SMPTE Standard

Common LUT Format — Core Specification

Table of contents

50 Main St., Suite 1000, White Plains, NY 10606, (914) 761-1100
Approved February 26, 2026

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in its Standards Operations Manual. This SMPTE Engineering Document was prepared by Technology Committee TC-10E Essence.

Introduction

Look-Up Tables (LUTs) are widely used to perform transformations between different sets of color values. With a large number of product developers providing software and hardware solutions for LUTs, there is an explosion of unique, vendor-specific LUT file formats, which are often only trivially different from each other. This can create workflow problems when a LUT being used on a production is not supported by one or more of the applications being used. Furthermore, many LUT formats are designed for a particular use case only and lack the quality, flexibility, and metadata needed to meet modern requirements.

The Common LUT Format (CLF) addresses these issues by providing a standardized method for describing an arbitrary sequence of color processing operations, called process nodes, which are sequentially applied to achieve an end result. Supported operator types include matrices, 1D and 3D LUTs, ASC-CDL adjustments, logarithmic or exponential shaper functions. CLF can be used to encapsulate any supported color transforms as an XML document conforming to the CLF schema, even when no 1D or 3D LUTs are included.

At the time of publication, no notice had been received by SMPTE claiming patent rights essential to the implementation of this Engineering Document. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. SMPTE shall not be held responsible for identifying any or all such patent rights.

1 Scope

This document defines a human-readable XML file format for the interchange of color transformations, specified using an XML schema. The format supports Look-Up Tables of several types: 1D LUTs, 3D LUTs, and 3×1D LUTs, as well as additional transformation elements such as matrices, range scaling, and “shaper LUTs.” The document defines the requirements for a valid CLF instance.

2 Conformance

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any clause explicitly labeled as "Informative" or individual paragraphs that start with "NOTE:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword "reserved" indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword "forbidden" indicates "reserved" and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; tables shall be next; then formal languages; then figures; and then any other language forms.

3 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this standard. Dated references require that the specific edition cited shall be used as the reference. Undated citations refer to the edition of the referenced document (including any amendments) current at the date of publication of this document. All standards are subject to revision, and users of this engineering document are encouraged to investigate the possibility of applying the most recent edition of any undated reference.

Amendment 1:2011 to SMPTE ST 433:2008, *D-Cinema — XML Data Types*. https://doi.org/10.5594/SMPTE.ST433.2008Am1.2011

IEEE 754-2019, *IEEE Standard for Floating-Point Arithmetic*

SMPTE ST 433:2008, *D-Cinema — XML Data Types*. https://doi.org/10.5594/SMPTE.ST433.2008

World Wide Web Consortium (W3C) (26 November 2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*

World Wide Web Consortium (W3C) (2004, October 28), *XML Schema Part 1: Structures (Second Edition)*

World Wide Web Consortium (W3C) (28 October 2004). *XML Schema Part 2: Datatypes (Second Edition)*

World Wide Web Consortium (W3C) (11 April 2013). *XML Signature Syntax and Processing Version 1.1*

4 Terms and definitions

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

4.1
Look-Up Table
LUT

pre-computed array that replaces computation by an array indexing operation

4.2
color tuple

set of numbers consisting of either one or three ordered elements that represent color components

Note 1 to entry: If the color tuple is one component, then the element is named ‘gray’. If the color tuple has 3 components, then the elements are named ‘red’, ‘green’, and ‘blue’, in that order.

4.3
ProcessList

top-level element in a CLF instance that describes an ordered set of `ProcessNode` instances

4.4
ProcessNode

abstract element within a CLF that describes one or more mathematical operations to be applied to a color tuple

4.5
1D LUT

LUT of one column to be applied to all color components

4.6
3×1D LUT

LUT of three columns, where each column defines a 1D LUT corresponding to each color component

4.7
3D LUT

LUT of three columns defining a 3D lattice, where each axis is an entry point for one of the three color components where output values from the 3D LUT are calculated by interpolating the nearest points in the lattice

5 Specification

5.1 General

A Common LUT Format (CLF) file shall be written using Extensible Markup Language (XML) and adhere to the schema defined in Subclause 5.2. A CLF file shall have the file extension '.clf'.

Each CLF file shall be completely self-contained, requiring no external information or metadata.

Subclause 6.5 defines the `ProcessList`. The top-level element in a CLF instance is a `ProcessList` which describes an ordered set of color transformations, individually called `ProcessNode` instances (see Figure 1). `ProcessNode` instances shall be applied sequentially to a color tuple in the order defined by the `ProcessList`. All `ProcessNode` instances in the `ProcessList` shall be compatible with the number of color components included in the initial input color tuple.

At least one `ProcessNode` element shall be included in a `ProcessList`.

The abstract `ProcessNode` is defined in Subclause 6.4 and allowed substitutes for `ProcessNode` are defined in Clause 7.

Clause 8 defines Profiles constraining the Common LUT Format.

Annex B contains sample CLF instances.

Annex C lists the Media Type registration of the CLF with IANA.

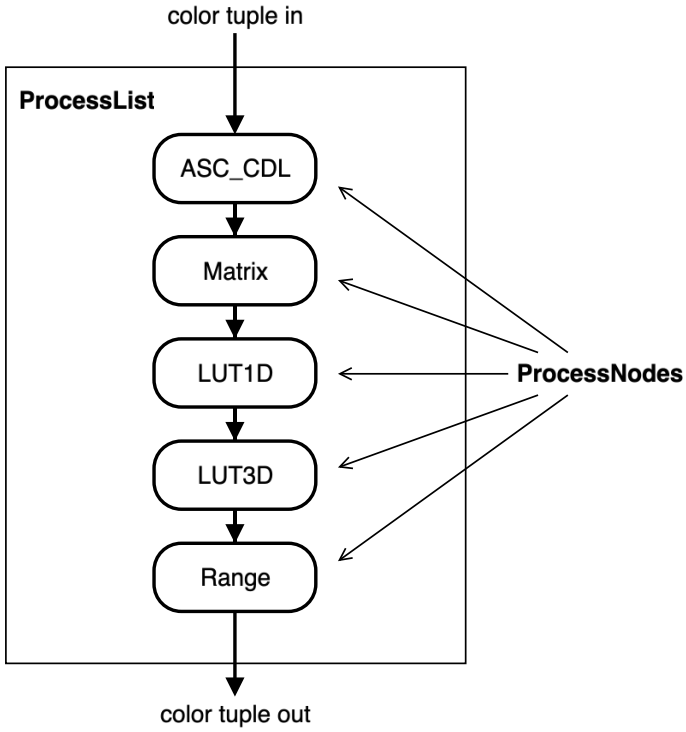

Figure 1 — Example of a `ProcessList` containing a sequence of multiple `ProcessNode` instances

5.2 XML Structure

5.2.1 XML Schema and Namespace

The XML Schema defined in this document shall conform to the following specifications:

- W3C XML Schema Part 1: Structures,
- W3C XML Schema Part 2: Datatypes,
- W3C XML Signature Syntax and Processing Version 1.1, and
- SMPTE ST 433.

XML elements and attributes defined by this standard shall conform to the XML schema definition found in the prose element and element “a” (see Annex D for a list of elements).

The XML schema root element shall be as defined in Table 1.

The namespace prefixes used in XML Schema definitions herein are not normative values and implementations shall perform correctly with any XML-compliant prefix values.

Table 1 — XML Schema root element definition

```
<xs:schema targetNamespace="http://www.smpte-ra.org/ns/2136-1/2024"
  xmlns:dcml="http://www.smpte-ra.org/schemas/433/2008/dcmlTypes/"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:clf="http://www.smpte-ra.org/ns/2136-1/2024"
  xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"
  vc:minVersion="1.0" elementFormDefault="qualified"
                          attributeFormDefault="unqualified">
  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"/>
  <xs:import namespace="http://www.smpte-ra.org/schemas/433/2008/dcmlTypes/"/>
<!--  schema definitions found in this document excluding this line  -->
</xs:schema>
```

A CLF instance shall consist of a single `ProcessList` element. The `ProcessList` element shall contain an ordered list of one or more `ProcessNode` elements.

NOTE 1 The output of the last node of the `ProcessList` is the final output of the CLF instance. CLF allows for output values to be floating-point, which can allow applications to cast the output values to any format, including integer formats.

NOTE 2 A `ProcessList` is required even if only one `ProcessNode` is present.

5.2.2 XML Version

A CLF file shall be an XML document as specified in W3C XML 1.0.

5.2.3 Text Encoding

Text shall be encoded as UTF-8.

5.2.4 Language (informative)

Elements of type `dcml:UserTextType` can signal a language by specifying the `language` attribute, as defined in SMPTE ST 433. If the `language` attribute is not present, the default language is English.

6 XML Data Types

6.1 Simple Data Types

Simple Data Types have no attributes and are referenced by complex Data Types as defined in Subclause 6.2.

The Simple Data Types shall be as defined in Table 2.

Table 2 — Simple Data Types

```
<!--  Data Type: floatListType  -->
<xs:simpleType name="floatListType">
  <xs:list itemType="xs:float"/>
</xs:simpleType>

<!--  Data Type: positiveFloatType  -->
<xs:simpleType name="positiveFloatType">
  <xs:restriction base="xs:float">
    <xs:minExclusive value="0"/>
  </xs:restriction>
</xs:simpleType>

<!--  Data Type: nonNegativeFloatType  -->
<xs:simpleType name="nonNegativeFloatType">
  <xs:restriction base="xs:float">
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>

<!--  Data Type: floatListLength3Type  -->
<xs:simpleType name="floatListLength3Type">
  <xs:restriction>
    <xs:simpleType>
      <xs:list itemType="xs:float"/>
    </xs:simpleType>
    <xs:length value="3"/>
  </xs:restriction>
</xs:simpleType>

<!--  Data Type: positiveFloatListLength3Type  -->
<xs:simpleType name="positiveFloatListLength3Type">
  <xs:restriction>
    <xs:simpleType>
```

```
      <xs:list itemType="clf:positiveFloatType"/>
    </xs:simpleType>
    <xs:length value="3"/>
  </xs:restriction>
</xs:simpleType>

<!--  Data Type: nonNegativeFloatListLength3Type  -->
<xs:simpleType name="nonNegativeFloatListLength3Type">
  <xs:restriction>
    <xs:simpleType>
      <xs:list itemType="clf:nonNegativeFloatType"/>
    </xs:simpleType>
    <xs:length value="3"/>
  </xs:restriction>
</xs:simpleType>

<!--  Data Type: dimType  -->
<xs:simpleType name="dimType">
  <xs:restriction base="xs:string">
    <xs:minLength value="3"/>
    <xs:pattern value="[\d\s]*\d"/>
  </xs:restriction>
</xs:simpleType>

<!--  Data Type: bitDepthType  -->
<xs:simpleType name="bitDepthType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="8i"/>
    <xs:enumeration value="10i"/>
    <xs:enumeration value="12i"/>
    <xs:enumeration value="16i"/>
    <xs:enumeration value="16f"/>
    <xs:enumeration value="32f"/>
  </xs:restriction>
</xs:simpleType>

<!--  Data Type: channelType  -->
<xs:simpleType name="channelType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[RGB]"/>
  </xs:restriction>
</xs:simpleType>
```

Table 3 explains the usage of the Simple Data Types.

Table 3 — Simple Data Types (informative)

Element	Description
floatListType	Used in LUT1D and LUT3D to carry a sorted list of LUT entries.
positiveFloatType	Basic data type for positive, non-zero float values. It is used in the positiveFloatListLength3Type.
nonNegativeFloatType	Basic data type for non-negative float values including zero. It is used in the Saturation element of the ASC_CDL ProcessNode.
floatListLength3Type	Defines a list of three float values. It is used in the Offset element of the ASC_CDL ProcessNode.
positiveFloatListLength3Type	Defines a list of three positive, non-zero float values. It is used in the Power element of the ASC_CDL ProcessNode.
nonNegativeFloatListLength3Type	Defines a list of three positive float values including zero. It is used in the Slope element of the ASC_CDL ProcessNode.
dimType	Defines a list of integer values representing, depending on its context, the dimensions of a matrix, or the length and number of components of a LUT.
bitDepthType	A set of combinations of bit depth and encoding (integer or float) of color tuple components.
channelType	Defines a pattern for specifying a color component as "R", "G" or "B".

6.2 Complex Data Types

6.2.1 ArrayType

The ArrayType type is used in the LUT1D and LUT3D ProcessNode elements. The dim attribute shall specify the dimensions of a LUT. The specific formatting of the dim attribute shall match the type of node in which it is being used. See Subclauses 7.2.3.2 and 7.3.3.2 for the semantics of dim for the LUT1D and LUT3D process nodes, respectively.

The ArrayType shall be as defined in Table 4.

Table 4 — ArrayType definition

```
<xs:complexType name="ArrayType">
  <xs:simpleContent>
    <xs:extension base="clf:floatListType">
      <xs:attribute name="dim" type="clf:dimType" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

6.2.2 MatrixArrayType

The `MatrixArrayType` type is used in the `Matrix ProcessNode`. The `dim` attribute shall specify the dimensions in rows and columns of the matrix and shall be limited to either `"3 3"` or `"3 4"`.

The `MatrixArrayType` shall be as defined in Table 5.

Table 5 — `MatrixArrayType` definition

```
<xs:complexType name="MatrixArrayType">
  <xs:simpleContent>
    <xs:restriction base="clf:ArrayType">
      <xs:attribute name="dim" use="required">
        <xs:simpleType>
          <xs:restriction base="clf:dimType">
            <xs:enumeration value="3 3"/>
            <xs:enumeration value="3 4"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

6.3 Bit Depths

All processing shall be performed using 32-bit floating point values. The values of the `inBitDepth` and `outBitDepth` attributes shall not affect the range or quantization of color values.

An integer `inBitDepth` or `outBitDepth` of a `ProcessNode` does not indicate that any clamping or quantization should be done. These attributes are strictly used to indicate the scaling of parameter and array values within the `LUT1D`, `LUT3D`, `Matrix`, and `Range ProcessNode` instances.

Using values other than `32f` allows direct compatibility with legacy LUT formats where array values may be represented in their original scaling.

The `inBitDepth` of a `ProcessNode` shall match the `outBitDepth` of the preceding `ProcessNode` (if a preceding `ProcessNode` exists).

The possible values for the `inBitDepth` and `outBitDepth` strings shall be as listed in Table 6.

Table 6 — Supported values for bitDepth attributes

bitDepth	Description	Scale Factor
`8i`	8-bit unsigned integer	255
`10i`	10-bit unsigned integer	1023
`12i`	12-bit unsigned integer	4095
`16i`	16-bit unsigned integer	65535
`16f`	16-bit floating point (half-float)	1.0
`32f`	32-bit floating point (single)	1.0

6.4 ProcessNode

6.4.1 General

The abstract `ProcessNode` element contains attributes and elements that are common to and inherited by any of the substitutes for `ProcessNode` defined in Clause 7.

The `ProcessNode` element shall be as defined in Table 7.

Table 7 — `ProcessNode` definition

```
<xs:element name="ProcessNode" type="clf:ProcessNodeType"/>
<xs:complexType name="ProcessNodeType" abstract="true">
  <xs:sequence>
    <xs:element name="Description" type="dcml:UserTextType" minOccurs="0"
                        maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="name" type="xs:string" use="optional"/>
  <xs:attribute name="inBitDepth" type="clf:bitDepthType" use="required"/>
  <xs:attribute name="outBitDepth" type="clf:bitDepthType" use="required"/>
  <xs:anyAttribute processContents="lax"/>
</xs:complexType>
```

6.4.2 Attributes

The XML attributes of `ProcessNode` shall be as specified in Table 8.

Table 8 — XML attributes of ProcessNode

Attribute	Description
`id`	A string for identifying a `ProcessNode` instance. NOTE The `id` attribute cannot be expected to be unique.
`name`	A concise string to define a name for the `ProcessNode` that can be used by an application for display in a user interface.
`inBitDepth`	A string indicating how array or parameter values have been scaled.
`outBitDepth`	A string indicating how array or parameter values have been scaled.

Additional user-specific attributes may appear in any order and in any namespace.

6.4.3 Elements

The XML child elements of the `ProcessNode` element shall be as specified in Table 9.

Table 9 — XML child elements of `ProcessNode`

Element	Description
`Description`	An arbitrary string to describe the function, usage, or any notes about the `ProcessNode`.

6.5 ProcessList

6.5.1 General

The `ProcessList` element shall be as defined in Table 10.

Table 10 — `ProcessList` definition

```
<xs:element name="ProcessList" type="clf:ProcessListType"/>
<xs:complexType name="ProcessListType">
  <xs:sequence>
    <xs:element ref="clf:Id" minOccurs="0"/>
    <xs:element ref="clf:Description" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="clf:InputDescriptor" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="clf:OutputDescriptor" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="clf:Info" minOccurs="0"/>
    <xs:element ref="clf:ProcessNode" maxOccurs="unbounded"/>
    <xs:element name="Signer" type="ds:KeyInfoType" minOccurs="0"/>
    <xs:element ref="ds:Signature" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="compCLFversion" type="xs:string" use="optional"/><!--
                          deprecated, CLF version is signalled by the schema URI -->
  <xs:attribute name="name" type="xs:string" use="optional"/>
  <xs:attribute name="inverseOf" type="xs:string" use="optional"/>
  <xs:anyAttribute processContents="lax"/>
</xs:complexType>
<xs:element name="Id" type="dcml:UUIDType"/>
<xs:element name="Description" type="dcml:UserTextType"/>
<xs:element name="InputDescriptor" type="dcml:UserTextType"/>
<xs:element name="OutputDescriptor" type="dcml:UserTextType"/>
```

6.5.2 Attributes

The XML attributes of `ProcessList` shall be as specified in Table 11.

Table 11 — XML attributes of `ProcessList`

Attribute	Description
`id`	A string for identifying a `ProcessList` instance. NOTE 1 The `id` attribute cannot be expected to be unique.
`compCLFversion`	Indicates the minimum compatible CLF specification version required to read this file. If present, the `compCLFversion` corresponding to this version of the specification shall be "ST2136-1:2026". NOTE 2 This attribute is defined for legacy reasons and can be ignored by new implementations.
`name`	Text name of the `ProcessList` for display or selection from an application's user interface.
`inverseOf`	References another `ProcessList` `id` attribute that is the inverse of this one.

Additional user-specific attributes may appear in any order and in any namespace.

6.5.3 Elements

The XML child elements of `ProcessList` shall be as specified in Table 12 and shall be ordered as given in Table 10.

Table 12 — XML child elements of `ProcessList`

Element	Description
`Id`	If present, uniquely identifies the `ProcessList` instance. No two `ProcessList` elements shall have the same `Id` values unless they are identical. NOTE The `Id` element can be used for referencing a CLF instance. It is up to implementations to ensure that the `Id` element is updated when the CLF file is modified.
`Description`	Describes the function, usage, or any notes about the `ProcessList`. A `ProcessList` can contain zero or more Description elements.
`InputDescriptor`	Describes the color space encoding expected to be input to the `ProcessList`.
`OutputDescriptor`	Describes the intended output target of the `ProcessList` (e.g., target display).
`Info`	May include additional custom metadata. Some child elements are specified in Table 14. Info may contain any other child element in any order.
`ProcessNode`	A generic XML element that in practice is substituted with a particular color operator. The `ProcessList` shall contain at least one `ProcessNode`. The `ProcessNode` is defined in 6.4.

Element	Description
Signer	The Signer element shall uniquely identify the entity that digitally signed the ProcessList. If the Signer element is present, then the Signature element shall also be present. If X.509 certificates are used as specified in W3C XML Signature Syntax and Processing Version 1.1, then the Signer element shall contain one X509Data element containing one X509IssuerSerial element, which uniquely identifies the certificate used to sign the ProcessList.
Signature	The Signature element shall contain a digital signature authenticating the CLF file. If the Signature element is present, then the Signer element shall be present. The digital signature shall be enveloped, as specified in W3C XML Signature Syntax and Process Version 1.1, and apply to the entire ProcessList. The signature is generated by the signer, as identified by the Signer element.

The XML Info child element of the ProcessList element shall be as defined in Table 13. The schema definition of the Info element does not require a specific ordering of sub-elements and allows custom sub-elements to be used. Any custom metadata should be stored in custom sub-elements of Info.

Table 14 defines several optional sub-elements of Info. The Info element shall contain zero or one instance of the following sub-elements defined in Table 14: Profile, Keywords.

All elements of Info are intended for information only and can be ignored by CLF processors.

Table 13 — Info definition

```
<xs:element name="Info" type="clf:InfoType"/>
<xs:complexType name="InfoType">
  <xs:sequence>
    <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"
                          processContents="lax"/>
  </xs:sequence>
  <xs:anyAttribute processContents="lax"/>
</xs:complexType>
<!-- Examples of elements which may be used in the Info element -->
<xs:element name="Profile" type="xs:string"/>
<xs:element name="AppRelease" type="dcml:UserTextType"/>
<xs:element name="Copyright" type="dcml:UserTextType"/>
<xs:element name="Revision" type="dcml:UserTextType"/>
<xs:element name="Keywords" type="dcml:UserTextType"/>
```

Table 14 — XML child elements of `Info`

Element	Description
`Profile`	If present, the value of the `Profile` element shall be the URI of the Profile to which the CLF instance conforms (see Clause 8).
`AppRelease`	Indicates the application and release level of software used to create the CLF.
`Copyright`	A string containing a copyright notice for authorship of the CLF file.
`Revision`	A string used to track the version of the LUT itself (e.g., an increased resolution from a previous version of the LUT).
`Keywords`	It is optional in this document, but can be further constrained in CLF Profile specifications.

6.6 General (informative)

Figure 2 depicts the XML structure of the Common LUT Format.

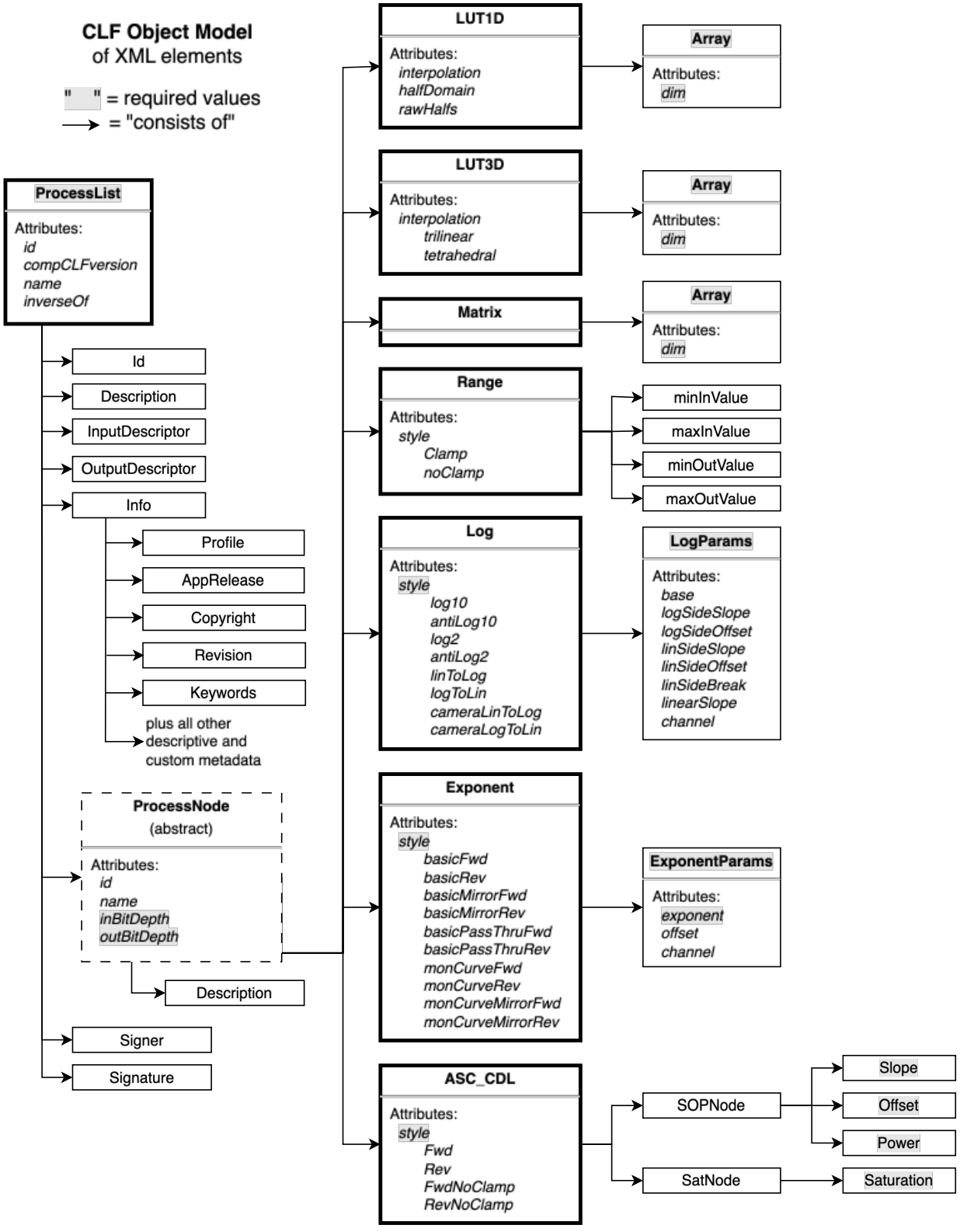

Figure 2 — Object Model of XML Elements

7 Substitutes for ProcessNode

7.1 General

The attributes and elements defined for `ProcessNode` are inherited by the substitutes for `ProcessNode`. Clause 7 defines the available substitutes for the generalized `ProcessNode` element.

7.2 LUT1D

7.2.1 General

The `LUT1D` element shall be as defined in Table 15.

Table 15 — `LUT1D` definition

```
<xs:element name="LUT1D" type="clf:LUT1DType" substitutionGroup="clf:ProcessNode"/>
<xs:complexType name="LUT1DType">
  <xs:complexContent>
    <xs:extension base="clf:ProcessNodeType">
      <xs:sequence>
        <xs:element name="Array" type="clf:ArrayType"/>
      </xs:sequence>
      <xs:attribute name="interpolation" use="optional" fixed="linear"
                    type="xs:string"/>
      <xs:attribute name="halfDomain" type="xs:string" use="optional" fixed="true"/>
      <xs:attribute name="rawHalfs" type="xs:string" use="optional" fixed="true"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A `LUT1D` transform takes in an input color tuple value, finds the two nearest index positions in a Look-Up Table, and then interpolates an output value using the table entries associated with those positions.

Linear interpolation shall be used for `LUT1D`. Linear interpolation for both normal and half-domain LUTs is described in Annex A.

EXAMPLE
A simple `LUT1D`.

```
<LUT1D name="2 Value Lut" inBitDepth="12i" outBitDepth="12i">
     <Description>1D LUT - Inverts coded values</Description>
     <Array dim="2 1">
          4095
          0
     </Array>
</LUT1D>
```

7.2.2 Attributes

7.2.2.1 General

The XML attributes of `LUT1D` shall be as specified in Table 16.

Table 16 — XML attributes of `LUT1D`

Attribute	Description
`interpolation`	See Subclause 7.2.2.2.
`halfDomain`	See Subclause 7.2.2.3.
`rawHalfs`	See Subclause 7.2.2.4.

7.2.2.2 interpolation

This attribute is a string that indicates the preferred algorithm used to interpolate values in the 1D LUT.

This attribute is optional but, if present, its value shall be "linear".

7.2.2.3 halfDomain

This attribute's presence indicates that the input domain to the node shall be all possible 16-bit floating-point values.

If this attribute is present, its value shall be `"true"`, and the `Array` element shall contain 65536 entries.

EXAMPLE
The unsigned integer 15360 has the same bit-pattern (0011110000000000) as the half-float value 1.0, so the 15360th entry (zero-indexed) in the `Array` element is the output value corresponding to an input value of 1.0.

7.2.2.4 rawHalfs

This attribute's presence indicates that the output array values in the form of unsigned 16-bit integers shall be interpreted as the equivalent bit pattern, half floating-point values.

If this attribute is present, its value shall be `"true"`.

EXAMPLE
To represent the value 1.0, one would use the integer 15360 in the `Array` element because it has the same bit-pattern. This allows the specification of exact half-float values without relying on conversion from decimal text strings.

7.2.3 Elements

7.2.3.1 General

The XML child elements of the `LUT1D` element shall be as specified in Table 17.

Table 17 — XML child elements of LUT1D

Element	Description
`Array`	See Subclause 7.2.3.2.

7.2.3.2 LUT1D/Array

The `Array` element contains the table of numeric values that represent the output values of the 1D or 3×1D LUT.

The values in the array shall be normalized by dividing by the scale factor in Table 6 for the corresponding `outBitDepth` attribute. The `inBitDepth` attribute is not used.

The XML attributes of `LUT1D/Array` shall be as specified in Table 18.

Table 18 — XML attributes of `LUT1D/Array`

Attribute	Description
`dim`	Consists of two integers that represent the number of rows and columns of the array. The first value defines the length of the array and shall equal the number of entries (lines) in the LUT. The second value indicates the number of components per entry and shall equal 1 for a 1D LUT or 3 for a 3×1D LUT.

EXAMPLE 1
`dim="1024 3"` indicates a 1024 element 1D LUT with 3-component color (a 3×1D LUT)

EXAMPLE 2
`dim="256 1"` indicates a 256 element 1D LUT with 1-component color (a 1D LUT)

7.3 LUT3D

7.3.1 General

The `LUT3D` element shall be as defined in Table 19.

Table 19 — `LUT3D` definition

```
<xs:element name="LUT3D" type="clf:LUT3DType" substitutionGroup="clf:ProcessNode"/>
<xs:complexType name="LUT3DType">
  <xs:complexContent>
    <xs:extension base="clf:ProcessNodeType">
      <xs:sequence>
        <xs:element name="Array" type="clf:ArrayType"/>
      </xs:sequence>
      <xs:attribute name="interpolation" use="optional" default="trilinear">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="trilinear"/>
            <xs:enumeration value="tetrahedral"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A LUT3D transform uses the three color components of an input value to find the nearest indexed values along each axis of the 3D cube defined in a 3D LUT. A 3-component output value is calculated by interpolating within the volume defined by the nearest corresponding positions in the LUT.

EXAMPLE
A simple LUT3D.

```
<LUT3D name="green look" interpolation="trilinear" inBitDepth="12i" outBitDepth="16f">
     <Description>3D LUT</Description>
          <Array dim="2 2 2 3">
               0.0  0.0  0.0
               0.0  0.0  1.0
               0.0  1.0  0.0
               0.0  1.0  1.0
               1.0  0.0  0.0
               1.0  0.0  1.0
               1.0  1.0  0.0
               1.0  1.0  1.0
          </Array>
</LUT3D>
```

7.3.2 Attributes

7.3.2.1 General

The XML attributes of `LUT3D` shall be as specified in Table 20.

Table 20 — XML attributes of `LUT3D`

Attribute	Description
`interpolation`	See Subclause 7.3.2.2

7.3.2.2 interpolation

This attribute is a string indicating the preferred algorithm used to interpolate values in the 3D LUT. This attribute is optional and shall default to `"trilinear"` if the attribute is not present. Supported values are as specified in Table 21.

Table 21 — Supported values for `LUT3D` interpolation attribute

interpolation	Description
`trilinear`	Indicates that the 3D LUT shall be interpolated using the trilinear interpolation method. Trilinear interpolation is described in A.4.
`tetrahedral`	Indicates that the 3D LUT shall be interpolated using the tetrahedral interpolation method. Tetrahedral interpolation is described in A.5.

7.3.3 Elements

7.3.3.1 General

The XML child elements of the `LUT3D` element shall be as specified in Table 22.

Table 22 — XML child elements of `LUT3D`

Element	Description
`Array`	Contains an array of numeric values that are the output values of the 3D LUT. The table entries should be ordered from the minimum to the maximum input values, with the third component index changing fastest.

7.3.3.2 LUT3D/Array

The `Array` element contains the table of numeric values that represent the values of the 3D LUT.

The values in the array shall be normalized by dividing by the scale factor in Table 6 for the corresponding `outBitDepth` attribute. The `inBitDepth` attribute is not used.

The XML attributes of LUT3D/Array shall be as specified in Table 23.

Table 23 — XML attributes of `LUT3D/Array`

Attribute	Description
`dim`	Consists of four integers that represent the dimensions of the 3D LUT and the number of color components. The first three values shall define the axial dimensions of the LUT and if multiplied shall equal the length (i.e., number of entries) of the array. Only 3D LUTs of equal dimensions are supported and so the first three values shall be equal. The fourth value indicates the number of components per entry and shall always equal 3.

EXAMPLE
`dim="17 17 17 3"` indicates a 17-cubed 3D Look-Up Table with 3-component color

7.4 Matrix

7.4.1 General

The `Matrix` element shall be as defined in Table 24.

Table 24 — `Matrix` definition

```
<xs:element name="Matrix" type="clf:MatrixType" substitutionGroup="clf:ProcessNode"/>
<xs:complexType name="MatrixType">
  <xs:complexContent>
    <xs:extension base="clf:ProcessNodeType">
      <xs:sequence>
        <xs:element name="Array" type="clf:MatrixArrayType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A `Matrix` transform specifies a 3×3 or 3×4 matrix transformation to be applied to the input values. The input and output of `Matrix` are always 3-component values.

The output values are calculated using row-order matrix convention, as in Formula (7.1).

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} r_1 \\ g_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} r_2 \\ g_2 \\ b_2 \end{bmatrix} \tag{7.1}$$

Output values for a 3×3 matrix shall be calculated as in Formula (7.2).

$$\begin{aligned} r_2 &= (r_1 \cdot a_{11}) + (g_1 \cdot a_{12}) + (b_1 \cdot a_{13}) \\ g_2 &= (r_1 \cdot a_{21}) + (g_1 \cdot a_{22}) + (b_1 \cdot a_{23}) \\ b_2 &= (r_1 \cdot a_{31}) + (g_1 \cdot a_{32}) + (b_1 \cdot a_{33}) \end{aligned} \tag{7.2}$$

Matrices using an offset calculation will have one more column than rows. An offset matrix may be defined using a 3×4 array, wherein the fourth column is used to specify offset terms, k_1, k_2, k_3, as in Formula (7.3).

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & k_1 \\ a_{21} & a_{22} & a_{23} & k_2 \\ a_{31} & a_{32} & a_{33} & k_3 \end{bmatrix} \begin{bmatrix} r_1 \\ g_1 \\ b_1 \\ 1.0 \end{bmatrix} = \begin{bmatrix} r_2 \\ g_2 \\ b_2 \end{bmatrix} \tag{7.3}$$

In the case of a 3×4 matrix, the offset terms k_1, k_2, and k_3 are added to each of the normal matrix calculations, and output value shall be calculated as in Formula (7.4).

$$\begin{aligned} r_2 &= (r_1 \cdot a_{11}) + (g_1 \cdot a_{12}) + (b_1 \cdot a_{13}) + k_1 \\ g_2 &= (r_1 \cdot a_{21}) + (g_1 \cdot a_{22}) + (b_1 \cdot a_{23}) + k_2 \\ b_2 &= (r_1 \cdot a_{31}) + (g_1 \cdot a_{32}) + (b_1 \cdot a_{33}) + k_3 \end{aligned} \tag{7.4}$$

EXAMPLE 1

`Matrix` node with `dim="3 3"`.

```
<Matrix name="AP0 to AP1" inBitDepth="16f" outBitDepth="16f">
     <Description>3×3 color space conversion from AP0 to AP1</Description>
     <Array dim="3 3">
         1.45143931614567     -0.236510746893740    -0.214928569251925
        -0.0765537733960204    1.17622969983357     -0.0996759264375522
         0.00831614842569772  -0.00603244979102103   0.997716301365324
     </Array>
</Matrix>
```

EXAMPLE 2

`Matrix` node with `dim="3 4"`.

```
<Matrix name="colorspace conversion" inBitDepth="10i" outBitDepth="10i">
     <Description>3×4 Matrix, 4th column is offset</Description>
     <Array dim="3 4">
         1.2      0.0     0.0      0.002
         0.0      1.03    0.001   -0.005
         0.004   -0.007   1.004    0.0
     </Array>
</Matrix>
```

7.4.2 Attributes

The XML attributes of `Matrix` shall be as specified in Table 25.

Table 25 — XML attributes of `Matrix`

Attribute	Description
`dim`	Consists of two integers that describe the dimensions of the matrix array. The first value shall match the number of rows and be equal to 3. The second value shall match the number of columns and be equal to 3 or 4.

EXAMPLE 1
`dim="3 3"` indicates a 3×3 matrix

EXAMPLE 2
`dim="3 4"` indicates a 3×4 matrix

7.4.3 Elements

The XML child elements of the `Matrix` element shall be as specified in Table 26.

The matrix coefficients in the array shall be normalized by multiplying by the scale factor in Table 6 for the corresponding `inBitDepth` attribute and dividing by the scale factor in Table 6 for the corresponding `outBitDepth` attribute.

The matrix offset values in the array, if present, shall be normalized by dividing by the scale factor in Table 6 for the corresponding `outBitDepth` attribute. The `inBitDepth` attribute is not used.

Table 26 — XML child elements of `Matrix`

Element	Description
`Array`	Contains the coefficients of the transformation matrix. The matrix shall be serialized row by row from left to right and from top to bottom.

EXAMPLE
A 3×3 matrix written as:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

would be serialized "$a_{11}\ a_{12}\ a_{13}\ a_{21}\ a_{22}\ a_{23}\ a_{31}\ a_{32}\ a_{33}$".

7.5 Range

7.5.1 General

The Range element shall be as defined in Table 27.

Table 27 — Range definition

```
<xs:element name="Range" type="clf:RangeType" substitutionGroup="clf:ProcessNode"/>
<xs:complexType name="RangeType">
  <xs:complexContent>
    <xs:extension base="clf:ProcessNodeType">
      <xs:sequence>
        <xs:element name="minInValue" type="xs:float" minOccurs="0"/>
        <xs:element name="maxInValue" type="xs:float" minOccurs="0"/>
        <xs:element name="minOutValue" type="xs:float" minOccurs="0"/>
        <xs:element name="maxOutValue" type="xs:float" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="style" use="optional" default="Clamp">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Clamp"/>
            <xs:enumeration value="noClamp"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A Range node maps the input domain to the output range by scaling and offsetting values and clamping values outside the output range, although the clamping step is optional. The Range node can also be used to clamp values at the lower end only, upper end only, or both.

7.5.2 Attributes

7.5.2.1 General

The XML attributes of Range shall be as specified in Table 28.

Table 28 — XML attributes of Range

Attribute	Description
style	See Subclause 7.5.2.2

7.5.2.2 style

This attribute describes the preferred handling of the scaling calculation used by the `Range` node. The default shall be `"Clamp"` if the style attribute is not present. The options for style are listed in Table 29.

Table 29 — Supported values for `Range` style attribute

style	Description
`Clamp`	If each of `minInValue`, `minOutValue`, `maxInValue`, and `maxOutValue` is provided, a scale and offset with clamping shall be applied according to Formula (7.6). When the `Range` node is used for clamping only, only one pair of in/out values is required: If only `minInValue` and `minOutValue` are provided, then clamping shall be performed at the low end according to Formula (7.8). If only `maxInValue` and `maxOutValue` are provided, then clamping shall be performed at the high end according to Formula (7.9).
`noClamp`	Indicates that a scale and offset shall be applied according to Formula (7.5).

$$y = Y_{min} + S \cdot (x - X_{min}) \tag{7.5}$$

where

y is the output value

x is the input value

S is the scale factor, as in Formula (7.7)

Y_{min} is `minOutValue`, normalized as defined in 7.5.3

X_{min} is `minInValue`, normalized as defined in 7.5.3

$$y_{clamp} = \mathrm{MIN}\left(Y_{max}, \mathrm{MAX}\left(Y_{min}, Y_{min} + S \cdot (x - X_{min})\right)\right) \tag{7.6}$$

where

y_{clamp} is the clamped output value

x is the input value

S is the scale factor, as in Formula (7.7)

Y_{max} is `maxOutValue`, normalized as defined in 7.5.3

Y_{min} is `minOutValue`, normalized as defined in 7.5.3

X_{min} is `minInValue`, normalized as defined in 7.5.3

$\mathrm{MAX}(a, b)$ returns the value of a if $a > b$ and b if $b \geq a$

$\mathrm{MIN}(a, b)$ returns the value of a if $a < b$ and b if $b \leq a$

$$S = \frac{(Y_{max} - Y_{min})}{(X_{max} - X_{min})} \tag{7.7}$$

where

S is the scale factor, used in Formula (7.6)

Y_{max} is `maxOutValue`, normalized as defined in 7.5.3

Y_{min} is `minOutValue`, normalized as defined in 7.5.3

X_{max} is `maxInValue`, normalized as defined in 7.5.3

X_{min} is `minInValue`, normalized as defined in 7.5.3

$$y_{clamp} = \mathrm{MAX}(Y_{min}, x) \tag{7.8}$$

where

y_{clamp} is the clamped output value

x is the input value

Y_{min} is `minOutValue`, normalized as defined in 7.5.3

$\mathrm{MAX}(a, b)$ returns the value of a if $a > b$ and b if $b \geq a$

$$y_{clamp} = \mathrm{MIN}(Y_{max}, x) \tag{7.9}$$

where

y_{clamp} is the clamped output value

x is the input value

Y_{max} is `maxOutValue`, normalized as defined in 7.5.3

$\mathrm{MIN}(a, b)$ returns the value of a if $a < b$ and b if $b \leq a$

EXAMPLE
Using `Range` for scaling 10-bit full range to 10-bit SMPTE (legal) range.

```
<Range inBitDepth="10i" outBitDepth="10i">
    <Description>10-bit full range to SMPTE range</Description>
    <minInValue>0</minInValue>
    <maxInValue>1023</maxInValue>
    <minOutValue>64</minOutValue>
    <maxOutValue>940</maxOutValue>
</Range>
```

7.5.3 Elements

The XML child elements of the `Range` element shall be as specified in Table 30 and shall be ordered as defined in Table 27.

The `minInValue` and `maxInValue` shall be normalized by dividing by the scale factor in Table 6 for the corresponding `inBitDepth` attribute.

The `minOutValue` and `maxOutValue` shall be normalized by dividing by the scale factor in Table 6 for the corresponding `outBitDepth` attribute.

At least one pair of either minimum or maximum values, or all four values, shall be provided.

Table 30 — XML child elements of `Range`

Element	Description
`minInValue`	The minimum input value. Required if `minOutValue` is present.
`maxInValue`	The maximum input value. Required if `maxOutValue` is present. The `maxInValue` shall be greater than the `minInValue`.
`minOutValue`	The minimum output value. Required if `minInValue` is present.
`maxOutValue`	The maximum output value. Required if `maxInValue` is present. The `maxOutValue` shall be greater than or equal to the `minOutValue`.

7.6 Log

7.6.1 General

The `Log` element shall be as defined in Table 31.

Table 31 — `Log` definition

```
<xs:element name="Log" type="clf:LogType" substitutionGroup="clf:ProcessNode"/>
<xs:complexType name="LogType">
  <xs:complexContent>
    <xs:extension base="clf:ProcessNodeType">
      <xs:sequence>
        <xs:element name="LogParams" type="clf:LogParamsType" minOccurs="0"
                    maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="style" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="log10"/>
            <xs:enumeration value="antiLog10"/>
            <xs:enumeration value="log2"/>
            <xs:enumeration value="antiLog2"/>
            <xs:enumeration value="linToLog"/>
```

```
            <xs:enumeration value="logToLin"/>
            <xs:enumeration value="cameraLinToLog"/>
            <xs:enumeration value="cameraLogToLin"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="LogParamsType">
  <xs:attribute name="base" type="xs:float" use="optional" default="2"/>
  <xs:attribute name="logSideSlope" type="xs:float" use="optional" default="1.0"/>
  <xs:attribute name="logSideOffset" type="xs:float" use="optional" default="0.0"/>
  <xs:attribute name="linSideSlope" type="xs:float" use="optional" default="1.0"/>
  <xs:attribute name="linSideOffset" type="xs:float" use="optional" default="0.0"/>
  <xs:attribute name="linSideBreak" type="xs:float" use="optional" default="0.0"/>
  <xs:attribute name="linearSlope" type="xs:float" use="optional"/>
  <xs:attribute name="channel" type="clf:channelType" use="optional"/>
</xs:complexType>
```

A Log node contains parameters for processing color tuples through one of a few different styles of logarithmic or anti-logarithmic functions. The most basic formula follows a pure logarithm or anti-logarithm of either base 2 or base 10. Also supported are a logarithmic function with a gain factor and offset or a piecewise function that joins a logarithmic function with a gain factor and offset with a linear segment.

EXAMPLE 1
Log node applying a base 10 logarithm.

```
<Log inBitDepth="16f" outBitDepth="16f" style="log10">
     <Description>Base 10 Logarithm</Description>
</Log>
```

EXAMPLE 2
Log node applying a camera log formula.

```
<Log inBitDepth="32f" outBitDepth="32f" style="cameraLinToLog">
     <Description>Linear to Log</Description>
     <LogParams base="10" logSideSlope="0.256663" logSideOffset="0.584555"
        linSideSlope="0.9892" linSideOffset="0.0108" linSideBreak="0.0078"
        linearSlope="6.025"/>
</Log>
```

7.6.2 Attributes

7.6.2.1 General

The XML attributes of `Log` shall be as specified in Table 32.

Table 32 — XML attributes of `Log`

Attribute	Description
`style`	See Subclause 7.6.2.2

7.6.2.2 style

The `style` attribute shall specify the form of the log function to be applied. Supported values for `style` and the formula each shall follow as specified in Table 33.

Table 33 — Supported values for `Log` style attribute

style	Description
`log10`	Indicates that a base 10 logarithm shall be applied according to Formula (7.10)
`antiLog10`	Indicates that a base 10 anti-logarithm shall be applied according to Formula (7.11)
`log2`	Indicates that a base 2 logarithm shall be applied according to Formula (7.12)
`antiLog2`	Indicates that a base 2 anti-logarithm shall be applied according to Formula (7.13)
`linToLog`	Indicates that a logarithm with gain and offset shall be applied according to Formula (7.14).
`logToLin`	Indicates that an anti-logarithm with gain and offset shall be applied according to Formula (7.15).
`cameraLinToLog`	Indicates that a piecewise function, joining a logarithm with gain and offset with a linear segment, shall be applied according to Formula (7.16).
`cameraLogToLin`	Indicates that a piecewise function, joining an anti-logarithm with gain and offset with a linear segment, shall be applied according to Formula (7.17).

The following definitions apply for formulae in 7.6.2.2.

$\mathrm{FLT\_MIN}$ is equal to $1.175494e^{-38}$

x is the input value

y is the output value

$\mathrm{MAX}(a, b)$ returns the value of a if $a > b$ and b if $b \geq a$

NOTE For formulae (7.10) to (7.20), any integer values have been normalized as specified in Subclause 6.3. LogParams do not change based on the input and output bit depths.

$$y = log_{10}\big(\mathrm{MAX}(x, \mathrm{FLT\_MIN})\big) \tag{7.10}$$

$$y = 10^{x} \tag{7.11}$$

$$y = log_{2}\big(\mathrm{MAX}(x, \mathrm{FLT\_MIN})\big) \tag{7.12}$$

$$y = 2^{x} \tag{7.13}$$

$$y = m_{log} \times \log_{n}\big(\mathrm{MAX}(m_{lin} \times x + b_{lin}, \mathrm{FLT\_MIN})\big) + b_{log} \tag{7.14}$$

where

m_{log} is the value of `logSideSlope`

n is the base of the logarithm

m_{lin} is the value of `linSideSlope`

b_{lin} Is the value of `linSideOffset`

b_{log} Is the value of `logSideOffset`

$$y = \frac{\left(n^{\left(\frac{x - b_{log}}{m_{log}}\right)} - b_{lin}\right)}{m_{lin}} \tag{7.15}$$

where

m_{log} is the value of `logSideSlope`

n is the base of the logarithm

m_{lin} is the value of `linSideSlope`

b_{lin} is the value of `linSideOffset`

b_{log} is the value of `logSideOffset`

$$y = \begin{cases} M \times x + B & \text{if } x \leq J \\ y = m_{log} \times \log_n\left(\text{MAX}(m_{lin} \times x + b_{lin}, \text{FLT\_MIN})\right) + b_{log} & \text{otherwise} \end{cases} \quad (7.16)$$

where

M is the value of `linearSlope`, as specified in Formula (7.19)

B is the value of `linearOffset`, as specified in Formula (7.20)

J is the value of `linSideBreak`

m_{log} is the value of `logSideSlope`

n is the base of the logarithm

m_{lin} is the value of `linSideSlope`

b_{lin} is the value of `linSideOffset`

b_{log} is the value of `logSideOffset`

$$y = \begin{cases} \dfrac{(x - B)}{M} & \text{if } x \leq K \\ \dfrac{\left(n^{\left(\frac{x - b_{log}}{m_{log}}\right)} - b_{lin}\right)}{m_{lin}} & \text{otherwise} \end{cases} \quad (7.17)$$

where

M is the value of `linearSlope`, as specified in Formula (7.19)

B is the value of `linearOffset`, as specified in Formula (7.20)

K is the value of `logSideBreak`, as specified in Formula (7.18)

m_{log} is the value of `logSideSlope`

n is the base of the logarithm

m_{lin} is the value of `linSideSlope`

b_{lin} is the value of `linSideOffset`

b_{log} is the value of `logSideOffset`

7.6.3 Elements

7.6.3.1 General

The XML child elements of the `Log` element shall be as specified in Table 34.

Table 34 — XML child elements of `Log`

Element	Description
`LogParams`	Contains the attributes that control the `"linToLog"`, `"logToLin"`, `"cameraLinToLog"`, or `"cameraLogToLin"` styles and shall be required if style is any of these four types (i.e., style is not that of a simple base 10 or base 2 logarithm). There shall be either one or three `LogParams` child elements. If one `LogParams` child element is present, it shall be applied to all elements of a color tuple. The same value of base shall be used for all channels. If three `LogParams` child elements are present, they shall be applied to the elements R,G,B of a color tuple according to the value of the respective `channel` attribute.

7.6.3.2 Attributes of Log/LogParams

The XML attributes of `Log/LogParams` shall be as specified in Table 35.

Table 35 — XML attributes of `Log/LogParams`

Attribute	Description	Default Value
`base`	The logarithmic base of the function. The base value shall be identical for all three channels if the channel attribute is used to provide channel-specific values for the other attributes.	2
`logSideSlope`	The gain (or slope) applied to the log side of the function.	1.0
`logSideOffset`	The offset applied to the log side of the function.	0.0
`linSideSlope`	The gain (or slope) applied to the linear side of the function.	1.0
`linSideOffset`	The offset applied to the linear side of the function.	0.0
`linSideBreak`	The break point, defined in linear space, at which the piecewise function transitions between the logarithmic and linear segments. It shall be required if `style` is `"cameraLinToLog"` or `"cameraLogToLin"`.	0.0
`linearSlope`	The slope of the linear segment of the piecewise function. If `linearSlope` is not present, it shall be calculated as specified in 7.6.3.4.	n/a
`channel`	Specifies the channel to which the logarithmic function shall be applied. If present, the value shall be one of "R", "G", or "B".	n/a

7.6.3.3 Solving for `LogSideBreak`

The value of the break point on the log-axis is calculated according to Formula (7.18) using the value of `linSideBreak` as input to the logarithmic segment of the piecewise function.

$$K = m_{log} \times \log_n(m_{lin} \times J + b_{lin}) + b_{log} \tag{7.18}$$

where

K is the value of `logSideBreak`

J is the value of `linSideBreak`

m_{log} is the value of `logSideSlope`

n is the base of the logarithm

m_{lin} is the value of `linSideSlope`

b_{lin} is the value of `linSideOffset`

b_{log} is the value of `logSideOffset`

7.6.3.4 Solving for `linearSlope`

If the value of `linearSlope` was not provided as one of the attributes, its default value shall be calculated according to Formula (7.19).

NOTE The value of `linearSlope` is calculated using the `linSideBreak` attribute to assure that the linear portion remains continuous with the logarithmic portion of the curve. The value of `linSideBreak` is used to solve for the derivative of the logarithmic function and `linearSlope` is then set to be equal to the instantaneous slope, or derivative, at the break point.

$$M = m_{log} \times \left[\frac{m_{lin}}{(m_{lin} \times J + b_{lin}) \times \ln(n)}\right] \tag{7.19}$$

where

M is the value of `linearSlope`

J is the value of `linSideBreak`

m_{log} is the value of `logSideSlope`

n is the base of the logarithm

m_{lin} is the value of `linSideSlope`

b_{lin} is the value of `linSideOffset`

7.6.3.5 Solving for `linearOffset`

The value of `linearOffset` shall be calculated according to Formula (7.20).

$$B = K - M \times J \tag{7.20}$$

where

B is the value of `linearOffset`

M is the value of `linearSlope`, as specified in Formula (7.19)

J is the value of `linSideBreak`

K is the value of `logSideBreak`, as specified in Formula (7.18)

7.7 Exponent

7.7.1 General

The `Exponent` element shall be as defined in Table 36.

Table 36 — `Exponent` definition

```
<xs:element name="Exponent" type="clf:ExponentType"
                          substitutionGroup="clf:ProcessNode"/>
<xs:complexType name="ExponentType">
  <xs:complexContent>
    <xs:extension base="clf:ProcessNodeType">
      <xs:sequence>
        <xs:element name="ExponentParams" type="clf:ExponentParamsType"
                          maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="style" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="basicFwd"/>
            <xs:enumeration value="basicRev"/>
            <xs:enumeration value="basicMirrorFwd"/>
            <xs:enumeration value="basicMirrorRev"/>
            <xs:enumeration value="basicPassThruFwd"/>
            <xs:enumeration value="basicPassThruRev"/>
            <xs:enumeration value="monCurveFwd"/>
            <xs:enumeration value="monCurveRev"/>
            <xs:enumeration value="monCurveMirrorFwd"/>
            <xs:enumeration value="monCurveMirrorRev"/>
          </xs:restriction>
```

```
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ExponentParamsType">
  <xs:attribute name="exponent" type="xs:float" use="required"/>
  <xs:attribute name="offset" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:float">
        <xs:minInclusive value="0."/>
        <xs:maxInclusive value="0.9"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="channel" type="clf:channelType" use="optional"/>
</xs:complexType>
```

This node contains parameters for processing color tuples through a power law function. Two main formulations are supported. The first follows a pure power law. The second is a piecewise function that follows a power function for larger values and follows a linear segment for small and negative values.

EXAMPLE 1
Using Exponent node for applying a 2.2 gamma.

```
<Exponent inBitDepth="32f" outBitDepth="32f" style="basicFwd">
     <Description>Basic 2.2 Gamma</Description>
     <ExponentParams exponent="2.2"/>
</Exponent>
```

EXAMPLE 2
Using Exponent node for applying the intended EOTF found in IEC 61966-2-1:1999 (sRGB).

```
<Exponent inBitDepth="32f" outBitDepth="32f" style="monCurveFwd">
     <Description>EOTF (sRGB)</Description>
     <ExponentParams exponent="2.4" offset="0.055"/>
</Exponent>
```

EXAMPLE 3
Using Exponent node to apply CIE L* formula.

```
<Exponent inBitDepth="32f" outBitDepth="32f" style="monCurveRev">
     <Description>CIE L*</Description>
     <ExponentParams exponent="3.0" offset="0.16"/>
</Exponent>
```

EXAMPLE 4
Using Exponent node to apply Rec. 709 OETF.

```
<Exponent inBitDepth="32f" outBitDepth="32f" style="monCurveRev">
     <Description>Rec. 709 OETF</Description>
     <ExponentParams exponent="2.2222222222222222" offset="0.099"/>
</Exponent>
```

7.7.2 Attributes

7.7.2.1 General

The XML attributes of `Exponent` shall be as specified in Table 37.

Table 37 — XML attributes of `Exponent`

Attribute	Description
`style`	See Subclause 7.7.2.2

7.7.2.2 style

The `style` attribute shall specify the form of the exponential function to be applied. Supported values for `style` are defined in Table 38.

Table 38 — Supported values for `Exponent` style attribute

style	Description
`basicFwd`	This attribute shall apply a power law using the exponent value specified in the `ExponentParams` element, as in Formula (7.22). Values less than zero shall be clamped.
`basicRev`	This attribute shall apply a power law using the exponent value specified in the `ExponentParams` element, as in Formula (7.23). Values less than zero shall be clamped.
`basicMirrorFwd`	This attribute shall apply a basic power law using the exponent value specified in the `ExponentParams` element for values greater than or equal to zero and mirror that function for values less than zero (i.e., rotationally symmetric around the origin), as in Formula (7.24).
`basicMirrorRev`	This attribute shall apply a basic power law using the exponent value specified in the `ExponentParams` element for values greater than or equal to zero and mirror that function for values less than zero (i.e., rotationally symmetric around the origin), as in Formula (7.25).
`basicPassThruFwd`	This attribute shall apply a basic power law using the exponent value specified in the `ExponentParams` element for values greater than or equal to zero and pass values less than zero unchanged, as in Formula (7.26).
`basicPassThruRev`	This attribute shall apply a basic power law using the exponent value specified in the `ExponentParams` element for values greater than or equal to zero and pass values less than zero unchanged, as in Formula (7.27).
`monCurveFwd`	This attribute shall apply a power law function with a linear segment near the origin, as in Formula (7.28).

style	Description
`monCurveRev`	This attribute shall apply a power law function with a linear segment near the origin, as in Formula (7.29).
`monCurveMirrorFwd`	This attribute shall apply a power law function with a linear segment near the origin and mirror that function for values less than zero (i.e., rotationally symmetric around the origin), as in Formula (7.30).
`monCurveMirrorRev`	This attribute shall apply a power law function with a linear segment near the origin and mirror that function for values less than zero (i.e., rotationally symmetric around the origin), as in Formula (7.31).

NOTE 1 The "`monCurve`"-style formulae can be used to represent the Rec. 709, sRGB, and CIE L* formulae.

The following definitions apply for all style formulae (7.22) through (7.31).

y is the output value

x is the input value

g is the exponent

k is the offset

$\mathrm{MAX}(a, b)$ returns the value of a if $a > b$ and b if $b \geq a$

s is a scalar as specified in Formula (7.21)

$$s = \left(\frac{g-1}{k}\right)\left(\frac{kg}{(g-1)(1+k)}\right)^{g} \tag{7.21}$$

$$y = [\mathrm{MAX}(0, x)]^{g} \tag{7.22}$$

$$y = [\mathrm{MAX}(0, x)]^{1/g} \tag{7.23}$$

$$y = \begin{cases} x^{g} & \text{if } x \geq 0 \\ -[(-x)^{g}] & \text{otherwise} \end{cases} \tag{7.24}$$

$$y = \begin{cases} x^{1/g} & \text{if } x \geq 0 \\ -\left[(-x)^{1/g}\right] & \text{otherwise} \end{cases} \tag{7.25}$$

$$y = \begin{cases} x^{g} & \text{if } x \geq 0 \\ x & \text{otherwise} \end{cases} \tag{7.26}$$

$$y = \begin{cases} x^{1/g} & \text{if } x \geq 0 \\ x & \text{otherwise} \end{cases} \tag{7.27}$$

$$\mathrm{y} = \begin{cases} \left(\dfrac{x+k}{1+k}\right)^{g} & \text{if } x \geq B_x \\ x \times s & \text{otherwise} \end{cases} \tag{7.28}$$

where

B_x is the x coordinate of the breakpoint equal to $\dfrac{k}{g-1}$

NOTE 2 An exponent value of 1.0 and/or an offset value of 0.0 could result in a divide-by-zero error.

$$y = \begin{cases} (1+k) \times x^{(1/g)} - k & \text{if } x \geq B_y \\ \dfrac{x}{s} & \text{otherwise} \end{cases} \tag{7.29}$$

where

B_y is the y coordinate of the breakpoint equal to $\left(\dfrac{kg}{(g-1)(1+k)}\right)^{g}$

$$y = \begin{cases} \mathrm{C}(x) & \text{if } x \geq 0 \\ -[\mathrm{C}(-x)] & \text{otherwise} \end{cases} \tag{7.30}$$

where

$\mathrm{C}(x)$ is the `monCurveFwd` defined in Formula (7.28)

$$y = \begin{cases} V(x) & \text{if } x \geq 0 \\ -[V(-x)] & \text{otherwise} \end{cases} \tag{7.31}$$

where

$\mathrm{V}(x)$ is the `monCurveRev` defined in Formula (7.29)

NOTE 3 For formulae (7.22) through (7.31), any Integer values have been normalized as specified in Subclause 6.3.

7.7.3 Elements

7.7.3.1 General

The XML child elements of the `Exponent` element shall be as specified in Table 39.

Table 39 — XML child elements of `Exponent`

Element	Description
`ExponentParams`	There shall be one or three `ExponentParams` child elements. If one `ExponentParams` child element is present, it shall be applied to all components of a color tuple. If three `ExponentParams` child elements are present, they shall be applied to the components R,G,B of a color tuple according to the value of the respective `channel` attribute.

7.7.3.2 Attributes of Exponent/ExponentParams

The XML attributes of `Exponent/ExponentParams` shall be as specified in Table 40.

Table 40 — XML attributes of `Exponent/ExponentParams`

Attribute	Description
`exponent`	Represents the power to which the input value is to be raised. If style is any of the "`monCurve`" types, the valid range is [1.0, 10.0]. The nominal value is 1.0.
`offset`	Represents the offset value to use and shall be used only when `style` is any of the "`monCurve`" types. The valid range is [0.0, 0.9]. This attribute shall not be allowed when `style` is any of the "`basic`" types.
`channel`	Specifies the color channel to which the exponential function shall be applied. Possible values are "R", "G", "B".

7.8 ASC_CDL

7.8.1 General

The ASC_CDL element shall be as defined in Table 41.

Table 41 — ASC_CDL definition

```
<xs:element name="ASC_CDL" type="clf:ASC_CDLType"
                          substitutionGroup="clf:ProcessNode"/>
<xs:complexType name="ASC_CDLType">
  <xs:complexContent>
    <xs:extension base="clf:ProcessNodeType">
      <xs:sequence>
        <xs:element name="SOPNode" type="clf:SOPNodeType" minOccurs="0"/>
        <xs:element name="SatNode" type="clf:SatNodeType" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="style" default="Fwd">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Fwd"/>
            <xs:enumeration value="Rev"/>
            <xs:enumeration value="FwdNoClamp"/>
            <xs:enumeration value="RevNoClamp"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="SOPNodeType">
  <xs:sequence>
    <xs:element name="Slope" type="clf:nonNegativeFloatListLength3Type" default="1.0
                          1.0 1.0"/>
    <xs:element name="Offset" type="clf:floatListLength3Type" default="0.0 0.0 0.0"/>
    <xs:element name="Power" type="clf:positiveFloatListLength3Type" default="1.0 1.0
                          1.0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SatNodeType">
  <xs:sequence>
    <xs:element name="Saturation" type="clf:nonNegativeFloatType" minOccurs="0"
                          default="1.0"/>
  </xs:sequence>
</xs:complexType>
```

An ASC_CDL node processes values according to the American Society of Cinematographers' Color Decision List (ASC CDL) formulae.

The ASC CDL formulae are an industry-wide method of recording and exchanging basic color correction adjustments via parameters that set particular color processing formulae. They are designed to work on an input domain of floating-point values of range [0.0, 1.0], although values greater than 1.0 can be present. The output data might be clamped depending on the processing style used.

EXAMPLE
A simple ASC_CDL node.

```
<ASC_CDL inBitDepth="16f" outBitDepth="16f" style="Fwd">
      <Description>scene 1 exterior look</Description>
      <SOPNode id="urn:uuid:29b9db77-ac19-4f44-b88b-1a5c6b595323">
            <Slope>1.000000 1.000000 0.900000</Slope>
            <Offset>-0.030000 -0.020000 0.000000</Offset>
            <Power>1.2500000 1.000000 1.000000</Power>
      </SOPNode>
      <SatNode id="urn:uuid:069df9aa-2167-4e0b-8ec3-c00c53ea9658">
            <Saturation>1.700000</Saturation>
      </SatNode>
</ASC_CDL>
```

7.8.2 Attributes

7.8.2.1 General

The XML attributes of the ASC_CDL element shall be as specified in Table 42.

Table 42 — XML attributes of ASC_CDL

Attribute	Description
style	See Subclause 7.8.2.2

7.8.2.2 style

The style attribute determines the formula to be applied by the operator and shall be one of the options specified in Table 43.

Table 43 — Supported values for ASC_CDL style attribute

style	Description
Fwd	Indicates that the standard forward ASC CDL formulae with clamping shall be applied according to Formula (7.33) followed by Formula (7.34).
Rev	Indicates that the standard reverse ASC CDL formulae with clamping shall be applied according to Formula (7.37) followed by Formula (7.38).
FwdNoClamp	Indicates that the alternate ASC CDL formulae with no clamping shall be applied according to Formula (7.35) followed by Formula (7.36).
RevNoClamp	Indicates that the alternate ASC CDL formulae with no clamping shall be applied according to Formula (7.39) followed by Formula (7.40).

Formula (7.32) defines the generic computation of a luma value, l, which is referenced in Formulae (7.34), (7.37), (7.36), and (7.39). The value of l shall be computed within each formula, using the input value specified in the formula where it is used.

$$l = 0.2126 \times R + 0.7152 \times G + 0.0722 \times B \tag{7.32}$$

where

l is luma

R is the red color component value of the input color tuple

G is the green color component value of the input color tuple

B is the blue color component value of the input color tuple

$$y_c = \mathrm{CLAMP}(x \times s + o)^p \tag{7.33}$$

where

y_c is the output value, the clamped result of the v1.2 ASC CDL SOP formula

x is the input value, one of the components of the input color tuple

s is the value of `Slope`, as defined in 7.8.3.2

o is the value of `Offset`, as defined in 7.8.3.2

p is the value of `Power`, as defined in 7.8.3.2

$\mathrm{CLAMP}(a)$ clamps the value of a to the range [0, 1] using the formula $\mathrm{MAX}\big(0, \mathrm{MIN}(x, 1)\big)$

NOTE 1 `Slope` is similar to gain, but changes the slope of the transfer function without shifting the black level established by offset. `Offset` raises or lowers the overall brightness of a color component by shifting the transfer function up or down while keeping the slope unchanged. `Power` controls the intermediate shape of the transfer function.

$$z_c = \mathrm{CLAMP}[l + t \times (y_c - l)] \tag{7.34}$$

where

z_c is the output value, the clamped result of v1.2 ASC CDL saturation formula

y_c is the clamped result from Formula (7.33)

l is luma, calculated using y_c as input to Formula (7.32)

t is the value of `Saturation`, as defined in 7.8.3.3

$\mathrm{CLAMP}(a)$ clamps the value of a to the range [0, 1] using the formula $\mathrm{MAX}\big(0, \mathrm{MIN}(x, 1)\big)$

$$y = \begin{cases} x \times s + o & \text{if } (x \times s + o) < 0 \\ (x \times s + o)^p & \text{otherwise} \end{cases} \tag{7.35}$$

where

y is the output value, the unclamped result of the v1.2 ASC CDL SOP formula

x is the input value, one of the components of the input color tuple

s is the value of `Slope`, as defined in 7.8.3.2

o is the value of `Offset`, as defined in 7.8.3.2

p is the value of `Power`, as defined in 7.8.3.2

$$z = l + t \times (y - l) \tag{7.36}$$

where

z is the output value, the result of v1.2 ASC CDL saturation formula with no clamp

y is the unclamped result from Formula (7.35)

l is luma, calculated using y as input to Formula (7.32)

t is the value of `Saturation`, as defined in 7.8.3.3

$$y_c = l + \frac{(\mathrm{CLAMP}(z) - l)}{t} \tag{7.37}$$

where

y_c is the output value, the clamped result of the reverse v1.2 ASC CDL saturation formula

z is the input value, one of the components of the input color tuple

l is luma, calculated using $\mathrm{CLAMP}(z)$ as input to Formula (7.32)

t is the value of `Saturation`, as defined in 7.8.3.3

$\mathrm{CLAMP}(a)$ clamps the value of a to the range [0, 1] using the formula $\mathrm{MAX}\big(0, \mathrm{MIN}(x, 1)\big)$

$$x_c = \mathrm{CLAMP}\left[\frac{\mathrm{CLAMP}(y_c)^{\frac{1}{p}} - o}{s}\right] \tag{7.38}$$

where

x_c is the output value, the clamped result of the reverse v1.2 ASC CDL SOP formula

y_c is the input value, the clamped result from Formula (7.37)

s is the value of `Slope`, as defined in 7.8.3.2

o is the value of `Offset`, as defined in 7.8.3.2

p is the value of `Power`, as defined in 7.8.3.2

$\mathrm{CLAMP}(a)$ clamps the value of a to the range [0, 1] using the formula $\mathrm{MAX}\big(0, \mathrm{MIN}(x, 1)\big)$

$$y = l + \frac{(z - l)}{t} \tag{7.39}$$

where

y is the output value, the unclamped result of the reverse v1.2 ASC CDL saturation formula

z is the input value, one of the components of the input color tuple

l is luma, calculated according to Formula (7.32)

t is the value of `Saturation`, as defined in 7.8.3.3

$$x = \begin{cases} \dfrac{(y)^{\frac{1}{p}} - o}{s} & \text{if y} < 0 \\ \dfrac{y - o}{s} & \text{otherwise} \end{cases} \tag{7.40}$$

where

x is the output value, the unclamped result of the reverse v1.2 ASC CDL SOP formula

y is the input value, the unclamped result from Formula (7.39)

s is the value of `Slope`, as defined in 7.8.3.2

o is the value of `Offset`, as defined in 7.8.3.2

p is the value of `Power`, as defined in 7.8.3.2

NOTE 2 For formulae (7.32) to (7.40), x and y have been normalized as specified in Subclause 6.3. The values of slope, offset, power, and saturation do not change based on the input and output bit depths.

7.8.3 Elements

7.8.3.1 General

The XML child elements of the `ASC_CDL` element shall be as specified in Table 44 and shall be ordered as given in Table 41. If either element is not specified, values should default to the nominal values for each element.

Table 44 — XML child elements of `ASC_CDL`

Element	Description
`SOPNode`	See Subclause 7.8.3.2
`SatNode`	See Subclause 7.8.3.3

NOTE The structure of this `ProcessNode` matches the structure of the XML format described in the v1.2 ASC CDL specification. However, unlike the ASC CDL XML format, there are no alternate spellings allowed for these elements.

7.8.3.2 SOPNode

If present, `SOPNode` shall contain each of the sub-elements specified in Table 45.

Table 45 — XML child elements of `ASC_CDL/SOPNode`

Element	Description	Nominal (Default) Value
`Slope`	Contains three decimal values representing the R, G, and B slope values. Valid values for slope shall be greater than or equal to zero.	1.0
`Offset`	Contains three decimal values representing the R, G, and B offset values.	0.0
`Power`	Contains three decimal values representing the R, G, and B power values. Valid values for power shall be greater than zero.	1.0

7.8.3.3 SatNode

If present, `SatNode` shall contain the sub-element specified in Table 46.

Table 46 — XML child elements of `ASC_CDL/SatNode`

Element	Description	Nominal (Default) Value
`Saturation`	Contains a decimal value representing the percent saturation to be applied to all color channels. Valid values for saturation shall be greater than or equal to zero.	1.0

8 Profiles

Each Profile shall define one or more constraints on CLF instances.

A CLF instance may conform to a Profile. The Profile should be signaled using the `Profile` element defined in Table 14. Profiles can be specified in other documents of the SMPTE ST 2136 suite or elsewhere.

Each Profile shall define a unique URI as value for the `Profile` element. A CLF Profile specification shall make the `Profile` element mandatory in its schema (i.e., `minOccurs="1" maxOccurs="1"`).

NOTE CLF constraints defined by a Profile can reflect a workflow configuration or business requirements.

9 Implementation Notes (informative)

9.1 Efficient Processing

The transform engine can merge some `ProcessNode` instances in order to obtain better performance. For example, adjacent `Matrix` operators can be combined into a single matrix. However, in general, combining operators in a way that preserves accuracy is difficult and is better avoided.

Hardware implementations can convert all `ProcessNode` instances into some other form that is consistent with what the hardware supports. For example, all `ProcessNode` instances might need to be combined into a single 3D LUT. Using a grid size of 64 or larger can be considered to preserve accuracy. Implementers are reminded that the success of such approximations varies greatly with the nature of the input and output color spaces. For example, if the input color space is scene-linear in nature, a “shaper LUT” or similar non-linearity before the 3D LUT can be used to convert values into a more perceptually uniform representation.

9.2 Extensions

Implementers of CLF readers are advised to protect against unrecognized elements or attributes that are not defined in this specification. Unrecognized elements that are not children of the `Info` element would be expected to raise an error or at least provide a warning message to the user to indicate that there is an operator present that is not recognized by the reader. Applications that need to add custom metadata can place it under the `Info` element rather than at the top level of the `ProcessList`.

One or more `Description` elements in the `ProcessList` can be used for metadata that does not fit into a provided field in the `Info` element and/or is unlikely to be recognized by other applications.

Annex A
(informative)

Interpolation

A.1 General

When an input value falls between sampled positions in a LUT, the output value is calculated as a proportion of the distance along some function that connects the nearest surrounding values in the LUT.

Three types of interpolation are specified for use with the Common LUT Format (CLF): linear interpolation in the `LUT1D` element, and trilinear or tetrahedral interpolation in the `LUT3D` element.

A.2 Linear Interpolation

Table A.1 is an abbreviated table showing the implicit sampled input values in $x[i]$ where i ranges from 0 to $(n-1)$, and the corresponding output values in $y[i]$ which are the LUT contents.

Table A.1 — 1-D LUT Table (abbreviated)

Index i	Input values x	Output values y
0	0	$\mathrm{LUT}[0]$
⋮	⋮	⋮
i	$i\ /\ (n-1)$	$\mathrm{LUT}[i]$
⋮	⋮	⋮
$n-1$	1	$\mathrm{LUT}[n-1]$

The input value, x_{input}, is compared against the input values, x, to find adjacent indices i and $i+1$ such that the x_{input} is between $x[i]$ and $x[i+1]$. The corresponding output value is then computed using the interpolation formula in Formula (A.1). If $x[i] = x_{input}$, the result can be evaluated directly as $y[i]$.

$$y_{interpolated} = y[i] + \left(x_{input} - x[i]\right) \frac{y[i+1] - y[i]}{x[i+1] - x[i]} \tag{A.1}$$

where

$y_{interpolated}$ is the interpolated value

x_{input} is the input value

$x[i]$ is the implicit sampled input value at the nearest table entry where $x[i] < x_{input}$

$y[i]$ is the output value corresponding to the value $x[i]$

$x[i+1]$ is the implicit sampled input value at the next table entry after $x[i]$

$y[i+1]$ is the output value corresponding to the value $x[i+1]$

A.3 Half-Domain Linear Interpolation

This interpolation method is similar to normal linear interpolation, but in this case, the implicit sampled input values are not uniformly spaced between 0 and 1. Instead, the sampled input values are spaced according to the distribution of representable floating-point numbers in the IEEE 754 "binary16" format. Table A.2 illustrates the relationship between the index i and the sampled implicit input values, which are found by taking the bit pattern of the index as a 16-bit unsigned integer and interpreting it as a "binary16" 16-bit float.

NOTE The "half-domain" name for this method of linear interpolation derives from the common practice of referring to "binary16" 16-bit floats as "half" floats, as opposed to 32-bit "full" floats. Additionally, the input to a function is conventionally referred to as its "domain". Combining the two descriptors results in this method's label as "half-domain" linear interpolation.

Table A.2 — Half-Domain LUT Table (abbreviated)

Index i	Input Values x	Output Values y
0	0.0	LUT[0]
1	5.96046e−8	LUT[1]
2	11.9209e−8	LUT[2]
⋮	⋮	⋮
1024	6.1035e−5	LUT[1024]
⋮	⋮	⋮
15360	1.0	LUT[15360]
⋮	⋮	⋮
31743	65504.0	LUT[31743]
31744	Infinity	LUT[31744]
31745	NaN	LUT[31745]
⋮	⋮	⋮
32767	NaN	LUT[32767]
32768	−0.0	LUT[32768]
⋮	⋮	⋮
48128	−1.0	LUT[48128]
⋮	⋮	⋮
64511	−65504.0	LUT[64511]
64512	−Infinity	LUT[64512]
64513	NaN	LUT[64513]
⋮	⋮	⋮
65535	NaN	LUT[65535]

As with normal linear interpolation, the input value, x_{input}, is compared against the input values, x, to find adjacent indices i and $i+1$ such that the x_{input} is between $x[i]$ and $x[i+1]$. The corresponding output value is then computed using the interpolation formula in Formula (A.1).

Because the LUT follows the layout of an IEEE 754 float, there are entries for positive and negative infinity and NaN ("not a number") values. Since using infinity in linear interpolation math does not make sense, the input value is clamped to the representable domain of finite values for a half-float: [−65504, +65504]. The LUT entries at indices of 31744 and 64512 are therefore not used.

The region of the LUT corresponding to NaN value can be used to remap NaN inputs to a designated output value. However, since there are 2048 values for NaN in the table and in the context of imaging all NaN values are typically considered interchangeable, there is no need to interpolate. Implementations can therefore return any one of the LUT entries corresponding to NaNs. LUT authors are encouraged to set all NaN entries to the same output value.

Handling of exceptional floating-point values such as infinity and NaN can vary considerably among various hardware, especially when comparing devices such as an FPGA or GPU to a typical CPU. Furthermore, it can require extra computation to carefully handle these values when they occur in the input. Therefore, LUT authors are reminded that the exact behavior of infinity and NaN handling can vary somewhat among implementations.

A half-float LUT has separate entries for both positive and negative zero. LUT authors are asked to ensure that the LUT entries at indices of 0 and 32768 are identical to avoid any potential differences in behavior based on how different hardware handles signed zeros.

A.4 Trilinear Interpolation

Trilinear interpolation implements linear interpolation in three dimensions by successively interpolating each direction.

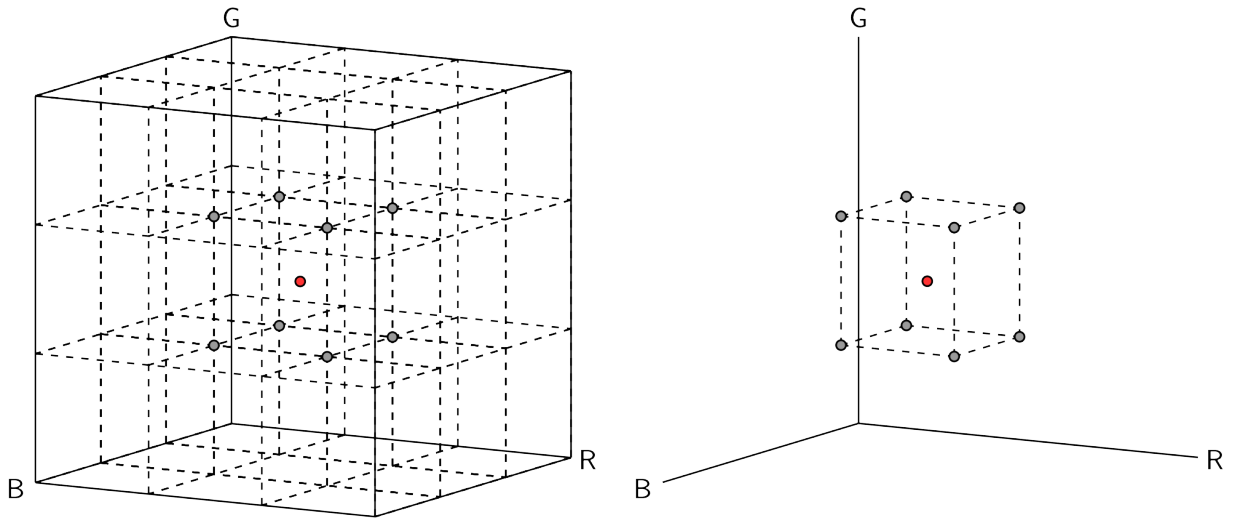

Figure A.1 — Illustration of a sampled point located within a basic 3D LUT mesh grid (left) and the same point with only the vertices surrounding the sampled point (right).

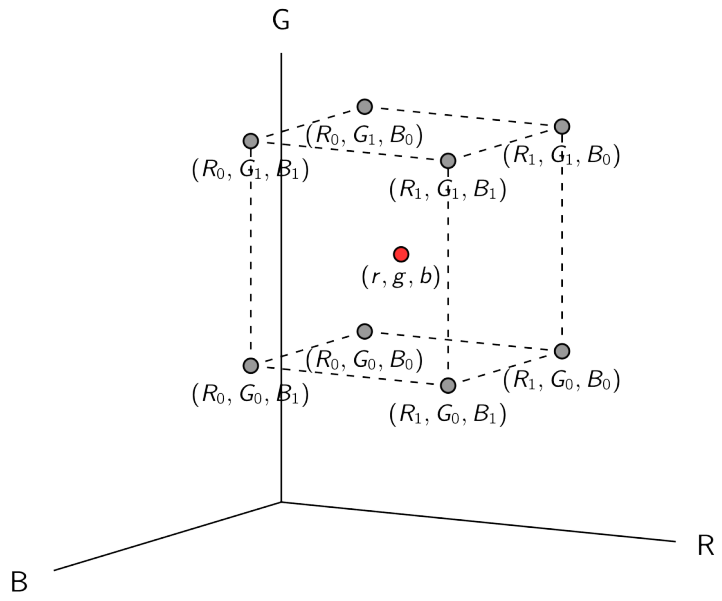

Figure A.2 — Labeling the mesh points surrounding the sampled point (*r, g, b*).

NOTE 1 The convention used for notation is uppercase variables for mesh points and lowercase variables for the points on the grid.

Consider a sampled point as depicted in Figure A.1. Figure A.2 assigns labels to the input point and the vertices of the cubelet defined by the mesh points that enclose the input value.

The following symbols are used in Formulae (A.2) through (A.7).

$V( )$	is the value at the point with the coordinates enclosed by the parentheses
r, g, b	are the coordinates of the input point
$R_i,\ G_j,\ B_k \in \{0,1\}$	are the coordinates of the grid points that directly enclose the sampled point, where *x* is the set {0,1}, and as depicted in Figure A.2
$\mathbf{A}^{\mathrm{T}}$	is common matrix notation indicating the transpose of the matrix $\mathbf{A}$

The distance between each node per color coordinate shows the proportion, Δ, of each mesh point's color coordinate values that contribute to the sampled point, and is calculated for each color coordinate according to Formula (A.2).

$$\Delta_r = \frac{r - R_0}{R_1 - R_0} \quad \Delta_g = \frac{g - G_0}{G_1 - G_0} \quad \Delta_b = \frac{b - B_0}{B_1 - B_0} \tag{A.2}$$

The general expression for trilinear interpolation can be written as in Formula (A.3).

$$V(r,g,b) = c_0 + c_1\Delta_b + c_2\Delta_r + c_3\Delta_g + c_4\Delta_b\Delta_r + c_5\Delta_r\Delta_g + c_6\Delta_g\Delta_b + c_7\Delta_r\Delta_g\Delta_b \quad \text{(A.3)}$$

where

$$\begin{aligned}
c_0 &= V(R_0, G_0, B_0) \\
c_1 &= V(R_0, G_0, B_1) - V(R_0, G_0, B_0) \\
c_2 &= V(R_1, G_0, B_0) - V(R_0, G_0, B_0) \\
c_3 &= V(R_0, G_1, B_0) - V(R_0, G_0, B_0) \\
c_4 &= V(R_1, G_1, B_1) - V(R_1, G_0, B_0) - V(R_0, G_0, B_1) + V(R_0, G_0, B_0) \\
c_5 &= V(R_1, G_1, B_0) - V(R_0, G_1, B_0) - V(R_1, G_0, B_0) + V(R_0, G_0, B_0) \\
c_6 &= V(R_0, G_1, B_1) - V(R_1, G_1, B_0) - V(R_0, G_0, B_1) + V(R_0, G_0, B_0) \\
c_7 &= V(R_1, G_1, B_1) - V(R_1, G_1, B_0) - V(R_0, G_1, B_1) - V(R_1, G_0, B_1) \\
&\quad + V(R_0, G_0, B_1) + V(R_0, G_1, B_0) + V(R_1, G_0, B_0) - V(R_0, G_0, B_0)
\end{aligned}$$

Formula (A.3) can be rewritten as a matrix product as in Formula (A.4).

$$V(r,g,b) = \mathbf{C}^{\mathrm{T}}\mathbf{\Delta} \quad \text{(A.4)}$$

where

$$\mathbf{C} = [c_0\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7]^{\mathrm{T}}$$

$$\mathbf{\Delta} = \begin{bmatrix} 1 & \Delta_b & \Delta_r & \Delta_g & \Delta_b\Delta_r & \Delta_r\Delta_g & \Delta_g\Delta_b & \Delta_r\Delta_g\Delta_b \end{bmatrix}^{\mathrm{T}}$$

The matrix $\mathbf{C}$ can itself be written as the product of two matrices according to Formula (A.5).

$$\mathbf{C} = \mathbf{AV} \quad \text{(A.5)}$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} V(R_0, G_0, B_0) \\ V(R_0, G_1, B_0) \\ V(R_1, G_0, B_0) \\ V(R_1, G_1, B_0) \\ V(R_0, G_0, B_1) \\ V(R_0, G_1, B_1) \\ V(R_1, G_0, B_1) \\ V(R_1, G_1, B_1) \end{bmatrix}$$

If the definition of $\mathbf{C}$ from (A.5) is substituted into the matrix product first formulated in (A.4), trilinear interpolation can be expressed as in Formula (A.6).

$$V(r,g,b) = \mathbf{V}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{\Delta} \tag{A.6}$$

NOTE 2 The term $\mathbf{V}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$ does not depend on the variable (r,g,b) and can be computed in advance for optimization. Each sub-cube can have the values of the vector $\mathbf{C}$ already stored in memory. Thus, the trilinear interpolation algorithm can be summarized as:

1. Find the cubelet containing the point (r,g,b)
2. Select the vector $\mathbf{C}$ corresponding to that cubelet
3. Compute Δ_r, Δ_g, Δ_b
4. Return $V(r,g,b) = \mathbf{C}^{\mathrm{T}}\mathbf{\Delta}$

A.5 Tetrahedral Interpolation

Tetrahedral interpolation subdivides the cubelet defined by the vertices surrounding a sampled point into six tetrahedra by segmenting along the main (and usually neutral) diagonal (Figure A.3).

The tetrahedra are formed by starting at the (0,0,0) corner, moving in the direction of the largest component, then the second largest component, then the smallest component to arrive at the (1,1,1) corner.

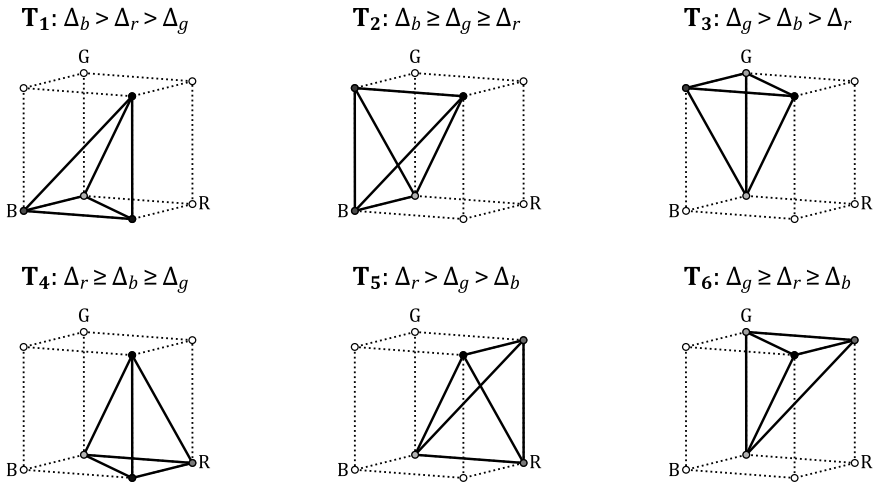

Figure A.3 — Illustration of the six possible subdivided tetrahedra.

To find the tetrahedron containing the point (r, g, b), first calculate Δ for each color coordinate according to Formula (A.2). Tetrahedral interpolation is done according to Formula (A.7).

$$V(r,g,b) = \begin{cases} \mathbf{\Delta}_t^{\mathrm{T}}\mathbf{T}_1\mathbf{V} & \text{if } \Delta_b > \Delta_r > \Delta_g \\ \mathbf{\Delta}_t^{\mathrm{T}}\mathbf{T}_2\mathbf{V} & \text{if } \Delta_b \geq \Delta_g \geq \Delta_r \\ \mathbf{\Delta}_t^{\mathrm{T}}\mathbf{T}_3\mathbf{V} & \text{if } \Delta_g > \Delta_b > \Delta_r \\ \mathbf{\Delta}_t^{\mathrm{T}}\mathbf{T}_4\mathbf{V} & \text{if } \Delta_r \geq \Delta_b \geq \Delta_g \\ \mathbf{\Delta}_t^{\mathrm{T}}\mathbf{T}_5\mathbf{V} & \text{if } \Delta_r > \Delta_g > \Delta_b \\ \mathbf{\Delta}_t^{\mathrm{T}}\mathbf{T}_6\mathbf{V} & \text{if } \Delta_g \geq \Delta_r \geq \Delta_b \end{cases} \tag{A.7}$$

where

$$\mathbf{V} = \begin{bmatrix} V(R_0, G_0, B_0) \\ V(R_0, G_1, B_0) \\ V(R_1, G_0, B_0) \\ V(R_1, G_1, B_0) \\ V(R_0, G_0, B_1) \\ V(R_0, G_1, B_1) \\ V(R_1, G_0, B_1) \\ V(R_1, G_1, B_1) \end{bmatrix}$$

$$\mathbf{\Delta_t} = \begin{bmatrix} 1 & \Delta_b & \Delta_r & \Delta_g \end{bmatrix}^{\mathrm{T}}$$

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \quad \mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{T}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

$$\mathbf{T}_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{T}_6 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

NOTE The vectors $\mathbf{T}_i\mathbf{V}$ for $i = 1,2,3,4,5,6$ do not depend on the variable (r, g, b) and can be computed in advance for optimization.

Annex B
(informative)

Sample CLF instances

This annex lists sample CLF instances for illustration. The sample CLF instances are also provided as elements “b”, and “c” (see Annex D).

EXAMPLE 1
CIE XYZ to CIELAB L*a*b*

```
<ProcessList xmlns="http://www.smpte-ra.org/ns/2136-1/2024" id="test"
compCLFversion="3.0">
	<Id>urn:uuid:3bae2da8-59c7-44c2-b6c9-76dc0f8cae8c</Id>
	<Description>CIE-XYZ D65 to CIELAB L*, a*, b* (scaled by 1/100, neutrals at 0.0
chroma)</Description>
	<InputDescriptor>CIE-XYZ, D65 white (scaled [0,1])</InputDescriptor>
	<OutputDescriptor>CIELAB L*, a*, b* (scaled by 1/100, neutrals at 0.0
chroma)</OutputDescriptor>
	<Matrix inBitDepth="16f" outBitDepth="16f">
		<Array dim="3 3">
			1.052126639 0.000000000 0.000000000
			0.000000000 1.000000000 0.000000000
			0.000000000 0.000000000 0.918224951
		</Array>
	</Matrix>
	<Exponent inBitDepth="16f" outBitDepth="16f" style="monCurveRev">
		<ExponentParams exponent="3.0" offset="0.16"/>
	</Exponent>
	<Matrix inBitDepth="16f" outBitDepth="16f">
		<Array dim="3 3">
			0.00000000  1.00000000  0.00000000
			4.31034483 -4.31034483  0.00000000
			0.00000000  1.72413793 -1.72413793
		</Array>
	</Matrix>
</ProcessList>
```

EXAMPLE 2
IDENTITY TRANSFORM ILLUSTRATING ARRAY BIT DEPTH SCALING

```
<ProcessList xmlns="http://www.smpte-ra.org/ns/2136-1/2024" id="test"
compCLFversion="3.0">
	<Id>urn:uuid:9d768121-0cf9-40a3-a8e3-7b49f79858a7</Id>
	<Description>Identity transform illustrating Array bit depth
scaling</Description>
	<Description>Can be loaded by either SMPTE or CLF v3 parsers</Description>
	<Matrix inBitDepth="8i" outBitDepth="16i">
		<Array dim="3 3">
			257.0    0.0    0.0
			  0.0  257.0    0.0
			  0.0    0.0  257.0
		</Array>
	</Matrix>
	<LUT1D inBitDepth="16i" outBitDepth="16i">
		<Array dim="7 1">
			0.0  10922.5  21845.0  32767.5  43690.0  54612.5  65535.0
		</Array>
	</LUT1D>
	<Matrix inBitDepth="16i" outBitDepth="16i">
		<Array dim="3 3">
			1.0    0.0    0.0
			0.0    1.0    0.0
			0.0    0.0    1.0
		</Array>
	</Matrix>
</ProcessList>
```

Annex C
(informative)

IANA Media type registration

This annex serves to register and document the Media Type application/clf-xml, which is associated with files conforming to SMPTE ST 2136-1. The registration is intended to address the requirements of IETF RFC 6838.

C.1 Author (name and email)

Fred Walls, fwalls@m.smpte.org

C.2 Type name

application

C.3 Subtype name

clf+xml

C.4 Required Parameters

N/A

C.5 Optional Parameters

N/A

C.6 Encoding considerations

Same as encoding considerations of application/xml as specified in IETF RFC 7303. UTF-8 only.

C.7 Security considerations

Same as general security considerations for application/xml, see Section 10 of IETF RFC 7303.

C.8 Interoperability considerations

Same as Interoperability considerations for application/xml, see Section 9.1 of IETF RFC 7303.

C.9 Published Specification

SMPTE ST 2136-1

C.10 Applications that use this media type

Hardware and software implementations using the Common LUT Format as defined in SMPTE ST 2136-1 for image and video processing.

C.11 Fragment identifier considerations

N/A

C.12 Restrictions on usage

N/A

C.13 Provisional Registration?

YES

C.14 Additional information

N/A

C.15 Deprecated alias names for this type

N/A

C.16 Magic number(s)

None

C.17 File extension(s)

.clf

C.18 Macintosh file type code(s)

"TEXT"

C.19 Object Identifier(s) or OID(s)

None

C.20 Intended usage

COMMON

C.21 Other information & Comments

IMPORTANT: This is a draft presented as a request for feedback from IANA experts. We respectfully request a provisional registration while the document is in process.

C.22 Contact Person

Director of Standards, Society of Motion Picture and Television Engineers (SMPTE)
standards-support@smpte.org

Annex D
(informative)

Additional elements

This annex lists non-prose elements of this document.

Non-prose element	File name	Description
a	st2136-1a-2026.xsd	XML schema document (normative)
b	st2136-1b-2026.clf	EXAMPLE 1 in Annex B (informative)
c	st2136-1c-2026.clf	EXAMPLE 2 in Annex B (informative)

Bibliography (informative)

Academy S-2014-002, *Academy Color Encoding System – Versioning System*. http://j.mp/S-2014-002

Academy TB-2014-002, *Academy Color Encoding System Version 1.0 User Experience Guidelines*. http://j.mp/TB-2014-002

ASC Color Decision List (ASC CDL) Transfer Functions and Interchange Syntax. ASC-CDL Release1.2. Joshua Pines and David Reisner. 2009-05-04.

IETF RFC 6838: *Media Type Specifications and Registration Procedures*. https://datatracker.ietf.org/doc/html/rfc6838

IETF RFC 7303: *XML Media Types*. https://datatracker.ietf.org/doc/html/rfc7303