SMPTE STANDARD

XML Representation of SMPTE Registered Data (Reg-XML) — Mapping Rules

Table of Contents

Approved
November 6, 2013

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 Part XIII of its Operations Manual.

SMPTE ST 2001-1 was prepared by Technology Committee 31FS.

Intellectual Property

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.

Introduction

This section is entirely informative and does not form an integral part of this Engineering Document.

The purpose of this standard is to define a canonical XML representation of SMPTE-registered data models. The XML representation enables round-trip conversion of KLV-encoded metadata and streams to XML and back to KLV.

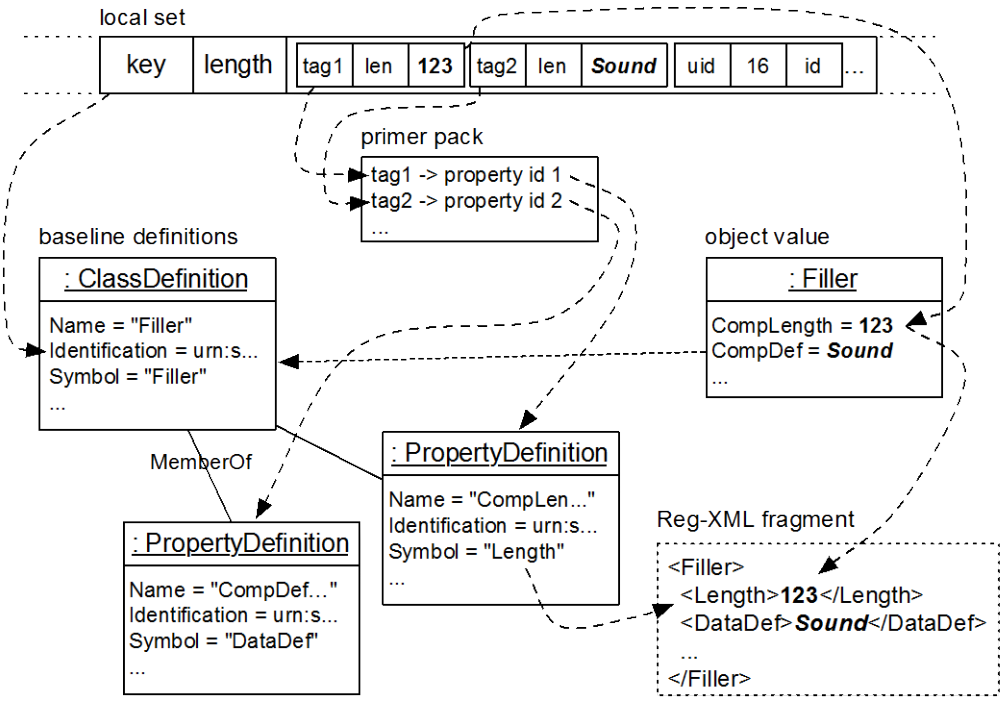

Figure 1 – KLV local set to Reg-XML fragment

Figure 1 illustrates the relationship between data encoded as a KLV local set and its corresponding Reg-XML fragment. A Reg-XML data model defines class, property and type definitions in terms of names, identifications and the symbols used for XML elements. Using the definitions of a Reg-XML data model and the data mapping rules of Reg-XML, it is possible to convert the KLV-encoded data set-by-set, property-by-property, to XML elements and values. The reverse process to that shown in the figure is also possible.

Reg-XML instance documents and fragments are not defined by XML Schema documents. The link between instance data and its definition in a data model is established by the XML namespace used for the instance data. This namespace is used as the data model's scheme URI. An XML Schema can be derived from a data model to assist with Reg-XML document validation.

The XML representation also supports extension data models. This allows low-overhead static schemas to co-exist with ad hoc extension schemas so that decoders can dynamically validate instance documents and fragments independent of their source.

Reg-XML does not redefine what already exists in terms of data model. Nor is it imposing additional constraints on encoders. For example, no requirement exists to sort properties into an externally predetermined order. Also, Reg-XML does not add requirements on underlying standards. For example, Reg-XML has no new restrictions on MXF (SMPTE ST 377-1) etc. to enable the XML representation.

The SMPTE Registers (Elements, Types, Groups, Labels) already contain a full statement of the data model for an application, for example a MXF *Preface* (urn:smpte:ul:060E2B34.02060101.0D010101.01012F00). In effect, the registers' contents are an Object Model. Similarly, any SMPTE engineering document containing definitions for elements, groups or types can be considered as specifying its own data model.

This is a multi-part specification, in which this part (Part 1) defines what a data model is and the rules for creating an XML representation of data that conforms to that model as a *Reg-XML document*. Part 1 also includes an informative set of rules for how to create an XSD from a data model that can be uses to validate a Reg-XML document that conforms to the model.

Other parts specify baseline Reg-XML data models. For example, Part 2 provides a baseline data model for AAF and MXF and the specifics of applying the mapping rules of part 1 to those models. Part 2 states how to create the XML representation of the data contained in a MXF file or an AAF file. Part 2 also defines how to start with an XML representation and create a MXF file or an AAF file.

1 Scope

This standard defines an XML representation for data that is registered in the SMPTE metadata registers (SMPTE ST 395, SMPTE ST 335, SMPTE ST 2003 and SMPTE ST 400). The standard is applicable for data that can be described by a Reg-XML data model.

The specification defines the items with which a Reg-XML data model is specified, namely classes, properties and types.

The specification defines data mapping rules for representing the data in XML.

The specification defines informative model mapping rules that describe how to map a Reg-XML data model onto W3C XML Schemas (XSDs) for validation.

This specification provides a normative XSD for the representation of Reg-XML data models as Meta Dictionary XML documents.

2 Conformance Notation

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 section 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; followed by formal languages; then figures; and then any other language forms.

3 Normative References

Note: All references in this document to other SMPTE documents use the current numbering style (e.g. SMPTE ST 298:2009) although, during a transitional phase, the document as published (printed or PDF) may bear an older designation (such as SMPTE 298-2009). Documents with the same root number (e.g. 298) and publication year (e.g. 2009) are functionally identical.

The following standards contain provisions that, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this recommended practice are encouraged to investigate the possibility of applying the most recent edition of the standards indicated below.

SMPTE ST 298:2009, Universal Labels for Unique Identification of Digital Data

SMPTE ST 330:2011, Unique Material Identifier (UMID)

SMPTE ST 335:2012, Metadata Element Dictionary Structure

SMPTE ST 395:2013, Metadata Groups Registry Structure

SMPTE ST 400:2012, SMPTE Labels Structure

SMPTE ST 2003:2012, Types Dictionary Structure

SMPTE ST 2029:2009, Uniform Resource Names for SMPTE Resources

Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, 4 February 2004, http://www.w3.org/TR/2004/REC-xml-20040204/

Namespaces in XML 1.0 (Third Edition), W3C Recommendation, 8 December 2009, http://www.w3.org/TR/REC-xml-names/

XML Schema Part 1: Structures Second Edition, W3C Recommendation, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

XML Schema Part 2: Datatypes Second Edition, W3C Recommendation, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

XML Linking Language (XLink) Version 1.1, W3C Recommendation, 6 May 2010, http://www.w3.org/TR/xlink11/

XML Path Language (XPath) Version 1.0, W3C Recommendation, 16 November 1999, http://www.w3.org/TR/xpath/

IETF RFC 1738, Uniform Resource Locators (URL), December 1994

IETF RFC 2141, URN Syntax, May 1997

IETF RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax, August 1998

IETF RFC 3061, A URN Namespace of Object Identifiers, February 2001

IETF RFC 4122, A Universally Unique Identifier (UUID) URN Namespace, July 2005

4 Definition of Acronyms, Terms and Notation

4.1 Acronyms and Terms

AUID A 16-byte UID that shall contain a UL or a UUID. If the value is a UUID, it shall be stored such that the top and bottom 8 bytes of the UUID are swapped. For UUIDs, this makes the most significant bit of the first byte a '1' and thus creates a UID value that is always distinct from a UL.

Namespace XML Namespace

URI Uniform Resource Identifier

URL Uniform Resource Locator

URN Uniform Resource Name

UUID Universally Unique IDentifier

XML eXtensible Markup Language

XSDL XML Schema Definition Language — the W3C XML schema language used to define a class of XML documents

XSD XML Schema Definition — an XML document containing a schema expressed in XSDL

XLink XML Linking Language — allows elements to be inserted into XML documents in order to create and describe links between resources

XPath XML Path Language - a language for addressing parts of an XML document

4.2 Notation

A property *X* owned by class *Y* is shown using the notation *Y*::*X*, e.g. Preface::FileLastModified refers to the FileLastModified property in the Preface class.

The mapping rules use the notation {*xxx*} to indicate that the value *xxx* must be filled in when applying the mapping rule, e.g. the name of the type must be filled in the XSD definition <*simpleType name="{name}"*>.

The mapping rules use the notation *[…]?* for things that are only required when certain conditions are met, e.g. *[reg:uid="{id}"]?* means that the *reg:uid* attribute is only required when the conditions given in the notes are met.

The mapping rules use the Namespace prefix *reg* to refer to the baseline namespace, *cx* for an example company X namespace representing user/vendor extensions and *this* for ad hoc extensions not explicitly assigned to a namespace. Applications are not required to use these Namespace prefixes.

5 Prologue

Reg-XML has two distinct components:

- the specification of data models that correspond to definitions in the SMPTE metadata registers (SMPTE ST 395, SMPTE ST 335, SMPTE ST 2003 and SMPTE ST 400) and the normative source of those definitions;

- the representation of data defined by the data models using XML.

The XML representation consists of an XML document, called a *Reg-XML document*, and zero or more files or resources containing stream data. The stream data files are referenced from within the Reg-XML document. The Reg-XML document and the referenced stream data are together called a *Reg-XML file group*. A fragment of a Reg-XML document, called a *Reg-XML fragment*, may be included in another context.

A Reg-XML file group is intended for use directly as a data interchange format between systems, or it may be converted to or from some other data representation for the purpose of data interchange between systems. For example, a Reg-XML file group may be converted to or from an AAF or MXF file.

SMPTE ST 2001 is a multipart standard, with this part defining the rules for specifying a Reg-XML data model and its use. SMPTE ST 2001-1 is independent of any particular Reg-XML data model.

The structure of this document is as follows:

- Section 6 provides an informative description of a Reg-XML data model and defines the use of a Reg-XML data model. The Reg-XML data model is used as input for both the data and model mapping rules and is normatively specified by the meta dictionary XML schema in Annex A.

- Section 7 defines the association between a Reg-XML document and Reg-XML data models that contain baseline and extension definitions for the data the document contains. This association is achieved through the use of XML Namespaces.

- Section 8 defines the *data mapping rules* for representing data as a Reg-XML file group. The data mapping rules are described independently from any XML schema language.

- Section 9 defines the informative *model mapping rules* for generating W3C XML Schemas (XSDs) from a Reg-XML data model for validation.

6 Reg-XML Data Models (Informative)

6.1 Definition of a Reg-XML Data Model

A Reg-XML data model is a collection of class, property and type definitions.

A Reg-XML data model is described using one or more *Schemes*, with a *Meta Dictionary* used to define each *Scheme*. The XML schema in Annex A provides the normative specification of a Reg-XML data model, with additional annotations provided in this section.

Class, property and type definitions share the following attributes:

- *Identification*: an AUID that uniquely identifies the definitions.

- *Symbol*: a string that uniquely identifies the definition in the context of a Scheme.

- *Description*: describes the definition.

- *Name*: a descriptive name for the definition.

6.1.1 Meta Dictionary

A Meta Dictionary is a collection of class, property and type definitions that share a Scheme. It is described using the following attributes:

- *SchemeID*: an AUID that uniquely identifies the Meta Dictionary.

- *SchemeURI*: a URI that uniquely identifies the Meta Dictionary.

- *Preferred Prefix*: the preferred prefix to use in an XML document.

- *Description*: describes the contents of the Meta Dictionary.

6.1.2 Class Definition

A class definition describes a group of properties. A class definition has the following attributes.

- *Parent Class*: specifies the parent of the class being defined.
- *Is Concrete*: specifies if the class is concrete. If the class is not concrete then it is abstract. An instance of an abstract class also belongs to a concrete sub-class of the abstract class.

6.1.3 Property Definition

A property definition describes a property allowed in a class. A property definition has the following attributes:

- *Type*: specifies the property type
- *Is Optional*: specifies whether objects can omit a value for the property.
- *Is Unique Identifier*: specifies that this property provides a unique identification for the object.
- *Local Identification*: specifies a local integer identification that is used to identify the property in a file. For dynamic properties where the local identification is generated per file, set this value to 0.
- *Member Of*: specifies the class to which the property belongs.

6.1.4 Property Alias Definition

A property alias definition is a kind of property definition that specifies a globally unique alias for a property that is already defined for another class. This allows the property to be reused unambiguously in more than one class.

A property alias definition has all the attributes of a property definition and the following additional attribute:

- *Original Property*: specifies the original definition of the reused property.

The property alias definition and original property is of the same type.

6.1.5 Character Type Definition

A character type definition describes a property type that has a single Unicode character. A character type definition does not have any additional attributes.

6.1.6 Enumeration Type Definition

An enumeration type definition describes a property type that can have one of a set of integer values. An enumeration type definition has the following attributes:

- *Element Type*: specifies the type definition that defines the underlying integer type.
- *Elements*: specifies the name, value and description associated with each enumerated value. Each value is a positive integer value. Each name and value is unique within the enumeration type definition. The description is optional and may be omitted.

6.1.7 Extendible Enumeration Type Definition

An extendible enumeration type definition describes a property type that can have one of an extendible set of AUID values. An extendible enumeration type definition has the following attributes:

- *Elements*: specifies the name, value and description associated with each enumerated value. Each name and value is unique within the extendible enumeration type definition. The description is optional and may be omitted.

6.1.8 Fixed Array Type Definition

A fixed array type definition describes a property type that has a fixed number of values of the element type. The order of the values is meaningful. A fixed array type definition has the following attributes:

- *Element Type*: specifies the type of the elements in the array.
- *Element Count*: specifies the number of elements in the array.

6.1.9 Indirect Type Definition

An indirect type definition describes a property type that has value whose type is specified in each instance. An indirect type definition does not have any additional attributes.

6.1.10 Integer Type Definition

An integer type definition describes a property type that is an integer with the specified number of bytes. An integer type definition has the following attributes:

- *Size*: specifies the number of bytes to store the value. Legal values are 1, 2, 4, and 8.
- *Is Signed*: specifies whether the integer is signed or unsigned.

6.1.11 Opaque Type Definition

An opaque type definition describes a property type that has a value whose type is specified in each instance. An opaque type definition does not have any additional attributes.

6.1.12 Record Type Definition

A record type definition describes a property type that consists of an ordered set of fields, where each field has a name and a type. A record type definition has the following attributes:

- *Members*: specifies the name and type for each field of the record, with an optional description.

6.1.13 Rename Type Definition

A rename type definition describes a property type that has the same structure and representation as the underlying type but has a different meaning. A rename type definition has the following attributes:

- *Renamed Type*: specifies the underlying type.

6.1.14 Set Type Definition

A set type definition describes a property type that has either a collection of object references to uniquely identified objects or a collection of unique values of a single type. The order of the objects has no meaning. A set type definition has the following attributes:

- *Element Type*: specifies the type of the elements in the set.

6.1.15 Stream Type Definition

A stream type definition describes a property type that is stored in a stream and has a value that consists of a varying number of bytes. The order of the bytes is meaningful. A stream type definition does not have any additional attributes.

6.1.16 String Type Definition

A string type definition describes a property type that consists of a zero-terminated array of the underlying character or integer type. A string type definition has the following attributes:

- *Element Type*: specifies the string element, which can be a character type or an integer type.

6.1.17 Strong Object Reference Type Definition

A strong object reference type definition describes a property type that defines an object relationship where the target of the strong reference is owned by the object with the property with the strong object reference type. An object can be the target of only one strong object reference. A strong object reference type definition has the following attributes:

- *Referenced Type*: specifies the class of the references object belongs to – the references object may also belong to a sub-class of the referenced class.

6.1.18 Variable Array Type Definition

A fixed array type definition describes a property type that has a variable number of values of the element type. The order of objects is significant and values of the element type may not be unique. A variable array type definition has the following attributes:

- *Element Type*: specifies the type of the elements in the array.

6.1.19 Weak Object Reference Type Definition

A weak object reference type definition describes a property type that defines an object relationship where the target of the weak reference is referenced by a unique identifier. Only objects that define a unique identification can be the targets of weak object references. An object can be the target of one or more than one weak reference. A weak object reference type definition has the following attributes:

- *Referenced Type*: specifies the class of the references object – the referenced object may also belong to a sub-class of the referenced class.

- *Target Set*: specifies the properties from the root of the file to the property that has the strong reference set containing the uniquely identified objects that may be the target of the weak reference. The first property in the array identifies the object in the file's root storage. The last property in the array contains the set of uniquely identified objects. The properties between the first and the last property must have a strong object reference type and define the containing hierarchy from the object in the root storage to the object containing the strong object reference set.

7 Reg-XML Documents and Meta Dictionaries

7.1 Conformant Reg-XML Documents

A Reg-XML document shall conform to a Reg-XML data model that consists of baseline meta definitions as well as extension meta definitions. A Reg-XML document shall be a complete representation of a set of data, starting at the declared root object of the baseline data model. The root object shall be encoded as a child of the declared root element of the baseline data model, as defined by the data mapping rules. The definitions shall be defined in meta dictionaries.

7.2 Conformant Reg-XML Fragments

A Reg-XML fragment shall be an element extracted from a Reg-XML document that represents the value of an object. A Reg-XML fragment shall be an XML element that conforms to the object-defining class definition of the Reg-XML data model of the overall document. All member properties of the object shall be encoded as descendent XML elements of the extracted element according to the data mapping rules.

Unlike for Reg-XML documents, the object does not have to be defined by the declared root object of a baseline data model.

Matching the SchemeURI of the data model with the namespace declaration of an XML element, such as a Reg-XML fragment, shall identify the data model that contains the definition of the object.

Note: The hierarchical position of an object within its data set relative to the root object of the defining data model can be provided through the reg:path attribute, as defined in Rule 3 of the data mapping rules. The path is an XPath Absolute Location, measured from the root element of the corresponding baseline data model.

7.3 Baseline Data Model and Meta Dictionary

The namespace name part of the expanded name of the root element of a Reg-XML document shall identify the baseline data model for the data contained in a Reg-XML document. A *baseline meta dictionary* shall define the baseline data model. The namespace name part of the expanded name of the root element shall be the same as the SchemeURI property of the baseline meta dictionary.

The baseline meta dictionary shall not be contained within the Reg-XML document. The baseline meta dictionary may be stored in an external XML document that conforms to the schema provided in Annex B. With the exception of extension meta definitions (see Section 7.4), a baseline meta dictionary shall contain the definition of any class, property or type with instance data in the Reg-XML document.

7.4 Extension Data Models and Meta Dictionaries

Extension meta definitions shall be used to supplement the data model provided by the baseline meta dictionary of a Reg-XML document. Extension meta definitions may be grouped into extension meta dictionaries, where each extension meta dictionary contains extensions that are specific to a particular application, user or vendor.

A Reg-XML document may contain zero or more extension meta dictionaries that list extension meta definitions relative to the baseline defined for the document. Any item of instance data shall either be defined by a meta definition in the Reg-XML's baseline data model or by an extension meta definition contained within the Reg-XML document.

Extension meta definitions of a Reg-XML document may be defined with reference to meta definitions defined in the baseline data model of the that document. Extension meta definitions shall not refer to other extension data models declared only within the same Reg-XML document. Extension data models shall not repeat

definitions from the baseline data model. In case of any ambiguity, the definition in the baseline data model shall take precedence over any definition defined in an extension.

7.5 Meta Dictionary Identifiers

A SMPTE UL or UUID, known as the SchemeID, shall uniquely identify a meta dictionary. Where a fixed meta dictionary is published, a SMPTE UL should be registered for identification of instances of this meta dictionary. Otherwise, meta dictionaries shall be identified by a UUID that is unique to each specific instance.

Acceptable values for registered SchemeID labels shall be as defined in the Table 1.

Table 1 – Value of the Meta Dictionary SchemeID Universal Label

Byte No.	Description	Value (hex)	Meaning
1	Object identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	04h	Labels
6	Registry Designator	01h	Labels
7	Structure Designator	01h	Labels
8	Version Number	xxh	Registry Version in which the Label first appeared
9	Item Designator	01h	Identification and Location
10	Globally Unique Identifiers	01h	
11	Definition Identifiers	03h	
12	Reg-XML Definition Identifiers	01h	
13	Reg-XML Meta Dictionary Identifiers	01h	
14	Reg-XML Meta Dictionary Type	01h 02h	Baseline Meta Dictionary Extension Meta Dictionary
15	Meta dictionary kind (1)	yyh	Issued on dictionary registration
16	Meta dictionary kind (2)	zzh	Issued on dictionary registration

In addition to the SchemeID, a meta dictionary shall be uniquely identified by a SchemeURI, which could be an XML namespace name (URI) or set to the same value as the SchemeID. Each meta definition shall be uniquely identified within a scheme by its symbol. The combination of the SchemeID and symbol shall be used as a global identifier for the meta definition.

Note: The combination of the SchemeURI and meta definition's symbol property provide an alternative, globally unique identifier for the meta definition.

7.6 XML Namespaces in Reg-XML Documents

Reg-XML documents shall fully comply with W3C Namespaces in XML.

For any data instance, the expanded name in the Reg-XML document shall have a namespace name part equal to its defining meta dictionary SchemeURI property and its local name part equal to the symbol property of its defining meta definition.

Note: Therefore, the SchemeURI defined for the baseline meta dictionary is used as the XML Namespace for all the XML elements defined by that meta dictionary in a Reg-XML document.

The SchemeURI of the baseline should be used as the default namespace for the Reg-XML document's root element. When this is the case, elements representing values of data instances defined by the baseline should have *unprefixed names* (see W3C Namespaces in XML).

The baseline meta dictionary SchemeURI should be used to define a namespace with prefix *reg* to allow for the use of fully qualified names.

For data instances defined by extension meta definitions, the PreferredPrefix property of the extension meta dictionary should be used as the namespace prefix in qualified names.

Note: The Reg-XML format uses the SchemeURIs as XML Namespaces and the symbols as XML element names or XSD type names. The Scheme of the set of baseline meta definitions that define the baseline meta dictionary will be referred to as the *baseline scheme* in this specification. The mapping rules in Section 8 and Section 9 make use of the XML terms *Namespace* for the *scheme* and *name* for the defined *symbol*.

7.7 Local Extension Meta Dictionary

If a Reg-XML document contains data instances not defined in the baseline meta dictionary or one of its defined extensions, an additional *local extension meta dictionary* shall be included in the file. This additional meta dictionary shall contain the omitted meta definitions and shall use the preferred prefix *this*. A UUID shall be used to identify the local extension meta dictionary.

Annex A shall be used as the format for the definition of a baseline meta dictionary and an extension meta dictionary.

8 Data Mapping Rules

This section defines how data is mapped into a Reg-XML file group.

8.1 Data Types

8.1.1 AUID

The AUID data type shall be either a SMPTE UL or a UUID.

Values that have type SMPTE UL shall be represented using the SMPTE ST 2029 UL URN notation and component values shall be represented in big endian byte order. For example:

urn:smpte:ul:060e2b34.04010103.04010202.01011100

AUID values that are UUIDs shall be represented using the IETF RFC 4122 UUID URN notation. For example:

urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6

8.1.2 Byte Order

The byte order type is used for the global attribute *reg:byteOrder* and the *Preface::ByteOrder* property. The allowed values are: 'BigEndian' and 'LittleEndian'.

8.1.3 Characters and Strings

A Reg-XML character reference mechanism is defined to allow characters to be used that, according to the underlying XML 1.0 specification, are not in the range that must be accepted by XML processors. The

Reg-XML character reference mechanism ensures that all possible characters can be represented in the Reg-XML document. For XML version 1.0, the range of characters that must be accepted by XML processors are:

#x09 | #x0A | #x0D | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]

A Reg-XML character reference is the same as an XML character reference except that it uses the '$' character (#x24) for escaping rather than the '&' character (#x26). A Reg-XML character reference is defined as follows:

'$#' [0-9]+ ';' | '$#x' [0-9a-fA-F]+ ';'

A character that is not in the range that must be accepted by XML processors, as defined by the underlying XML specification, shall be escaped using a Reg-XML character reference. Characters in the range that must be accepted by XML processors shall not be escaped using a Reg-XML character reference.

If Reg-XML character escaping is used then the *reg:escaped* attribute shall be present and the value shall be set to 'true'. If the *reg:escaped* attribute is present and set to 'true' then the '$' (#x24) character shall be escaped using a Reg-XML character reference, e.g. '$#x24;'.

The null character (#x00) is used as a zero-terminator for String types and shall not be used in property values based on the String type. The null character may be used in other property values based on the Character type.

The carriage return character (#x0D) is translated into a line feed character (#x0A) by XML processors before parsing the XML document. The carriage return character shall not be written directly in a Reg-XML document but shall be written using an XML character reference, e.g. '
'.

Note: Reg-XML is based on XML 1.0 in order to support the widest possible range of XML processors.

8.2 Global Attributes (Informative)

The rules applying to global attributes can be summarized as follows:

The *reg:uid* (see rule 3) attribute is used for weak references.

The *reg:byteOrder* attribute (see rule 5.7 and 5.11) is used to indicate the byte order for property values with a Stream or Opaque type. The byte order type in Section 8.1.2 defines the set of allowed values.

The *reg:stream* attribute (see rule 5.11) references an external file containing Stream data via an external unparsed entity declared in the internal DTD.

The *reg:actualType* attribute (see rule 5.5 and 5.7) is used in property values with an Indirect or Opaque type and the value is the long form Namespace qualified name or AUID of the actual type.

The *reg:escaped* attribute (see rule 5.1 and 5.12) is used to indicate the use of Reg-XML character escaping.

The optional *reg:path* attribute may be used on any object to indicate its XPath Absolute Location Path within its source document. Use this attribute when copying fragments of documents out of their source document without losing their overall context and location.

8.3 Internal DTD

Rule 1 Defines the internal DTD for the Reg-XML document. Rule 1 shall be applied for documents in which rule 5.11.1 is applied at least once. Other documents need not apply Rule 1. Each property that has a Stream type, e.g. EssenceData::Data and EssenceData::SampleIndex, is stored in a separate file. The stream data is treated as an external unparsed entity and the external files shall be declared in the internal DTD.	
Mapping	*<!DOCTYPE {root element name} [* *For each Type Definition Stream:* *<!NOTATION {stream type name} SYSTEM "{stream type id}">* *For each Stream property value:* *<!ENTITY {entity name} SYSTEM "{file location}" NDATA {stream type name}>* *]>*
Notes	{root element name}: the root element name is given by the normative XML element tag in SMPTE ST 395 and shall be qualified with the baseline namespace prefix if the baseline namespace is not the default namespace. {stream type name}: a name corresponding to the Stream type. It is recommended that applications use the symbol of the Stream type if it is unique. {stream type id}: the Stream type identification (an AUID) {entity name}: The name of the entity. It is recommended that applications choose a name that reflects the type of stream data it references. {file location}: the location of the file containing the Stream data. It is recommended that applications choose a filename that reflects the type of stream data it contains.
Examples	1. A Reg-XML document with 3 streams. *<!DOCTYPE MXF [* *<!NOTATION Stream SYSTEM "urn:smpte:ul:060e2b34.01040101.04100200.00000000">* *<!ENTITY audio1 SYSTEM "example/audio1" NDATA Stream>* *<!ENTITY audio2 SYSTEM "example/audio2" NDATA Stream>* *<!ENTITY video1 SYSTEM "example/video1" NDATA Stream>*]>

8.4 Root Element

Rule 2 Defines the root element of the Reg-XML document.	
Mapping	*<{root element name} version="{version}" {namespace declarations}>* *<Extensions>* *For each Extension Meta Dictionary:* *{Extension Meta Dictionary}* *</Extensions>* *{root object}* *</{root element name}>*
Notes	{root element name}: the root element name is defined by the baseline scheme providing the data model for the document. The element name shall be qualified with the baseline namespace prefix if the baseline namespace is not the default namespace.

	{version}: the value of the Preface::Version property. Note: the Preface::Version property must be present in the Preface XML element. {namespace declarations}: All namespaces that are used in the Reg-XML document to qualify attribute and element names shall be declared here. It is recommended that the baseline namespace be declared as the default namespace. Note: If the baseline namespace is declared as the default namespace then the baseline namespace is also defined with a prefix to allow the use of the global attributes such as *reg:uid* and *reg:actualType*. {Extension Meta Dictionary}: The *Extensions* element contains zero or more instances of an Extension Meta Dictionary. The XSD provided in Annex A defines the structure of an Extension Meta Dictionary. {root object}: The result of applying rule 3 to the object at the root of the baseline scheme's data model hierarchy, e.g. *Preface* for MXF and AAF. Note: The *Extensions* element will appear before the {root object}.
Examples	1. A Reg-XML document for the AAF baseline data model with extensions from company X and adhoc extensions. Note that this is an example of an instance document, not a data model, and so does not validate against the schemas for data models defined in this document. <AAF xmlns="http://www.smpte-ra.org/schemas/2001-2/2007/aaf" xmlns:reg=" http://www.smpte-ra.org/schemas/2001-2/2007/aaf" xmlns:cx="http://www.companyx.com/reg" xmlns:this="urn:uuid:fdf33c2-1623-47ab-b17e-cdac2d80590a"> <Extensions> <Extension> <!-- company X extensions --> … </Extension> <Extension> <!-- adhoc extensions --> …</Extension> </Extensions> <Preface>…</Preface> </AAF>

8.5 Object

Rule 3 Defines the mapping of objects.	
Mapping	*<{name} [reg:uid="{id}"]? [reg:path="{absolute location path}"]?>{properties}</{name}>*
Notes	{name}: The Namespace qualified name of the object's Class Definition. [reg:uid]: a *reg:uid* attribute shall be present if the object is contained in a Set. The *reg:uid* attribute, {id}, may be any valid attribute value but shall be unique within the set of objects. It is recommended that applications use the DefinitionObject::Name property value for DefinitionObjects and the unique identifier property value for other objects (e.g. Package::PackageID for Package objects). [reg:path]: a *reg:path* attribute may be included to provide the XPath Absolute Location path of the element in the context of the owning Reg-XML document. {properties}: Apply the property mapping rule, rule 4, for each of the properties contained by the object. The properties can appear in any order and optional properties may be omitted. The InterchangeObject::ObjectClass property is implied by the name and Namespace of the object and may be omitted.

Examples	1. The Sequence object <Sequence>…</Sequence> 2. The DataDefinition object for ‘Sound’ data <DataDefinition reg:uid=”Sound”>…</DataDefinition> 3. An Identification object with its optional contextual path included <Identification reg:path=”/AAF/Preface/IdentificationList/Identification[2]”>…</Identification>

8.6 Property and Property Alias

Rule 4 Defines the mapping of properties.	
Mapping	*Apply one of the following:* *1. Apply rule 4.1 for the Preface::ByteOrder property.* *2. For a property defined by a property alias:* *<{alias} {attributes}>{value}</{alias}>* *3. Otherwise:* *<{name} {attributes}>{value}</{name}>*
Notes	{alias}: The Namespace qualified name for the Property Alias Definition. The name of the original property that is reused shall not be used. {name}: The Namespace qualified name for the property’s Property Definition. {attributes} and {value}: Apply the property value mapping rule, rule 5.
Examples	1. A Duration property <Duration>4500</Duration> 2. A Color property defined by company X <cx:Color>…<cx:Color>

Rule 4.1 Defines the mapping of the Preface::ByteOrder property.	
Mapping	*<{name}>{byte order}</{name}>*
Notes	{name}: The Namespace qualified name for the property’s Property Definition. {byte order}: The Preface::ByteOrder property has type *Int16*. The value #x4949 is mapped onto ‘BigEndian’ and #x4D4D onto ‘LittleEndian’.
Examples	1. A Preface::ByteOrder property for a Reg-XML document converted from a binary file using a little endian byte order <ByteOrder>LittleEndian</ByteOrder>

8.7 Property Values

Rule 5 Defines the mapping for property values.	
Mapping	*Apply the sub-rule that corresponds to the property value type category.*
Notes	None.
Examples	None.

8.7.1 Character

Rule 5.1 Character type category.	
Mapping	*The* reg:escaped *attribute shall be present if Reg-XML character escaping is used and the value shall be set to 'true'* *The value is the character.*
Notes	See also Section 8.1.3.
Examples	1. An 'A' character <cx:ACharacter>A<cx:ACharacter> 2. A Reg-XML character reference to the null character <cx:Acharacter reg:escaped="true">$#x0;<cx:ACharacter> 3. A carriage return character <cx:ACharacter>
<cx:ACharacter>

8.7.2 Enumeration

Rule 5.2 Enumeration type category.	
Mapping	*The enumerated element name*
Notes	None.
Examples	1. Frame layout enumerated value <FrameLayout>FullFrame</FrameLayout>

8.7.3 Extendible Enumeration

Rule 5.3 Extendible Enumeration type category.	
Mapping	*Apply one of the following:* *1. If the enumerated name is defined in the same Extension Meta Dictionary as the Type Definition Extendible Enumeration or is a baseline type:* *{name}* *2. Otherwise the enumerated name qualified by the Extension Meta Dictionary Scheme:*

	{Namespace} ' ' {name}
Notes	{name}: the enumerated element name, including a prefix containing the extendible enumeration type's name (omitting the substring "Type" if present) and an underscore character ('_'). {Namespace}: the SchemeURI defined for the Extension Meta Dictionary that contains the Extendible Enumeration element
Examples	1. The *Sub Clip* PackageUsage defined in the baseline <PackageUsage>Usage_SubClip</PackageUsage> 2. The *CX Clip* PackageUsage defined in the company X Meta Dictionary <PackageUsage>http://www.companyx.com/reg Usage_CXClip</PackageUsage>

8.7.4 Fixed Array

Rule 5.4 Fixed Array type category.	
Mapping	*Apply one of the following rules:* *1. rule 5.4.1 if the elements of the fixed array have a Strong Object Reference base type category* *2. otherwise rule 5.4.2*
Notes	None.
Examples	None.

Rule 5.4.1 Element type with a Strong Object Reference base type category.	
Mapping	*For each strong referenced object apply rule 5.13.*
Notes	None
Examples	None.

Rule 5.4.2 Element type with "other" base type category.	
Mapping	*For each element of the array:* *<{element type name} {attributes}>{value}</{element type name}>*
Notes	{element type name}: the Namespace qualified name of the element type {attributes} and {value}: apply rule 5 for the element.
Examples	1. An array of 2 DateStruct values <cx:values> <DateStruct>2004-12-01Z</DateStruct> <DateStruct>2004-11-08Z</DateStruct> </cx:values>

8.7.5 Indirect

Rule 5.5 Indirect type category.	
Mapping	*The* reg:actualType *attribute shall be present and the attribute's value has the following format:* *[{Namespace} ‘ ‘]? {name}* *Apply rule 5 using the actual type.*
Notes	[{Namespace} ‘ ‘]?: the Namespace for the type and a space if the actual type is not a baseline type {name}: the actual type name
Examples	*1.* An Indirect property value with actual type *UTF16String* <Value reg:actualType=”UTF16String”>This shot is not in focus</Value> *2.* An Indirect property value of an enumerated type from an extension meta dictionary: <Value reg:actualType=”http://company.com/enums Genre”>KIDS</Value>

8.7.6 Integer

Rule 5.6 Integer type category.	
Mapping	*The integer value is written either in decimal notation or hexadecimal notation using the prefix ‘0x’.*
Notes	None.
Examples	1. A Component Length property value using the decimal notation <ComponentLength>1200</ComponentLength> 2. A TrackID property value using the hexadecimal notation <TrackID>0x01</TrackID>

8.7.7 Opaque

Rule 5.7 Opaque type category.	
Mapping	*A* reg:actualType *attribute shall be present and the value is the unique identifier (an* AUID*) of the actual type.* *A* reg:byteOrder *attribute shall be present to indicate the byte order.* *The value is represented as a sequence of bytes in hexadecimal notation. Each byte may be separated by zero or more of the following characters: ‘ ‘ (#x20) , ‘\n’ (#x0A), ‘\t’ (#x09), ‘\r’ (#x0D).*
Notes	The byte order indicates the originating byte order for the opaque data.
Examples	1. Opaque KLV data with big endian byte order <KLVData> < KLVDataValue reg:actualType=”urn:uuid:6207cf79-cbf2-4fdf-911d-43ac6a02fbc3” reg:byteOrder=”BigEndian”> 4C46DC5343565AF042A4 D89516DD98335FCDAB </ KLVDataValue> </KLVData>

8.7.8 Record

Rule 5.8 Record type category.	
Mapping	*Apply one of the following:* 1. *Apply rule 5.8.1 for types with well known string representations:* AUID, DateStruct, PackageID, Rational, TimeStruct, TimeStamp *and* VersionType *2. Otherwise, for each member of the record:* *<{name} {attributes}>{value}</{name}>*
Notes	{name}: the Namespace qualified name of the record member {attributes} and {value}: the result from applying rule 5 to the record member value.
Examples	1. A Identification::ApplicationVersion property value <ApplicationVersion> <major>3</major> <minor>5</minor> <tertiary>7</tertiary > <patchLevel>11</patchLevel> <buildType>VersionPrivateBuild</BuildType> </ApplicationVersion>

Rule 5.8.1 Record types with well-known string representations.	
Mapping	*Apply the rule matching the base type:* *1.* AUID*: see Section 8.1.1 and use* *2.* DateStruct*: Either the value shall conform to the* date *type defined in the XSDL data type specification and shall indicate the time zone relative to UTC, or all date component values are set to zero in canonical representation to indicate "unknown" as per SMPTE ST 377-1.* *3.* PackageID*: urn:smpte:umid:{8 sets of 4 bytes represented in hexadecimal and separated by a '.'} - as defined by SMPTE ST 2029. All component values shall be represented in big endian.* *4.* Rational*: {numerator value} '/' {denominator value}* *5.* TimeStamp*: Either the value shall conform to the* dateTime *type defined in the XSDL data type specification and shall indicate the time zone relative to UTC, or all time stamp component values are set to zero in canonical representation to indicate "unknown" as per SMPTE ST 377-1.* *6.* TimeStruct*: Either the value shall conform to the* time *type defined in the XSDL data type specification and shall indicate the time zone relative to UTC, or all time component values are set to zero in canonical representation indicate "unknown" as per SMPTE ST 377-1.* *7.* VersionType*: {major version value) '.' {minor version value}*
Notes	The *DateStruct* value shall have a time zone indicator, i.e. end with a 'Z' or indicate the time zone relative to UTC. See SMPTE ST 330 for a description of the UMID and SMPTE ST 2029 for the URN

	syntax. {numerator value}: apply rule 5.6, using the decimal notation, to the *numerator* member of the record. {denominator value}: apply rule 5.6, using the decimal notation, to the *denominator* member of the record. The *TimeStamp* value shall have a time zone indicator, i.e. end with a 'Z' or indicate the time zone relative to UTC The *TimeStruct* value shall have a time zone indicator, i.e. end with a 'Z' or indicate the time zone relative to UTC {major version value}: apply rule 5.6, using the decimal notation, to the *major* member of the record {minor version value}: apply rule 5.6, using the decimal notation, to the *minor* member of the record For "unknown" DateStruct, TimeStruct and TimeStamp values, the timezone shall be specified as 'Z'. For "unknown" TimeStruct and TimeStamp values, the fraction of a second value is omitted. These two conditions ensure compliance with *date*, *time* and *dateTime* XSDL canonical representations.
Examples	1. A SMPTE UL *AUID* value <cx:Value>urn:smpte:ul:060e2b34.01040101.04100200.00000000</cx:Value> 2. A UUID *AUID* value <cx:Value>urn:uuid:6207cf79-cbf2-4fdf-911d-43ac6a02fbc3</cx:Value> 3. A *DateStruct* value (UTC time zone) <cx:Value>2004-11-01Z</cx:Value> 4. A *PackageID* value <cx:Value>urn:smpte:umid:060a2b34.01010101.01010f00.00000000.05cba732.1daa11d3.80ad0060.08143e6f</cx:Value> 5. A *Rational* value <cx:Value>48000/1</cx:Value> 6. A *TimeStamp* value (UTC -05:00 time zone) <cx:Value>2004-11-01T15:10:05.400-05:00</cx:Value> 7. A *TimeStruct* value (UTC time zone) <cx:Value>20:10:05.400Z</cx:Value> 8. A *VersionType* value <cx:Value>1.1</cx:Value> 9. Example "unknown" DateStruct, TimeStruct and TimeStamp values: <cx:Value>0000-00-00Z</cx:Value> <cx:Value>00:00:00Z</cx:Value> <cx:Value>0000-00-00T00:00:00Z</cx:Value>

8.7.9 Rename

Rule 5.9 Rename type category.	
Mapping	*The rule corresponding to the renamed type is applied.*

Notes	None.
Exceptions	None.

8.7.10 Set

Rule 5.10 Set type category.	
Mapping	*Apply one of the following rules:* *1. rule 5.10.1 if the elements of the set have a Strong Object Reference base type category* *2. otherwise rule 5.10.2*
Notes	Elements shall be unique within the Set.
Examples	None.

Rule 5.10.1 Element type with a Strong Object Reference base type category.	
Mapping	*For each strong referenced object in the set apply rule 5.13.*
Notes	None
Examples	1. A set of Package objects <Packages> <MaterialPackage>…</MaterialPackage> <SourcePackage>…</SourcePackage> </Packages>

Rule 5.10.2 Element type with an “other” base type category.	
Mapping	*For each element of the set:* *<{element type name} {attributes}>{value}</{element type name}>*
Notes	{element type name}: the Namespace qualified name of the element type {attributes} and {value}: apply rule 5 for the element.
Examples	1. A set of 2 AUID values <cx:values> <AUID>urn:uuid:24cd6eb2-5315-4ce6-ad36-42365e638e6d</AUID> <AUID>urn:uuid:616be4ba-4e94-4e9a-8b5d-ba935467e1a6</AUID> </cx:values> 2. A set of 2 Integer values <cx:values> <UInt8>1</UInt8> <UInt8>0</UInt8> </cx:values>

8.7.11 Stream

Rule 5.11 Stream type category.	
Mapping	*One of either rules 5.11.1 or 5.11.2 shall be applied. Within a single document, the same rule should be applied for all streams throughout. Processors decoding a document shall support both kinds of stream encoding.*
Notes	None.
Examples	None.

Rule 5.11.1 Stream type category, stream data included by entity reference.	
Mapping	*The stream data shall be written to an external file.* *A* reg:stream *attribute shall be present with a value equal to the entity name, which shall be declared in the internal DTD, that corresponds to the external file that contains the stream data.* *An optional* reg:byteOrder *attribute indicates the byte order for the stream data.*
Notes	None.
Examples	1. An XML data stream *<!DOCTYPE MXF [* *<!NOTATION XMLStream SYSTEM "urn:uuid:6207cf79-cbf2-4fdf-911d-43ac6a02fbc3">* *<!ENTITY xmlstream1 SYSTEM "example/xmlstream" NDATA XMLStream>* *…* *<EssenceData>* *<Data reg:stream="xmlstream1"/>*

Rule 5.11.2 Stream type category, stream data referenced as a resource by an XLink Locator Attribute.	
Mapping	*The stream data shall be written to an external resource at a location identifiable with an URI.* *A Locator Attribute* xlink:href*, as defined in XLink, shall be present with a value of an URI that can be resolved to the location of the resource that contains stream data. If the URI specifies a relative location, that location shall be considered as relative to the document containing the stream reference.* *An optional* reg:byteOrder *attribute indicates the byte order for the stream data.*
Notes	None.
Examples	1. A XML data stream specified with a local path: *<EssenceData>* *<Data xlink:href ="./streams/essence/videoStream1"/>* 2. A XML data stream specified by a URL: *<EssenceData>* *<Data xlink:href ="http://www.any.org/essence/videoStream27"/>* 3. A XML data stream specified by a URN that requires further resolution to a location: *<EssenceData>* *<Data xlink:href ="urn:uuid:F2D6240D-0388-4137-8AE8-57EE70E53BBD"/>*

8.7.12 String

Rule 5.12	
String type category.	
Mapping	*Apply one of the following rules:* 1. *rule 5.12.1 if the elements of the String have an Integer base type* *The* reg:escaped *attribute shall be present if Reg-XML character escaping is used and the value shall be set to 'true'* 2. *Otherwise, apply rule 5.1 for each Character in the String. The zero-terminator (null character) shall not be included in the output value.*
Notes	The null character (#x00) is not allowed in a string. See also Section 8.1.3.
Examples	1. A String of Characters with markup and carriage return character escaped <Description>The first shot in the sequence <emph>after</emph> the
explosion</Description> 2. A String of Characters with Reg-XML character escaping <Description reg:escaped="true">Character #1 and #2:$#x01; $#x02;</Description>

Rule 5.12.1	
Element type with a Integer type category.	
Mapping	*Apply rule 5.6 for each Integer value, using the space character (#x20) to separate the values. The zero-terminator shall not be included in the output value.*
Notes	None.
Examples	1. A String of Integer values <cx:Range>1 2 3 4 5</cx:Range>

8.7.13 Strong Object Reference

Rule 5.13	
Strong Object Reference type category.	
Mapping	*Apply rule 3 for the referenced object.*
Notes	None.
Examples	1. A Strong Object Reference to a SourceClip object. <TrackSegment> <SourceClip>…</SourceClip> </TrackSegment>

8.7.14 Variable Array

Rule 5.14	
Variable Array type category.	
Mapping	*Do one of the following:* 1. *Apply rule 5.14.1 if the element type is* Character or the type name contains *StringArray*.

	2. *Apply rule 5.14.2 if the type name is* DataValue *3. Otherwise, apply rule 5.4.*
Notes	None.
Examples	None.

Rule 5.14.1 Variable Array type for string values. This array type is used to represent an array of strings. The null character (#x00) shall not be used in the output .	
Mapping	*For each element of the string array that is made up of a sequence of characters, output the following:* *<{name}>{characters}</{name}>*
Notes	{name}: the Namespace qualified name of a baseline *String* type with element type *Character* {characters}: apply rule 5.12 for each string in the array.
Examples	1. An array of strings <cx:Strings> <UTF16String>The first string</UTF16String> <UTF16String>The second string</UTF16String> <cx:Strings>

Rule 5.14.2 *DataValue* Variable Array type. The elements of this array have type *UInt8*.	
Mapping	*Each* UInt8 *Integer value in the array is represented in hexadecimal notation, separated by zero or more of the following characters: ' ' (#x20) , '\n' (#x0A), '\t' (#x09), '\r' (#x0D).*
Notes	None.
Examples	1. The WaveDescriptor::WaveSummary property value <WaveSummary> 52494646240000005741564566D742010000000010001 0044AC00008858010002001000 </WaveSummary>

8.7.15 Weak Object Reference

Rule 5.15 Weak Object Reference type category.	
Mapping	*Do one of the following:* *1. The value is a reference that targets the* reg:uid *attribute of an object declared in a set of values of the referenced type within in the same Reg-XML Document.* *2. The value is a space separated URI and target pair, where the URI identifies another document or scheme that contains the target identified by the* reg:uid *attribute.* *3. The value is the (globally) unique identifier of the target object, for example an* AUID *or a* PackageID*, that may be resolved to an object either represented within the same Reg-XML document or to an external data source.*

Notes	An external data source may be the SMPTE dictionaries (http://www.smpte-ra.org), a company's internal reference data repository or another authoritative data source.
Examples	1. A weak reference to a locally-declared DataDefinition object <ComponentDataDefinition>Sound</ComponentDataDefinition> 2. A weak reference to an externally-defined contributor: <Contributor>http://refdata.mycompany.org/contributors Ewan_McColl</Contributor> 3. A weak reference to a Package object by its globally-unique identifier: <PrimaryPackage> urn:smpte:umid:0602b34.01010105.01010f20.13000000.05cba732.1daa11d3.80ad0060.08143e6f </PrimaryPackage>

9 Model Mapping Rules (Informative)

The XSDs generated using the model mapping rules is the definitive specification of XSDs for checking syntactic validity of a Reg-XML document.

A Reg-XML file group is syntactically valid if it was generated from a file containing SMPTE registered data using the data mapping rules defined in this specification.

The provisions of underlying specifications whose elements are registered determine semantic validity of Reg-XML file groups. For example, SMPTE ST 377-1 defines the semantic validity of MXF files.

This section defines the model mapping rules for generating W3C XML Schemas (XSDs) from a Reg-XML data model for validation. The XSDs generated using the model mapping rules is the definitive specification for XSDs for checking syntactic conformance of a Reg-XML document.

Application developers are encouraged to supplement the XSDs with additional checks.

It is not a normative requirement for applications to use these particular XSDs, and it is not a normative requirement for applications to use W3C XML Schema.

The XSDs must set the attributes *attributeFormDefault* to ‘unqualified’ and *elementFormDefault* to ‘qualified’.

9.1 Data Types

Rule 1 The data types defined in the baseline namespace	
Mapping	*TargetType* *<simpleType name="TargetType">* *<union memberTypes="token">* *<simpleType>* *<restriction base="reg:AUID"/>* *</simpleType>* *<simpleType>* *<xs:restriction base="reg:PackageIDType"/>* *</simpleType>* *<simpleType>* *<restriction base="string">* *<pattern value="([^\s]+\s)?[^\s]+"/>* *</restriction>* *</simpleType>* *</union>* *</simpleType>*

	Byte order: *<simpleType name="ByteOrderType">* *<restriction base="string">* *<enumeration value="BigEndian"/>* *<enumeration value="LittleEndian"/>* *<restriction>* *</simpleType>* *Hexadecimal byte array:* *<simpleType name="HexByteArrayType">* *<restriction base="string">* *<pattern value="(\s*[0-9a-fA-F][0-9a-fA-F])*\s*"/>* *</restriction>* *</simpleType>* *Structure of Extension Meta Dictionary:* *<include schemaLocation="{metadictionary XSD}"/>*
Notes	*TargetType*: defines the targets for a weak object reference, as per rule 5.15 in Section 8.7.15. Alternative simple types for *AUID* and *PackageID* are provided in rule 6.8.1 in Section 9.6.8. *Byte order*: see Section 8.1.2. *Hexadecimal byte array*: A list of bytes represented in hexadecimal notation. Each byte is separated by zero or more '\s' characters (#x09, #x0A, #x0D, #x20) {metadicationary XSD}: locally resolvable URI for the meta dictionary schema, conforming to the XML schema provided in Annex A of this document.
Examples	None.

9.2 Global Attributes

Rule 2 The global attributes defined in the baseline namespace	
Mapping	*Use for Weak object reference types:* *<attribute name="uid" type="reg:TargetType"/>* *Byte order:* *<attribute name="byteOrder" type="reg:ByteOrderType"/>* *Stream types:* *<attribute name="stream" type="ENTITY"/>* *Indirect types:* *<attribute name="actualType" type="reg:TargetType"/>* *Reg-XML character escaping:* *<attribute name="escaped" type="boolean"/>* *Fragment references:* *<attribute name="path" type="string"/>*

	Source of XLink href definition: *<import namespace="http://www.w3.org/1999/xlink" schemaLocation="{xlink XSD}"/>*
Notes	{xlink XSD}: URI for the definition for an XML Schema definition of XLink , as provided at "http://www.loc.gov/standards/mets/xlink.xsd".
Examples	None.

9.3 Root Element

Rule 3 The root element defined in the baseline scheme *rootElement* attribute.	
Mapping	*<element name="{root element name}"> <complexType> <sequence> <element name="Extensions" minOccurs="0"> <complexType> <sequence> <element ref="reg:Extension" minOccurs="0" maxOccurs="unbounded"/> </sequence> </complexType> </element> <element ref="{root object name}"/> </sequence> <attribute name="version" type="reg:VersionType" use="required"/> </complexType> <unique name="_U_MDSchemeID"> <selector xpath="reg:Extensions/reg:Extension"/> <field xpath="reg:SchemeID"/> </unique> <unique name="_U_MDSchemeURI"> <selector xpath="reg:Extensions/reg:Extension"/> <field xpath="reg:SchemeURI"/> </unique> </element>*
Notes	The *reg:Extension* is defined in the Extension Meta Dictionary XSD provided in Annex A. {root element name}: root element name defined for the baseline scheme in use {root object name}: name of the element at the top of the class inheritance hierarchy for the baseline scheme The root object name element is produced by applying rule 4 to the class in the baseline scheme with the root object name.
Examples	None.

9.4 Class Definitions

<table>
<tr><td colspan="2">Rule 4
Class Definition and contained properties.</td></tr>
<tr><td>Mapping</td><td><element name=”{name}” [abstract=”true”]?>
<complexType>
<complexContent>
<all>
Apply rule 4.1
</all>
[<attribute ref=”reg:uid” use=”required”/>]?
<attribute ref="reg:path" use="optional"/>
</complexContent>
</complexType>
</element>

reg:uid: is declared if the class has a unique identifier property.

abstract: is present and set to ‘true’ if the class is not concrete.</td></tr>
<tr><td>Notes</td><td>{name}: the Class Definition name

reg:path: used to provide the absolute location path of the class within the context of its source document</td></tr>
<tr><td>Examples</td><td>1. The Sequence class

<element name="Sequence">
<complexType>
<all>
<element ref="reg:LinkedGenerationID" minOccurs="0"/>
<element ref="reg:ComponentDataDefinition"/>
<element ref="reg:ComponentLength" minOccurs="0"/>
<element ref="reg:ComponentKLVData" minOccurs="0"/>
<element ref="reg:ComponentUserComments" minOccurs="0"/>
<element ref="reg:ComponentAttributes" minOccurs="0"/>
<element ref="reg:ComponentObjects"/>
</all>
<attribute name="reg:path" use="optional"/>
</complexType>
</element>

2. The DataDefinition class (a sub-class of DefinitionObject)

<element name="DataDefinition">
<complexType>
<all>
<element ref="reg:LinkedGenerationID" minOccurs="0"/>
<element ref="reg:DefinitionObjectIdentification"/>
<element ref="reg:DefinitionObjectName"/>
<element ref="reg:DefinitionObjectDescription" minOccurs="0"/>
</all>
<attribute ref="reg:uid" use="required"/>
<attribute ref="reg:path" use="optional"/>
</complexType>
</element></td></tr>
</table>

Rule 4.1 Properties of a class definition.	
Mapping	*For all properties of a class, including the properties of its superclasses, apply one of the following:* *1. If the property is optional or InterchangeObject::ObjectClass:* *<element name="{name}" ref="{qualified name}" minOccurs="0"/>* *2. If the property is not optional:* *<element name="{name}" ref="{qualified name}"/>*
Notes	{name}: the property name. {qualified name}: the property name qualified with the Namespace prefix
Examples	1. The InterchangeObject::ObjectClass property <element name="ObjectClass" ref="reg:ObjectClass" minOccurs="0"/>

9.5 Property Definitions and Property Alias Definitions

Rule 5 Property Definition and Property Alias Definition	
Mapping	*Apply one of the following:* *1. rule 5.1 if the property definition is the Preface::ByteOrder property* *2. If a property alias definition, the rule is as follows:* *<element name="{alias}" type="{type name}"/>* *3. otherwise, the rule is as follows:* *<element name="{name}" type="{type name}"/>*
Notes	{alias}: The Property Alias Definition name. The name of the original property that is reused is not used for the alias. {name}: the Property Definition name. {type name}: the Namespace qualified property type name.
Examples	1. The Track::TrackID property <element name="TrackID" type="reg:Int32"/>

Rule 5.1 Property Definition for the Preface::ByteOrder property.	
Mapping	*<element name="{name}" type="reg:ByteOrderType"/>*
Notes	{name}: the property name.
Examples	1. ThePreface::ByteOrder property <element name="ByteOrder" type="reg:byteOrder"/>

9.6 Type Definitions

Rule 6 Type Definition.	
Mapping	*Apply all of the following:* *1. the rule that corresponds to the type category* *2. declare the XML element shown below for the type if one of the following is true:* *a. The type category is Character, Integer, Enumeration, Extendible Enumeration, Record, Renamed (if the type category of the renamed type is one of those listed here), Weak Object Reference and the type is defined as part of a Set, Fixed Array or Variable Array type* b. *The type is* String *<element name="{name}" type="{qualified name}"/>*
Notes	{name}: the type name. {qualified name}: the Namespace qualified type name The element declared here will be used in the Set, Fixed Array or Variable Array type mapping. The element for the type *String* will be used in the *StringArray* Variable Array type mapping.
Examples	1. The element declaration for the *AUID* type which is used in a Set type definition <element name="AUID" type="reg:AUID"/>

9.6.1 Character

Rule 6.1 Type Definition Character.	
Mapping	*<complexType name="{name}">* *<simpleContent>* *<extension base="string">* *<attribute ref="reg:escaped" use="optional"/>* *</extension>* *</simpleContent>* *</complexType>*
Notes	{name}: The type name
Examples	1. The Character type <simpleType name="Character"> <simpleContent> <extension base="string"> <attribute ref="reg:escaped" use="optional"/> </extension> </simpleContent> </simpleType>

9.6.2 Enumeration

Rule 6.2 Type Definition Enumeration.	
Mapping	*<simpleType name="{name}">* *<restriction base="token">* *For each enumeration element* *<enumeration value="{enum name}"/>* *</restriction>* </simpleType>
Notes	{name}: The type name {enum name}: the enumerated element's name
Examples	1. VideoSignalType type <simpleType name="VideoSignalType"> <restriction base="token"> <enumeration value="VideoSignalNull"/> <enumeration value="NTSCSignal"/> <enumeration value="PALSignal"/> </restriction> </simpleType>

9.6.3 Extendible Enumeration

Rule 6.3 Type Definition Extendible Enumeration.	
Mapping	*<simpleType name="{name}">* *<union>* *<simpleType>* *<restriction base="token">* *For each enumeration element defined within the same scheme:* *<enumeration value="{enum name}"/>* *</restriction>* *</simpleType>* *<simpleType>* *<restriction base="string">* *For each enumeration defined in Extendible Enumeration Extension:* *<enumeration value="{long form qualified name}"/>* *</restriction>* *</simpleType>* *</union>* *</simpleType>*
Notes	{name}: the type name {enum name}: the enumerated element's name prefixed by the type name (omitting the trailing "Type" if present) and an underscore character ('_'). {long form qualified name}: The SchemeURI defined for the MetaDictionary that contains the Extendible Enumeration Extension, followed by a space (#x20) and then followed by the enumerated name
Examples	1. UsageType defined in the baseline namespace and extended by company X <simpleType name="UsageType"> <union>

```
        <simpleType>
          <restriction base="token">
            <enumeration value="Usage_SubClip"/>
            <enumeration value="Usage_AdjustedClip"/>
            <enumeration value="Usage_TopLevel"/>
            <enumeration value="Usage_LowerLevel"/>
            <enumeration value="Usage_Template"/>
          </restriction>
        </simpleType>
        <simpleType>
          <restriction base="string">
            <enumeration value="http://www.companyx.com/reg Usage_CXClip"/>
          </restriction>
        </simpleType>
      </union>
    </simpleType>
```

9.6.4 Fixed Array

Rule 6.4 Fixed Array type definition.	
Mapping	*Apply one of the following rules:* 1. *rule 6.4.1 if the elements of the fixed array have a Strong Object Reference base type category* 2. *otherwise rule 6.4.2*
Notes	None.
Examples	None.

Rule 6.4.1 Element with a Strong Object Reference base type category.	
Mapping	*<complexType name="{name}"> <choice minOccurs="{len}" maxOccurs="{len}"> if the referenced class is concrete: <element ref="{referenced class name}"/> for all concrete sub-classes: <element ref="{referenced sub-class name}"/> </choice> </complexType>*
Notes	{name}: the type name {len}: the length of the fixed array {referenced class name}: the Namespace qualified name of the referenced class {referenced sub-class name}: the Namespace qualified name of the sub-class of the referenced class
Examples	None.

Rule 6.4.2 Element with an “other” base type category.	
Mapping	*<complexType name=”{name}”> <sequence> <element ref=”{type element name}” minOccurs=”{len}” maxOccurs=”{len}”/> </sequence> </complexType>*
Notes	{name}: the type name {len}: the length of the fixed array {type element name}: the Namespace qualified name of the XML element corresponding to the array element type
Examples	None.

9.6.5 Indirect

Rule 6.5 Type Definition Indirect.	
Mapping	*<complexType name=”{name}”> <complexContent mixed=”true”> <restriction base=”anyType”> <sequence> <any minOccurs=”0” maxOccurs=”unbounded” processContents=”skip”/> <sequence> <attribute ref=”reg:actualType” use=”required”/> <attribute ref=”reg:escaped” use=”optional”/> </restriction> </complexContent> </complexType>*
Notes	{name}: The type name Note: XSD validators will skip the contents of this element and therefore the element content is not validated.
Examples	1. The Indirect type <complexType name="Indirect"> <complexContent mixed="true"> <restriction base="anyType"> <sequence> <any minOccurs="0" maxOccurs="unbounded" processContents="skip”/> </sequence> <attribute ref=”reg:actualType” use=”required”/> <attribute ref=”reg:escaped” use=”optional”/> </restriction> </complexContent> </complexType>

9.6.6 Integer

Rule 6.6 Integer type definition.	
Mapping	*<simpleType name=”{name}”> <union>*

	<simpleType> *<restriction base="{XSDL integer type name}"/>* *</simpleType>* *<simpleType>* *<restriction base="string">* *<pattern value="{pattern}"/>* *</restriction>* *</simpleType>* *</union>* *</simpleType>*
Notes	{name}: The type name {XSD integer type name} and {pattern}: The XSDL integer type name and pattern for each size (in bytes) and sign combination are as follows: UInt8: {XSDL integer type name} is *unsignedByte* and {pattern} is *0x[0-9a-fA-F]{1,2}* Int8: {XSDL integer type name} is *byte* and {pattern} is *0x[0-9a-fA-F]{1,2}* UInt16: {XSDL integer type name} is *unsignedShort* and {pattern} is *0x[0-9a-fA-F]{1,4}* Int16: {XSDL integer type name} is *short* and {pattern} is *0x[0-9a-fA-F]{1,4}* UInt32: {XSDL integer type name} is *unsignedInt* and {pattern} is *0x[0-9a-fA-F]{1,8}* Int32: {XSDL integer type name} is *integer* and {pattern} is *0x[0-9a-fA-F]{1,8}* UInt64: {XSDL integer type name} is *unsignedLong* and {pattern} is *0x[0-9a-fA-F]{1,16}* Int64: {XSDL integer type name} is *long* and {pattern} is *0x[0-9a-fA-F]{1,16}*

Examples	1. A signed Integer type with size equal to 4 bytes
	`<simpleType name="Int32">`
	`<union>`
	`<simpleType>`
	`<restriction base="integer"/>`
	`</simpleType>`
	`<simpleType>`
	`<restriction base="string">`
	`<pattern value="0x[0-9a-FA-F]{1,8}"/>`
	`</restriction>`
	`</simpleType>`
	`</union>`
	`</simpleType>`
	2. An unsigned Integer type with size equal to 2 bytes, using *memberTypes* for brevity
	`<simpleType name="Int32">`
	`<union memberTypes="unsignedShort">`
	`<simpleType>`
	`<restriction base="string">`
	`<pattern value="0x[0-9a-FA-F]{1,4}"/>`
	`</restriction>`
	`</simpleType>`
	`</union>`
	`</simpleType>`

9.6.7 Opaque

Rule 6.7 Opaque type definition.	
Mapping	*<complexType name="{name}">* *<simpleContent>* *<extension base="reg:HexByteArrayType">*

	<attribute ref="reg:actualType" use="required"/> *<attribute ref="reg:byteOrder" use="required"/>* *</extension>* *</simpleContent>* *</complexType>*
Notes	{name}: the type name
Examples	1. The Opaque type <complexType name="Opaque"> <simpleContent> <extension base="reg:HexByteArrayType"> <attribute ref="reg:actualType" use="required"/> <attribute ref="reg:byteOrder" use="required"/> </extension> </simpleContent> </complexType>

9.6.8 Record

Rule 6.8 Type Definition Record.	
Mapping	*Apply one of the following:* 1. *Apply rule 6.8.1 for types with well known string representations, namely* AUID, DateStruct, PackageID, Rational, TimeStruct, TimeStamp *and* VersionType *2. Otherwise:* *<complexType name="{name}">* *<sequence>* *for each record member* *<element name="{member name}" type="{member type}"/>* *</sequence>* *</complexType>*
Notes	{name}: The type name {member name}: the record member name with its first letter capitalized {member type}: the Namespace qualified member type name
Examples	1. ApplicationVersion type <complexType name="ApplicationVersion"> <sequence> <element name="major" type="reg:UInt16"/> <element name="minor" type="reg:UInt16"/> <element name="tertiary" type="reg:UInt16"/> <element name="patchLevel" type="reg:UInt16"/> <element name="buildType" type="reg:ProductReleaseType"/> </sequence> </complexType>

Rule 6.8.1 Record types with well-known string representations.	
Mapping	*Apply the rule matching the type:* 1. *AUID*: *<simpleType name="AUID">*

```
    <restriction base="xs:anyURI">
      <pattern
          value="urn:smpte:ul:([0-9a-fA-F]{8}\.){3}[0-9a-fA-F]{8}"/>
      <pattern
        value="urn:uuid:[0-9a-fA-F]{8}-([0-9a-fA-F]{4}-){3}[0-9a-fA-F]{12}"/>
    </restriction>
  </simpleType>
```

2. *DateStruct*:

```
  <simpleType name="DateStruct">
    <union>
      <simpleType>
        <restriction base="date">
          <pattern value=".+(((\+|\-)\d\d:\d\d)|Z)"/>
        </restriction>
      </simpleType>
      <simpleType>
        <restriction base="xs:string">
          <enumeration value="0000-00-00Z"/>
        </restriction>
      </simpleType>
    </union>
  </simpleType>
```

3. *PackageID*:

```
  <simpleType name="PackageIDType">
    <restriction base="string">
      <pattern value=" urn:smpte:umid:([0-9a-fA-F]{8}\.){7}[0-9a-fA-F]{8}"/>
    </restriction>
  </simpleType>
```

4. *Rational*:

```
  <simpleType name="Rational">
    <restriction base="string">
      <pattern value="\-?\d{1,10}(/\-?\d{1,10})?"/>
    </restriction>
  </simpleType>
```

5. *TimeStruct*:

```
  <simpleType name="TimeStruct">
    <union>
      <simpleType>
        <restriction base="time">
          <pattern value=".+(((\+|\-)\d\d:\d\d)|Z)"/>
        </restriction>
      </simpleType>
      <simpleType>
        <restriction base="string">
          <enumeration value="00:00:00Z"/>
        </restriction>
      </simpleType>
    </union>
  </simpleType>
```

6. *TimeStamp*:

```
  <simpleType name="TimeStamp">
    <union>
```

	<simpleType> *<restriction base="dateTime">* *<pattern value=".+(((\+|\-)\d\d:\d\d)|Z)"/>* *</restriction>* *</simpleType>* *<simpleType>* *<restriction base="string">* *<enumeration value="0000-00-00T00:00:00Z"/>* *</restriction>* *</simpleType>* *</union>* *</simpleType>* 7. *VersionType*: *<simpleType name="VersionType">* *<restriction base="string">* *<pattern value="\-?\d{1,3}\.\-?\d{1,3}"/>* *</restriction>* *</simpleType>*
Notes	None.
Examples	None.

9.6.9 Rename

Rule 6.9 Type Definition Rename.	
Mapping	*Apply one of the following:* *1. Rule 6.9.1 if the actual type results in a XSD simple type definition* *2. Rule 6.9.2 if the actual type results in a XSD complex type definition*
Notes	None.
Examples	None.

Rule 6.9.1 Type Definition Rename for actual types that result in a simple type definition.	
Mapping	*<simpleType name="{name}">* *<restriction base="{renamed type name}"/>* *</simpleType>*
Notes	{name}: The type name {renamed type name}: the Namespace qualified renamed type name.
Examples	1. PositionType <simpleType name="PositionType"> <restriction base="reg:Int64"/> </simpleType>

Rule 6.9.2 Type Definition Rename for actual types that result in a complex type definition.	
Mapping	*<complexType name="{name}">* * <complexContent>* * <extension base="{renamed type name}"/>* * </complexContent>* *</complexType>*
Notes	{name}: The type name {renamed type name}: the Namespace qualified renamed type name.
Examples	None.

9.6.10 Set

Rule 6.10 Set type definition.	
Mapping	*Apply one of the following rules:* *1. rule 6.10.1 if the elements of the set have a Strong Object Reference base type category* *2. otherwise rule 6.10.2*
Notes	None.
Examples	None.

Rule 6.10.1 Element with a Strong Object Reference base type category.	
Mapping	*Apply the following:* *<complexType name="{name}">* * <choice minOccurs="0" maxOccurs="unbounded">* *if the referenced class is concrete:* * <element ref="{referenced class name}"/>* *for all concrete sub-classes:* * <element ref="{referenced sub-class name}"/>* * </choice>* * </complexType>*
Notes	{name}: The type name {referenced class name}: the Namespace qualified name for the referenced class {referenced sub-class name}: the Namespace qualified name for the sub-class of the referenced class
Examples	1. A Set of Data Definitions <complexType name="DataDefinitionStrongReferenceSet"> <choice minOccurs="0" maxOccurs="unbounded"> <element ref="reg:DataDefinition"/> </choice> </complexType>

Rule 6.10.2 Elements with a “other” base type category.	
Mapping	*<complexType name=”{name}”> <sequence> <element ref=”{type element name}” minOccurs=”0” maxOccurs=”unbounded”/> </sequence> </complexType>*
Notes	{name}: the type name {type element name}: the Namespace qualified name of the XML element corresponding to the element type
Examples	1. A Set of Weak Object References to DataDefinition <complexType name="DataDefinitionWeakReferenceSet"> <sequence> <element ref="reg:DataDefinitionWeakReference" minOccurs="0" maxOccurs="unbounded"/> </sequence> </complexType>

9.6.11 Stream

Rule 6.11 Stream type definition.	
Mapping	*<complexType name=”{name}”> <attribute ref="reg:stream" use="optional"/> <attribute ref="xlink:href" use="optional"/> <attribute ref="aaf:byteOrder" use="optional"/> </complexType>*
Notes	{name}: the type name
Examples	1. A Stream type definition <complexType name="Stream"> <attribute ref="reg:stream" use="optional"/> <attribute ref="xlink:href" use="optional"/> <attribute ref="aaf:byteOrder" use="optional"/> </complexType>

9.6.12 String

Rule 6.12 String type definition:	
Mapping	*<complexType name=”{name}”> <simpleContent> <extension base="string"> <attribute ref="reg:escaped" use="optional"/> </extension> </simpleContent> </complexType>*
Notes	{name}: the type name

Examples	1. String type with UTF-16 representation <complexType name="UTF16String"> <simpleContent> <extension base="string"> <attribute ref="reg:escaped" use="optional"/> </extension> </simpleContent> </complexType>

9.6.13 Strong Object Reference

Rule 6.13 Strong Object Reference type definition.	
Mapping	*<complexType name="{name}"> <choice> if the referenced class is concrete: <element ref="{referenced class name}"/> for all concrete sub-classes: <element ref="{referenced sub-class name}"/> </choice> </complexType>*
Notes	{name}: The type name {referenced class name}: the Namespace qualified name for the referenced class {referenced sub-class name}: the Namespace qualified name for the sub-class of the referenced class
Examples	1. Strong Object Reference to a Segment <complexType name="SegmentStrongReference"> <choice> <element ref="reg:EdgeCode"/> <element ref="reg:EssenceGroup"/> <element ref="reg:GPITrigger"/> <element ref="reg:CommentMarker"/> <element ref="reg:DescriptiveMarker"/> <element ref="reg:Filler"/> <element ref="reg:OperationGroup"/> <element ref="reg:NestedScope"/> <element ref="reg:Pulldown"/> <element ref="reg:ScopeReference"/> <element ref="reg:Selector"/> <element ref="reg:Sequence"/> <element ref="reg:SourceClip"/> <element ref="reg:TextClip"/> <element ref="reg:Timecode"/> <element ref="reg:TimecodeStream"/> </choice> </complexType>

9.6.14 Variable Array

Rule 6.14 Variable Array type definition.	
Mapping	*Apply one of the following rules:* *1. rule 6.14.1 if the elements of the array have a Strong Object Reference base type*

	category 2. *rule 6.14.2 if the element type is Character or the type name contains* StringArray 3. *rule 6.14.3 if the type name is* DataValue 4. *otherwise rule 6.14.4*
Notes	None.
Examples	None.

Rule 6.14.1 Element with a Strong Object Reference base type category.	
Mapping	*<complexType name="{name}"> <choice minOccurs="0" maxOccurs="unbounded"> if the referenced class is concrete: <element ref="{referenced class name}"/> for all concrete sub-classes: <element ref="{referenced sub-class name}"/> </choice> </complexType>*
Notes	{name}: The type name {referenced class name}: the Namespace qualified name of the referenced class {referenced sub-class name}: the Namespace qualified name of the sub-class of the referenced class
Examples	None.

Rule 6.14.2 Arrays of string values.	
Mapping	*<complexType name="{name}"> <sequence> <element ref="{referenced string type}" minOccurs="0" maxOccurs="unbounded"/> </sequence> </complexType>*
Notes	{name}: the type name. {referenced string type}: A string type that has the same element type as that defined for the string array type. See model mapping rule 6.
Examples	2. A type definition for an array of UTF16 string values: <complexType name="UTF16StringArray"> <sequence> <element ref="reg:UTF16String" minOccurs="0" maxOccurs="unbounded"/> </sequence> </complexType>

Rule 6.14.3 The *DataValue* type.	
Mapping	*<simpleType name="DataValue">* *<restriction base="reg:HexByteArrayType"/>* *</simpleType>*
Notes	None.
Examples	None.

Rule 6.14.4 Variable Array type definition with elements with “other” base type.	
Mapping	*<complexType name=”{name}”>* *<sequence>* *<element ref=”{type element name}” minOccurs=”0” maxOccurs=”unbounded”/>* *</sequence>* *</complexType>*
Notes	{name}: the type name {type name}: the Namespace qualified name of the XML element corresponding to the array element type
Examples	None.

9.6.15 Weak Object Reference

Rule 6.15 Weak Object Reference type definition.	
Mapping	*<simpleType name=”{name}”>* *<restriction base="reg:TargetType"/>* *</simpleType>*
Notes	{name}: the type name
Examples	1. A Weak Object Reference to Data Definition <simpleType name="DataDefinitionWeakReference"> <restriction base="reg:TargetType"/> </simpleType>

Annex A Extension Meta Dictionary Structure (Normative)

The meta dictionary XML schema defined in document element SMPTE ST 2001-1a shall be used for the representation of baseline meta dictionaries and extension meta dictionaries for Reg-XML documents.

This meta dictionary schema shall be incorporated into data-model-defining schemas using the XML schema *include* directive.

Note 1: The use of the include directive allows the meta dictionary schema to be reused to specify extension metadata dictionaries, both externally to and from within Reg-XML documents. The *include* directive allows a set of schema definitions to be incorporated into another schema, whereby the definitions take on the namespace of the including schema. This is different from the *import* directive, which allows definitions of a pre-existing namespace to be used in the definition of a schema that builds on them.

Note 2: As the document element is not intended for use as a standalone XML schema, no *target namespace* is defined for the schema file and for its definitions. The definitions of the schema are always included into another schema before interpretation, with the namespace of the included definitions then provided by the context into which they are included. No namespace is set to avoid the ambiguity that defining two or more namespace names for each definition could cause.

Annex B Schema for Baseline Meta Dictionaries (Informative)

An XML schema that can be used for the definition of baseline meta dictionaries based on the meta dictionary structure defined in Annex A is available in document element SMPTE ST 2001-1b.

Annex C Deriving Data Models from SMPTE Metadata Registers (Normative)

The SMPTE metadata registers contain definitions for groups, elements and types which can be used to derive a Reg-XML data model. This section defines how to create a baseline or extension data model from the SMPTE metadata registers.

The source for deriving a Reg-XML data model shall be the Metadata Groups Register, SMPTE ST 395. The Groups Register contains definitions for groups, and for their contents. The content items are referred to in the remainder of this annex as group elements. The Metadata Types Register, SMPTE ST 2003:2012, provides detailed type definition information for types referenced by the group elements in the Groups Register.

The groups in the Groups Register map to Reg-XML Class Definitions. The group elements in the Groups Register map to Reg-XML Property Definitions. The types in the Types Register, which are referenced by group elements, map to Reg-XML Type Definitions. The following sections define the mapping in more detail.

C.1 Data Model Identification

For purposes of identification and global reference, a Reg-XML data model requires a unique SchemeID and SchemeURI parameter.

The general requirements for SchemeID and SchemeURI are set out in section 7. When a SchemeID is provided in accordance with section 7, the SchemeURI is determined from it in accordance with SMPTE ST 2029

In the absence of SchemeURI and SchemeID for a data model containing definitions from the Groups Register, applications shall use a SchemeURI and SchemeID derived as follows from a namespace assigned to the Groups Register. The Groups Register derived data model SchemeURI shall have the following structure as stated in SMPTE ST 395:

http://www.smpte-ra.org/reg/395/2013

and the SchemeID shall be the UL of the root Node of the Groups Register

In the case of SMPTE ST 395 class 13 and class 14 registrations, the Scheme URI shall be the namespace URI as registered, and the SchemeID shall be a version 5 (SHA-1-hashed) UUID generated from the SchemeURI name using the URL namespace ID as specified in IETF RFC 4122.

In the case of unregistered Data Models (such as ad-hoc extensions), the SchemeID shall be a UUID per RFC 4122, and the SchemeURL shall be the text representation of that UUID per RFC 4122.

For example, suppose a hypothetical organization "erehwon.org" has been granted a class 13 registration with subid 42 (2a in hexdecimal) and defines a data model with a namespace URI: http://schemas.erehwon.org/location with subid 23 (17 in hexadecimal).

They might elect to use the SchemeID:

```
urn:smpte:ul:060e2834.027f0101.0d2a1700.00000000
```

which is the ST 2029 representation of the UL; or they might use the SchemeID:

```
urn:uuid:1554d688-2fd4-5d10-a1ce-16e15de21bb9
```

which is the RFC 4122 representation of the version 5 (SHA-1-hashed) UUID generated from "http://schemas.erehwon.org/location"

Note: this example was generated in Python by the command string "import uuid; uuid.uuid5(uuid.NAMESPACE_URL, 'http://schemas.erehwon.org/location')"

C.2 Definition Mapping

The following sub-sections describe how definitions contained in the Groups and Types Registers are mapped to the definitions of a Reg-XML data model.

Note: The mapping only applies to types in the Types Register that are referenced by group elements in the Groups Register.

C.2.1 Groups to class definitions

The group definitions (leaf rows) in the Groups Register are equivalent to Reg-XML Class Definitions. The Class Definition attributes are derived from the fields defined in the Groups Register as follows:

Attribute	Mapping	ST 395 Field	ST 395 Section
Identification	value of	Universal label	4.5.7
Symbol	value of	Symbol	4.5.8
Name	value of	Name	4.5.10
Description	value of	Definition	4.5.11
Parent Class	value of	Parent Group	4.5.12
Is Concrete	value of	Concrete	4.5.13

C.2.2 Group elements to property definitions

The group element definitions (link rows) in the Groups Register are equivalent to Reg-XML Property Definitions. The Property Definition attributes are derived from the fields defined in the Groups Register as follows:

Attribute	Mapping	ST 395 Field	ST 395 Section
Identification	value of	Item	4.5.16
Symbol	value of	Symbol	4.5.8
Name	value of	Name	4.5.10
Description	value of	Definition	4.5.11
Type	value of Type UL of	Item	4.5.16
Member Of	value of	Universal label	4.5.7
Is Optional	value of	Contents optional	4.5.20
Is Unique Identifier	value of	Is Unique Identifier	4.5.19
Local Identification	value of	Contents local tag	4.5.17

C.2.3 Group element types to type definitions

The type definitions in the Types Register, which are referenced by group elements in the Groups Register, are equivalent to Reg-XML Type Definitions. The attributes common to all Type Definitions are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Identification	value of	URN representation of universal label	4.4.6
Symbol	value of	Symbol	4.4.8
Name	value of	Name	4.4.9
Description	value of	Definition	4.4.10

The following section describes how the Reg-XML types are derived from the Types Register.

C.2.3.1 Array Type with non-zero Type Size

The Array Type with non-zero Type Size field maps to the Fixed Array Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Element Type	value of	Base Type	4.4.13
Element Count	value of	Type Size	4.4.12

C.2.3.2 Array Type with zero Type Size

The Array Type with zero Type Size field maps to the Variable Array Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Element Type	value of	Base Type	4.4.13

C.2.3.3 Character Type

The Character Type maps to the Character Type Definition. There are no additional attributes.

C.2.3.4 Enumerated Type

The Enumerated Type maps to the Enumeration Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Element Type	value of	Base Type	4.4.13
Elements (name and value)	list of values of	Facet Symbol and Value	4.4.16

C.2.3.5 Extendible Enumerated Type

The Extendible Enumerated Type maps to the Extendible Enumeration Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Elements (name and value)	list of values of	Facet Symbol and Value	4.4.16

C.2.3.6 Indirect Type

The Indirect Type maps to the Indirect Type Definition. There are no additional attributes.

C.2.3.7 Integer Type

The Integer Type maps to the Integer Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Size	value of	Type Size	4.4.12
Is Signed	value of isSigned qualifier of	Type Qualifiers	4.4.15

C.2.3.8 Opaque Type

The Opaque Type maps to the Opaque Type Definition. There are no additional attributes.

C.2.3.9 Renamed Type

The Renamed Type maps to the Rename Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Renamed Type	value of	Base Type	4.4.13

C.2.3.10 Record Type

The Record Type maps to the Record Type Definition. The attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Members (name and type)	list of values of	Facet Symbol and Type	4.4.16

C.2.3.11 Set Type

The Set Type maps to the Set Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Element Type	value of	Base Type	4.4.13

C.2.3.12 Stream Type

The Stream Type maps to the Stream Type Definition. There are no additional attributes.

C.2.3.13 String Type

The String Type maps to the String Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Element Type	value of	Base Type	4.4.13

C.2.3.14 Strong Reference Type

The Strong Reference Type maps to the Strong Object Reference Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Referenced Type	value of	Base Type	4.4.13

C.2.3.15 Weak Reference Type

The Weak Reference Type maps to the Weak Object Reference Type Definition. The additional attributes are derived from the fields defined in the Types Register as follows:

Attribute	Mapping	ST 2003 Field	ST 2003 Section
Referenced Type	value of	Base Type	4.4.13
Target Set	list of values of	Facet Symbol	4.4.16