.3.4.1 with the following:**

  The colorimetric data defined in the above clauses shall be specified either as CIEXYZ or CIELAB data. When specified as CIEXYZ data it shall be encoded using 16 bits/component while when specified as CIELAB data it shall be encoded as either 8 or 16 bits/component. Additionally, within the context of DToBx and BToDx tags CIEXYZ or CIELAB PCS data shall be encoded using float32Number values that directly express CIEXYZ or CIELAB PCS colorimetry.

- **Add the following to the end of Clause 6.3.4.1 (after Note 2):**

  NOTE 3 When converting from float32Number-based to integer-based encoding, component-wise clipping shall occur if the floating-point value is outside the range that can be encoded as integers.

- **Replace text in Clause 6.3.4.2 with the following:**

  For the float32Number-based PCS encodings the actual CIEXYZ or CIELAB values are directly encoded. For the integer based CIEXYZ encoding, each component (X, Y, and Z) is encoded as a u1Fixed15Number. The relationship between CIEXYZ encodings is shown in table in Table B.

Table B – CIEXYZ X, Y or Z encoding

Value(X, Y, or Z)	u1Fixed15Nubmer Encoding	float32Number Encoding
0,0	0000h	0,0
1,0	8000h	1,0
1,0 + (32767,0 / 32768,0)	FFFFh	1,999969482421875

NOTE: 1,999969482421875 is 1,0 + 32767,0 / 32768,0 expressed as a float32Number.

For the CIELAB PCS encodings, the L* values have a different encoding than the a* and b* values. The L* encoding is shown in table 8.

Table 8 – CIELAB L* encoding

Value(L*)	8-bit Encoding	16-bit Encoding	float32Number Encoding
0,0	00h	0000h	0,0
100,0	FFh	FFFFh	100,0

The a* and b* encoding is shown in table 9.

Table 9 – CIELAB a* or b* encoding

Value(a*or b*)	8-bit Encoding	16-bit Encoding	float32Number Encoding
-128,0	00h	0000h	-128,0
0,0	80h	8080h	0,0
127,0	FFh	FFFFh	127,0

NOTE 1 The integer encoding is not "two's complement" encoding, but a linear scaling after an offset of 128. This encoding was chosen to prevent discontinuities in CLUTs when going from negative to positive values.

NOTE 2 It is possible to convert between the 8-bit and 16-bit encodings by multiplying or dividing by 257. (See A.4.)

NOTE 3 Both the lut16Type and the namedColor2Type tag types (and ONLY those tag types) use a legacy 16 bit encoding of L*, a* and b* which is retained for backwards compatibility with an earlier profile version (version 2). To avoid confusion this encoding is specified in clause 10.8 "Lut16Type"..

- **Replace text in Clause 6.4 with the following text:**

  Conversions between the CIEXYZ and CIELAB encodings shall use the equations specified in CIE 15.2 (see A.3 in Annex A).

  When converting to integer-based encodings:

  Any colours in the PCS XYZ encoding range that are outside of the PCS LAB encoding range shall be clipped on a per-component basis to the outside limits of the range of PCS LAB when transforming from XYZ into LAB. Conversely, any colours that occur in the PCS LAB encoding range that are outside of the encoding range of PCS XYZ shall be clipped on a per-component basis to the PCS XYZ range when transforming from LAB into XYZ.

  When converting to float32Number-based encodings:

  Conversion of PCS XYZ to PCS LAB is performed and encoded using the float32Number encoding of PCS values as defined in section 6.3.4.2. Conversely, conversion of PCS XYZ to PCS LAB is performed and encoded using the float32Number encoding of PCS values as defined in section 6.3.4.2. No clipping is performed in either case.

  NOTE: In order to calculate LAB values from negative XYZ values, the straight line portion of the LAB color component transfer function below 0,008856 shall be extended so that its domain goes to negative infinity.

- **New Clause: 6.5 Device Encoding**

  The specification of device value encoding is determined by the device. Normally, device values in the range of 0,0 to 1,0 are encoded using a 0 to 255 (FFh) range when using 8 bits and are encoded using a 0 to 65535 (FFFFh) range when using 16 bits. When encoding using float32Number values in DToBx and BToDx tags, device values may be outside the 0,0 to 1,0.

- **Add to the end of Clause 8.3.2 "N-component LUT-based input profiles"**

  Optional DToB0Tag (see 9.2.x), DToB1Tag (see 9.2.x), DToB2Tag (see 9.2.x), DToB3Tag (see 9.2.x), BToD0Tag (see 9.2.x), BToD1Tag (see 9.2.x), BToD2Tag (see 9.2.x), BToD3Tag (see 9.2.x) may also be included.

- **Add to the end of Clause 8.4.2 "N-component LUT-based display profiles"**

  Optional DToB0Tag (see 9.2.x), DToB1Tag (see 9.2.x), DToB2Tag (see 9.2.x), DToB3Tag (see 9.2.x), BToD0Tag (see 9.2.x), BToD1Tag (see 9.2.x), BToD2Tag (see 9.2.x), BToD3Tag (see 9.2.x) may also be included.

- **Add to the end of Clause 8.5.2 "N-component LUT-based output profiles"**

  Optional DToB0Tag (see 9.2.x), DToB1Tag (see 9.2.x), DToB2Tag (see 9.2.x), DToB3Tag (see 9.2.x), BToD0Tag (see 9.2.x), BToD1Tag (see 9.2.x), BToD2Tag (see 9.2.x), BToD3Tag (see 9.2.x) may also be included.

- **Add to the end of Clause 8.6 "DeviceLink profile" (just before the NOTE)**

  A device link profile can optionally contain a DToB0Tag (see 9.2.x).

- **Add to the end of Clause 8.7 "ColorSpace conversion profile"**

Optional DToB0Tag (see 9.2.x), DToB1Tag (see 9.2.x), DToB2Tag (see 9.2.x), DToB3Tag (see 9.2.x), BToD0Tag (see 9.2.x), BToD1Tag (see 9.2.x), BToD2Tag (see 9.2.x), BToD3Tag (see 9.2.x) may also be included.

- **Add to the end of Clause 8.8 “Abstract profile”**

  An Abstract profile can optionally contain a DToB0Tag (see 9.2.x).

- **Replace text in Clause 8.10 “Priority of tag usage” with the following text:**

  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.

  For Input, Display, Output, or ColorSpace profile types, the priority of the tag usage for each rendering intent shall be:

  1. BToD0Tag, BToD1Tag, BToD2Tag, BToD3Tag, DToB0Tag, DToB1Tag, DToB2Tag, or DToB3Tag designated for the rendering intent
  2. BToA0Tag, BToA1Tag, BToA2Tag, AToB0Tag, AToB1Tag, or AToB2Tag designated for the rendering intent
  3. BToA0Tag or AToB0Tag
  4. TRCs (redTRCTag, greenTRCTag, blueTRCTag, or grayTRCTag) and colorants (redMatrixColumnTag, greenMatrixColumnTag, blueMatrixColumnTag)

  For DeviceLink or Abstract profile types, the priority of the tag usage shall be:

  1. DToB0Tag
  2. AToB0Tag

  For all profile types, the available valid tag with the lowest priority number defines the transform. If the CMM does not need or does not support the BToDxTags and DToBxTags, or if the CMM does not understand a particular processing element in the BToDxTags or DToBxTags, the B2Dx and D2Bx tags shall not be used by such a CMM.

- **Clause 9.1: Add new paragraph at the end**

  The DToBxTags and BToDxTags represent colour transforms that operate on a range of values encoded by 32-bit IEEE 754 floating-point numbers. The processing model is described in detail in 10.x. These tags are optional for all profile classes.

  The “D” space represents a 32-bit IEEE 754 floating-point-encoded Device Space. In this space, negative as well as positive values (including values greater than 1,0) are allowed when such values are supported by the device.

  The “B” space represents the PCS, identical to the “B” space in the AToBxTags and BToAxTags. The encoding range of the “B” space in the DToBxTags and BToDxTags is defined in section 6.3.4.2,as is the case with the “B” space in AToBxTags and BToAxTags. (See section 6.3).

  The DToB3 and BToD3 tags allow the ability to directly encode the absolute rendering intent in a profile. The PCS for DToB3 and BToD3 represents ICC-absolute colorimetry with the values encoded as float32Number. The mediaWhitePoint tag is NOT used in the calculation of ICC-Absolute colorimetry from the data in the DToB3 and BToD3 tags.

- **New Clause: 9.2.x BToD0Tag**

  Tag signature ‘B2D0’ (42324430h)

  Allowed tag types: multiProcessElementsType

  This tag defines a colour transform from PCS to Device. It supports float32Number-encoded input range, output range and transform, and provides a means to override the BToA0Tag. As with the BToA0Tag, it

defines a transform to achieve a perceptual rendering. The processing mechanism is described in multiProcessElementsType (see 10.x).

- **New Clause: 9.2.x BToD1Tag**

  Tag signature ‘B2D1’ (42324431h)

  Allowed tag types: multiProcessElementsType

  This tag defines a colour transform from PCS to Device. It supports float32Number-encoded input range, output range and transform, and provides a means to override the BToA1Tag. As with the BToA1Tag, it defines a transform to achieve a colorimetric rendering. The processing mechanism is described in multiProcessElementsType (see 10.x).

- **New Clause: 9.2.x BToD2Tag**

  Tag signature ‘B2D2’ (42324432h)

  Allowed tag types: multiProcessElementsType

  This tag defines a colour transform from PCS to Device. It supports float32Number-encoded input range, output range and transform, and provides a means to override the BToA2Tag. As with the BToA2Tag, it defines a transform to achieve a saturation rendering. The processing mechanism is described in multiProcessElementsType (see 10.x).

- **New Clause: 9.2.x BToD3Tag**

  Tag signature ‘B2D3’ (42324433h)

  Allowed tag types: multiProcessElementsType

  This tag defines a colour transform from PCS to Device. It supports float32Number-encoded input range, output range and transform, and provides a means to override the BToA1Tag and associated Absolute Rendering Intent processing. As with the BToA1Tag and associated Absolute Rendering Intent processing, it defines a transform to achieve an absolute rendering. The processing mechanism is described in multiProcessElementsType (see 10.x).

- **New Clause: 9.2.x DToB0Tag**

  Tag signature ‘D2B0’ (44324230h)

  Allowed tag types: multiProcessElementsType

  This tag defines a colour transform from Device to PCS. It supports float32Number-encoded input range, output range and transform, and provides a means to override the AToB0Tag. As with the AToB0Tag, it defines a transform to achieve a perceptual rendering. The processing mechanism is described in multiProcessElementsType (see 10.x).

- **New Clause: 9.2.x DToB1Tag**

  Tag signature ‘D2B1’ (44324231h)

  Allowed tag types: multiProcessElementsType

  This tag defines a colour transform from Device to PCS. It supports float32Number-encoded input range, output range and transform, and provides a means to override the AToB1Tag. As with the AToB1Tag, it defines a transform to achieve a colorimetric rendering. The processing mechanism is described in multiProcessElementsType (see 10.x).

- **New Clause: 9.2.x DToB2Tag**

  Tag signature ‘D2B2’ (44324232h)

  Allowed tag types: multiProcessElementsType

  This tag defines a colour transform from Device to PCS. It supports float32Number-encoded input range, output range and transform, and provides a means to override the AToB2Tag. As with the AToB2Tag, it

defines a transform to achieve a saturation rendering. The processing mechanism is described in multiProcessElementsType (see 10.x).

- **New Clause: 9.2.x DToB3Tag**

  Tag signature ‘D2B3’ (44324233h)

  Allowed tag types: multiProcessElementsType

  This tag defines a colour transform from Device to PCS. It supports float32Number-encoded input range, output range and transform, and provides a means to override the AToB1Tag with associated Absolute Rendering Intent processing. As with the AToB1Tag and associated Absolute Rendering Intent processing, it defines a transform to achieve an absolute rendering. The processing mechanism is described in multiProcessElementsType (see 10.x).

- **New Clause: 10.x multiProcessElementsType**

  10.x.1 General

  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. Currently supported processing elements are: a set of one dimensional curves, a matrix with offset terms, and a multidimensional lookup table (CLUT). 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 behavior shall be followed.

  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 31.

  When used, the byte assignment and encoding shall be as given in Table B.

Table B – 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, must be set to 0	
8..9	2	Number of Input Channels	uInt16Number
10..11	2	Number of Output Channels	uInt16Number
12..15	2	Number of Processing Elements (N)	uInt32Number
16..15+8N	8 x N	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.

NOTE: It is permitted to share data between processing elements. For example, the offsets for some processing elements can be identical.

10.x.2 multiProcessElementsType 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. 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.

Clipping of the results of a processing element shall not be performed. Some processing elements may perform clipping as needed on input.

NOTE: The specification for each processing element will indicate whether that element will perform clipping on input.

NOTE: The general element encoding for multiProcessElementsType elements is shown in Table C.

Table C – General Element encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	Element Signature	
4..7	4	reserved, must be set to 0	
8..9	2	Number of Input Channels (P)	uInt16Number
10..11	2	Number of Output Channels (Q)	uInt16Number
12..end		Data	

10.x.2.1 Curve Set Element

The Curve Set Element encodes multiple one dimensional curves. The encoding is shown in Table D.

Table D – Curve Set Element encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	'cvst' (6D666C74h) [curve set element table] type signature	
4..7	4	reserved, must be set to 0	
8..9	2	Number of Input Channels (P)	uInt16Number
10..11	2	Number of Output Channels (Q)	uInt16Number
12..11+8P	8 x P	Curve Positions (Offset and Size)	positionNumber[]
12+8P..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 will be the same as the number of inputs. Thus, the number of output channels (Q) shall be set to the same value as the number of input channels (P).

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.

NOTE: It is permitted to share data between one dimensional curves. For example, the offsets for some one dimensional curves can be identical.

Each curve is stored in one or more curve segments, with break-points specified between curve segments. The first curve segment always starts at –Infinity, and the last curve segment always ends at +Infinity. 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. The encoding for such a curve is shown in Table F.

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 E.

Table E – One-dimensional curves encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	'curf' (63757266h) type signature	
4..7	4	Reserved, must be set to 0	
8..9	2	Number of segment(s) (N)	uInt16Number
10..11	2	Reserved, must be set to 0	
12..4N+7	4 x (N-1)	N-1 Break-Points	float32Number[…]
4N+8..end		Segments 1 to N	

Break-points separate two curve segments. The 1st curve segment is defined between –Infinity and break-point 1 (included). The kth 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 +Infinity.

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.

Curve segments that are specified in terms of a formula shall be encoded as shown in Table F.

Table F – Formula curve segments encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	'parf' (70617266h) type signature	
4..7	4	Reserved, must be set to 0	
8..9	2	Encoded value of the function type	uInt16Number
10..11	2	Reserved, must be set to 0	
12..end	See Table G	Parameters (see table G)	float32Number[…]

The encoding for the function type field and the parameters is shown in Table G:

Table G – Formula curve segments 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

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 H.

Table H – Sampled curve segment encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	'samf' (73616D66h) type signature	
4..7	4	Reserved, must be set to 0	
8..11	4	Count (N) specifying the number of entries that follow	uInt32Number
12..end	4 x N	Curve entries	float32Number[…]

The Count (N) shall be greater than or equal to 2.

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 (BP_k) and break-point k+1 (BP_{k+1}), the j^{th} sample ($j \in [1, N]$) shall correspond to the input value $BP_k + j * (BP_{k+1} – BP_k) / N$. Thus BP_k is excluded.

If the number of grid points in a particular segment of a one-dimensional curve 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.x.2.2 Matrix Element

The matrix is organized as an array of PxQ+Q elements, where P is the number of input channels to the matrix, and Q is the number of output channels. The matrix elements are each float32Numbers. The array is organized as follows:

$array = [e_{11}, e_{12}, \ldots, e_{1P}, e_{21}, e_{22}, \ldots, e_{2P}, \ldots, e_{Q1}, e_{Q2}, \ldots, e_{QP}, e_1, e_2, \ldots, e_Q]$

Table I –Matrix Element encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	'matf' (6D617466h) type signature	
4..7	4	Reserved, must be set to 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 the following equation:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \dots \\ Y_Q \end{bmatrix} = \begin{bmatrix} e_{11} & e_{12} & \dots & e_{1P} \\ e_{21} & e_{22} & \dots & e_{2P} \\ \dots & \dots & \dots & \dots \\ e_{Q1} & e_{Q2} & & e_{QP} \end{bmatrix} \bullet \begin{bmatrix} X_1 \\ X_2 \\ \dots \\ X_P \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \dots \\ e_Q \end{bmatrix} \quad \text{(x)}$$

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.

10.x.2.3 CLUT Element

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 an 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. The equation for computing the byte size of the CLUT is defined below.

nGrid1 * nGrid2 *... * nGridP * number of output channels (Q) * 4 (x)

When used, the byte assignment and encoding for the CLUT shall be as given in Table J.

Table J – CLUT Element encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	‘clut’ (636C7574h) type signature	
4..7	4	Reserved, must be set to 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 set to 00h.	uInt8Number
28..end	See equation x above	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 2 grid points for each dimension.

10.x.2.4 Future Expansion Elements

The ‘bACS’ and ‘eACS’ element types are provided as placeholders for future expansion. If present, these elements shall be considered as pass through elements with no modification of channel data

Table K – bACS Element encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	‘bACS ‘ (62414353h) type signature	
4..7	4	Reserved, must be set to 0	
8..9	2	Number of Input Channels (P)	uInt16Number
10..11	2	Number of Output Channels (Q)	uInt16Number
12..15	4	Signature	

Table L – eACS Element encoding

Byte Position	Field Length (bytes)	Content	Encoded as…
0..3	4	‘eACS ‘ (65414353h) type signature	
4..7	4	Reserved, must be set to 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).

5. Applications and Workflows

5.1 Digital Motion Picture Workflow

This proposal enables the DMP workflow described in Section 1. Consider the connection of a new Profile A which is a DPX Scene profile, and a new Profile B which is an RGB Working Space profile.

Profile A:

- Contains traditional A2Bx and B2Ax tags.
- Contains new D2B0, D2B2, B2D0 and B2D2 tags that provide float32Number elements to convert between float32Number-encoded device space and the PCS.
- Contains new D2B3 and B2D3 tags containing unbounded XYZ absolute PCS.

Profile B:

- Contains traditional TRC/Matrix tags.
- Contains new D2B3 and B2D3 tags that include unbounded XYZ absolute PCS.

Both profiles include D2B3/B2D3 tags that make use of an unbounded XYZ absolute PCS.

When using a CMM that is unaware of the new optional D2Bx/B2Dx tags, the connection occurs using the A2Bx/B2Ax and the TRC/Matrix tags. This enables existing CMMs to use the new profiles in a completely backwards-compatible manner.

DMP workflows would generally use only the unbounded Absolute Colorimetric Tags D2B3 and B2D3. However, the other tags are included for completeness – these tags are useful to provide higher precision colour transforms when required.

5.2 Example of Other Workflows

The proposal (with possible future enhancements) allows the direct encoding of device modeling within the profile. Additional elements may be required to make this happen, but this proposal lays the groundwork for these future enhancements.