4 -

Description Definition Language

Jane Hunter¹ and Claude Seyrat²

¹DSTC Pty, Ltd., Australia, ²Expway, Paris, France

4.1 INTRODUCTION

The DDL provides the foundations for the MPEG-7 standard. It provides the language for defining the structure and content of MPEG-7 documents. The DDL is not a modeling language such as Unified Modeling Language (UML) but a schema language to represent the results of modeling audiovisual data, (i.e. descriptors and description schemes) as a set of syntactic, structural and value constraints to which valid MPEG-7 descriptors, description schemes, and descriptions must conform. It also provides the syntactic rules by which users can combine, extend and refine existing description schemes and descriptors to create application-specific description definitions or schemas. The purpose of a schema is to define a class of XML (Extensible Markup Language) documents. The purpose of an MPEG-7 schema is to define a class of MPEG-7 documents, MPEG-7 instances are XML documents that conform to a particular MPEG-7 schema (expressed in the DDL) and that describe audiovisual content.

According to the MPEG-7 DDL requirements [1], the DDL must be capable of expressing structural, inheritance, spatial, temporal, spatiotemporal and conceptual relationships between the elements within a description scheme and between description schemes. It must provide a rich model for links and references between one or more descriptions and the data that it describes. It must be platform and application independent, machine-readable and preferably human-readable. It must be capable of specifying descriptor data types, both primitive (integer, text, date, time) and composite (histograms, enumerated types).

In addition, a DDL Parser is required that is capable of validating the syntax of MPEG-7 schemas, which comprises description schemes (content and structure) and descriptors (data types). Given an MPEG-7 description (encoded in XML), the parser must also be able to check its conformance to the rules expressed in the corresponding MPEG-7 schema (a set of description schemes and descriptors defined using the DDL).

At the 51st MPEG meeting in Noordwijkerhout in March 2000, it was decided to adopt the W3C's XML Schema Language as the MPEG-7 DDL. However, because XML
Schema Language has not been designed specifically for audiovisual content, certain extensions are necessary in order to satisfy all of the MPEG-7 DDL requirements. Hence the DDL consists of the following logical components, which are described in Sections 4.3, 4.4 and 4.5: XML Schema structural components, XML Schema data types and MPEG-7-specific extensions.

In the remainder of this chapter, we will describe the events that have led to the current DDL and its key components and features. Complete specifications of the DDL can be found in the MPEG-7 DDL Final Draft International Standard (FDIS) [2] and the W3C XSLT Schema Recommendations [3-5].

4.2 HISTORICAL BACKGROUND

In response to the MPEG-7 Call for Proposals in October 1998, the MPEG-7 DDL evaluation team compared and evaluated ten DDL proposals at the MPEG-7 Test and Evaluation Meeting in Lancaster in February 1999. A summary report [6] was produced that concluded that XML [7] should be used as the syntax for the MPEG-7 DDL. There was also a consensus that the MPEG-7 DDL must support the validation of structural, relational and data typing constraints as well as the expression of semantics through the addition of richer constraints such as inheritance.

Although none of the proposals could satisfy all of the requirements, it was decided to base the DDL on Distributed Systems Technology Centre (DSTC’s) proposal, P547 [8], with the integration of ideas and components from other proposals and contributors. In addition, the strategy was to continue monitoring and liaising with related efforts in the W3C community, in particular the XML Schema [9], XML Linking Language (XLink) [10] and XML Path Language (XPath) [11] Working Groups (WGs).

In May 1999, the XML Schema WG produced the first version of a two-part working draft of the XML Schema Language: XML Schema Part 1: Structures [4] and XML Schema Part 2: Data types [5]. Preliminary encoding of the Multimedia Description Schemes (MDSs) [12] using XML Schema Language demonstrated its suitability as a basis for the DDL. However, reservations were raised at the 48th MPEG meeting in Vancouver in July 1999 concerning MPEG-7’s dependency on the output and timeline schedule of W3C XML Schema WG. As a result, the decision was made to develop a proprietary MPEG-7-specific language in parallel with the XML Schema Language developments within W3C. A new grammar based on DSTC’s proposal but using MPEG-7 terminology (description schemes and descriptors) and with modifications to ensure simple mapping to XML Schema, was developed.

However, when XML Schema moved to the Last Call for Review stage in March 2000, it was decided (at the 51st MPEG meeting in Noordwijkerhout in March 2000) to adopt XML Schema Language, with additional MPEG-7-specific extensions, as the DDL. This decision was made in recognition of XML Schema’s growing stability and expected widespread adoption, the availability of open source XML Schema parsers and the release of Last Call for Review Working Drafts by the W3C.

A detailed evaluation of XML Schema revealed that although XML Schema satisfies the majority of the MPEG-7 DDL requirements, there are certain requirements that are not satisfied and some features that are problematic. A list of problem issues and
feature requests was submitted to the XML Schema WG in response to the Last Call for Review [13]. Certain high priority features, which are not defined in XML Schema, have been implemented as MPEG-7-specific extensions. Validation of these extensions will be through extensions to existing XML Schema parsers implemented during the development of MPEG-7 parsers.

### 4.3 XML SCHEMA STRUCTURAL COMPONENTS

The purpose of this section is to describe the language constructs provided by XML Schema, which can be used to constrain the content structure and attributes associated with MPEG-7 description schemes and descriptors.

XML Schema consists of three categories of schema components. The primary components are:

- Namespaces and the schema wrapper around the definitions and declarations;
- Element declarations;
- Attribute declarations;
- Type definitions: simple, complex, derived and anonymous.

The secondary components are:

- Attribute group definitions;
- Model group definitions;
- Identity-constraint definitions;
- Notation declarations.

The third group is the "helper" components, which contribute to the other components and cannot stand alone:

- Annotations;
- Model groups;
- Parties;
- Wildcards.

In this chapter, we only describe those features of most importance to MPEG-7 - the primary components and group definitions. Details of other components not described here can be found in the XML Schema Recommendations [3–5].

#### 4.3.1 Namespaces and the Schema Wrapper

XML namespaces [14] provide a simple method for assigning universally unique names to element types or attribute names within a XML document, so that they can be reused in other XML documents. According to [14], an XML namespace is a collection of names, identified by a Uniform Resource Identifier (URI) reference, which are used in XML documents as element types and attribute names. Qualified names consist of a namespace prefix (which maps to a URI reference), a colon separator and the local part (an element type or attribute name). This combination produces identifiers for schema components, which are universally unique and thus reusable, for example, mpeg7:VideoSegmentType.
In the context of MPEG-7, the namespace mechanism enables descriptors and description schemes from multiple different MPEG-7 schemas to be reused and combined to create new schemas. Every schema definition must begin with a preamble in order to identify the current namespace and other imported namespaces. The mandatory preamble consists of an XML element ‘schema’, which includes the following attributes:

- `xmlns:` A URI to the XML Schema namespace;
- `xmlns:mpeg7:` A URI to the MPEG7 schema to be used for validating MPEG-7 description schemes and descriptors;
- `targetNamespace`: The URI by which the current schema is to be identified;
- `xmlns:xxx`: References to other imported schemas and abbreviations for referring to definitions in these external schemas, for example, `xmlns:dc` in the example below associates the prefix `dc` with the Dublin Core namespace, which is located at the given URI.

```xml
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:mpeg7="http://www.mpeg7.org/2001/MP7-7_Schema"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  targetNamespace="http://www.data.org/2001/DS16_MPEG-7_Schema"/>
```

### 4.3.2 Element Declarations

Element declarations enable the appearance in document instances of elements with specific names and types. An element declaration specifies a type definition for a schema element either explicitly or by reference, and may provide occurrence (through the `minOccurs` and `maxOccurs` attributes) and default information (through the `default` attribute). For example, the element declaration below associates the name `Country` with an existing type definition, `countryCode`, specifies that the default value for the `Country` element is ‘en’ (England) and that the `Country` element can occur zero or more times.

```xml
<element name="Country" type="countryCode" default="en"
  minOccurs="0" maxOccurs="unbounded"/>
```

The values for `minOccurs` and `maxOccurs` are computed in the following way:

- `minOccurs`:
  - the actual value of the `minOccurs` attribute, if present,
  - otherwise 1.

- `maxOccurs`:
  - `unbounded`, if the `maxOccurs` attribute equals `unbounded`;
  - otherwise the actual value of the `maxOccurs` attribute, if present;
  - otherwise 1.

Sometimes it is preferable to reference an existing element rather than declare a new element.

```xml
<element ref="Country" minOccurs="1"/>
```
This declaration references an existing element (Country) that was declared elsewhere in the schema. The value of the ref attribute must reference a global element, that is, one that has been declared under schema rather than as part of a complex type definition. The consequence of this declaration is that an element called Country must appear at least once in an instance document and its content must be consistent with that element’s type, the countryCode.

4.3.3 Attribute Declarations

Attribute declarations enable the appearance in document instances of attributes with specific names and types by associating an attribute name with a simple data type. Within an attribute declaration, the use attribute specifies presence (required|optional|prohibited). The default value of use is optional. The declared attribute can have a fixed or a default value specified by the default or fixed attribute. The declaration below indicates that the appearance of a lang attribute (in the element Annotation) is optional and its default value is ‘en-uk’.

```xml
<attribute name="lang" type="language" use="optional" default="en-uk"/>
<element name="Annotation"/>
<complexType/>
    <attribute ref="lang"/>
</complexType>
</element>
```

Below is a valid instance of an element of the above type:

```xml
<Annotation lang="en-us"/>
```

4.3.4 Type Definitions

In XML Schema there is a fundamental distinction between type definitions (which create new types) and declarations, which enable the appearance in document instances of elements and attributes with specific names and types.

Type definitions define internal schema components, which can be used in other schema components such as element or attribute declarations or other type definitions. For example, below we first define the (simple) type PostCode that is a string of length 7:

```xml
<simpleType name="Postcode"/>
    <restriction base="string">
        <length value="7"/>
    </restriction>
</simpleType>
```

We can then declare elements or attributes, which are of this type and which are to appear in instantiated XML documents. For example, we can declare an element MyPostcode, which is of type Postcode, which is to appear in an instance document:

```xml
<element name="MyPostcode" type="Postcode"/>
```

XML Schema provides simple, complex and derived type definitions, as described below.
Simple type definitions

Simple types cannot have children elements and cannot carry attributes. Both elements and attributes can be declared to have simple types. XML Schema provides a large number of simple types through a set of built-in primitive types and a set of built-in derived types (derived from the primitive types) [5]. These built-in data types are described in Section 4.4. In addition to the built-in simple types, new simple types can be derived by the application of restrictions to other simple types. These restrictions are specified by facets such as the enumeration facet on a string or the minInclusive and maxInclusive facets on an integer, as shown in the examples below. The facets that are applicable to each data type are listed in Appendix C of [5].

```
<simpleType name="directionType">
  <restriction base="string">
    <enumeration value="left"/>
    <enumeration value="right"/>
  </restriction>
</simpleType>

<simpleType name="unsigned6">
  <restriction base="nonNegativeInteger">
    <minInclusive value="0"/>
    <maxInclusive value="63"/>
  </restriction>
</simpleType>

<element name="direction" type="directionType"/>
<element name="length" type="unsigned6"/>
```

Below are valid instances of the defined elements:

```
<direction>left</direction>
<length>36</length>
```

In addition to atomic (indivisible) simple types, XML Schema also provides two aggregate simple types: list types and union types. List types are composed of sequences of atomic types. Union types enable element or attribute values to be instances of a type drawn from the union of multiple atomic and list types. These two aggregate types are described in Sections 4.4 and 4.5.

Complex type definitions

Unlike simple types, complex types allow children elements in their content and may carry attributes. Complex type definitions provide

- Constraints on the appearance and nature of attributes;
- Constraints on the appearance and nature of children elements;
- Derivation of complex types from other simple or complex types through extension or restriction.
The set of the rules that describe the content of an element is called a content model. It is possible to constrain the nature of the content model of a complexType to the following:

- **empty** — no child elements only attributes;
- **mixed** — character data appears between elements and their children. The mixed attribute must be set to 'true';
- **complexContent** — the default content type which consists of elements and attributes. Three compositors are provided to structure unnamed groups of elements within complexContent:
  - **sequence** — the elements may appear in the same order in which they are declared;
  - **choice** — only one of the elements may appear in an instance;
  - **all** — all of the elements may appear once or not at all and in any order.
- **simpleContent** — use when deriving a complexType from a simpleType because we want to add an attribute to a simpleType. This indicates that the new type contains only character data (i.e. does not contain any elements) but may also have attributes associated with it.

New complex types are defined using the **complexType** element and such definitions typically contain a set of element declarations, element references and attribute declarations.

Below is an example of a complex type with a complex content model and a valid instance of this example:

```xml
<complexType name="OrganizationType">
  <sequence>
    <element name="OrgName" type="string"/>
    <element name="ContactPerson" type="IndividualType"
      minOccurs="1" maxOccurs="unbounded"/>
    <element name="Address" type="PlaceType"
      minOccurs="1" maxOccurs="1"/>
  </sequence>
  <attribute name="id" type="ID" use="required"/>
</complexType>

<element name="ProdComp" type="OrganizationType"/>
```

Below is a valid instance corresponding to the above complexType definition:

```xml
<ProdComp id="ORG754">
  <OrgName>DSTC Pty Ltd</OrgName>
  <ContactPerson>Liz Armstrong</ContactPerson>
  <Address>University of Qld</Address>
</ProdComp>
```

The consequence of this definition is that any **ProdComp** elements appearing in an MPEG-7 description must consist of an **OrgName** element, one or more **ContactPerson** elements and one **Address** element. The first of these elements will contain a string, the second will contain the complexType **IndividualType** and the third will contain the complexType **PlaceType**. Finally, any element whose type is declared to be **Organization** must also appear with an attribute called **id**, which must contain an **ID** (unique identifier).
To define a type whose content is empty, we define a type that allows only elements in its content, but we do not actually declare any elements, only attributes. Below is an example of the empty content model and a valid instance of this example:

```xml
<element name="Price">
  <complexType>
    <attribute name="currency" type="currencyCode"/>
    <attribute name="value" type="decimal"/>
  </complexType>
</element>

<Price currency="EU" value="423.46"/>
```

Below is an example of the mixed content model and a valid instance of the example:

```xml
<element name="Introduction">
  <complexType mixed="true"/>
  <sequence>
    <element name="Name" type="string"/>
  </sequence>
</element>

<Introduction>Dear Ms. <Name>Notty Wilson</Name>, </Introduction>
```

**Derived types**

It is possible to derive new complexTypes by extension or restriction of simple or complex base type definitions.

A complex type extends another by having additional content model particles at the end of the other definition's content model, or by having additional attribute declarations or both. A new complexType can be derived by extending a simpleType through the addition of attributes. To indicate that the content model of the new type contains only character data with attributes and no elements, the simpleContent element is used. In the following example, the source and target attributes are added to the simple string base type to define the RelationType.

```xml
<complexType name="RelationType">
  <simpleContent>
    <extension base="string">
    <attribute name="source" type="IDREF" use="optional"/>
    <attribute name="target" type="IDREF" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Relation" type="RelationType"/>

<Relation source="#edge1" target="#edge2" morph/>
```

A new complexType may also be derived by extending an existing complexType. In the example below, the PersonName type is extended through the addition of a new Nickname element, to create the FriendName type. The complexContent element is required.
to indicate that we intend to restrict or extend the content model of a complex type and that the new type contains only elements and no character data.

```xml
<complexType name="PersonNameType">
  <sequence>
    <element name="Title" type="string" minOccurs="0"/>
    <element name="Forename" type="string" minOccurs="0" maxOccurs="unbounded"/>
    <element name="Surname" type="string" minOccurs="1" maxOccurs="1"/>
  </sequence>
</complexType>

<element name="PersonName" type="PersonNameType"/>

<complexType name="FriendNameType">
  <complexContent>
    <extension base="PersonNameType">
      <sequence>
        <element name="Nickname" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="FriendName" type="FriendNameType"/>

Below are valid instances of the elements declared above:

```
<PersonName>
  <Title>Prof.</Title>
  <Forename>Simon</Forename>
  <Forename>Daniele</Forename>
  <Forename>Benjamin</Forename>
  <Surname>Kaplan</Surname>
</PersonName>

<FriendName>
  <Title>Dr.</Title>
  <Forename>Jocelyn</Forename>
  <Surname>Richards</Surname>
  <Nickname>Jo</Nickname>
</FriendName>
```

A type definition whose declarations or facets are in a one-to-one relation with those of another specified type definition, with each in turn restricting the possibilities of the one it corresponds to, is said to be a restriction. The specific restrictions might include narrowed ranges or reduced alternatives. Members of a type, A, whose definition is a restriction of the definition of another type, B, are always members of type B as well.

In the example below, the SimpleName type is derived through restriction of the PersonName type. The restriction is on the occurrence constraints of the Title and Forename elements that are no longer unbounded but restricted to be equal to 1.

```xml
<complexType name="SimpleNameType">
  <complexContent>
```
```
Below is a valid instance of an element of the above type because it has precisely one Title and one Forename element:

```xml
<SimpleName>
  <Title>Prof.</Title>
  <Forename>Simon</Forename>
  <Surname>Kaplan</Surname>
</SimpleName>
```

XML Schema allows derived types to appear in a description instead of expected types. In the preceding example, a description could use the FriendlyNameType everywhere a PersonNameType is expected. In this case, the derived type must be identified in the instance using the xsi:type attribute.

```xml
<PersonName xsi:type="FriendlyNameType">
  <Title>Dr.</Title>
  <Forename>Jocelyn</Forename>
  <Surname>Richards</Surname>
  <Nickname>Jo</Nickname>
</PersonName>
```

**Anonymous type definitions**

Schemas can be constructed by defining named types and then declaring elements that reference the types using the 'element name=.. type=..' construction as illustrated in the examples shown above. This style of schema construction is straightforward but it can become unwieldy if many of the defined types are only referenced once and contain few constraints. In these cases, a type can be more succinctly defined as an anonymous type, which saves the overhead of having to be named and explicitly referenced. For example, the element below contains a complexType definition that is unnamed or anonymous:

```xml
<element name="Affiliation">
  <complexType>
    <choice>
      <element name="Organization" type="OrganizationType"/>
      <element name="PersonGroup" type="PersonGroupType"/>
    </choice>
  </complexType>
</element>
```
4.3.5 Group Definitions

The attributeGroup and group elements provide mechanisms for creating and naming groups of attributes and groups of elements respectively. Such groups can then be incorporated by reference into complexType definitions:

```xml
<attributeGroup name="person_attributes">
  <attribute name="weight" type="decimal"/>
  <attribute name="height" type="decimal"/>
  <attribute name="age" type="integer"/>
</attributeGroup>

<complexType name="Person">
  <sequence>
    <element ref="Name"/>
  </sequence>
  <attributeGroup ref="person_attributes"/>
</complexType>
```

Three compositors (sequence, choice and all) are also provided to construct unnamed groups of elements within complex content. These are described in the previous section on ComplexType Definitions.

In the example below, the ContactGroup is defined as a choice between the two elements, Organization and Person. The PublisherType is then defined as a sequence of ContactGroup and Address with an id attribute.

```xml
<group name="ContactGroup">
  <choice>
    <element ref="Organization"/>
    <element ref="Person"/>
  </choice>
</group>

<complexType name="PublisherType">
  <sequence>
    <group ref name="ContactGroup"/>
    <element ref="Address"/>
  </sequence>
  <attribute name="id" type="ID" use="optional"/>
</complexType>
```

Example of a valid instance:

```xml
<Publisher id="CRC123">
  <Organization>DSTC Pty Ltd<Organization/>
  <Address>Uni. Of Qld<Address>
</Publisher>
```

4.4 XML SCHEMA DATA TYPES

This section describes the built-in primitive data types, the built-in derived data types and mechanisms for defining customized derived data types such as facets, lists and union data
types. These facilities can be used to constrain the possible values of MPEG-7 descriptors within instantiated descriptions.

4.4.1 Built-in Primitive Data Types

The following built-in primitive data types are provided within XML Schema Data types. Precise details including the lexical and canonical representations and the constraining facets for each of these data types can be found in [5]:

*string*: character strings;
*boolean*: (true/false);
*decimal*: finite-length sequence of decimal digits separated by a period as a decimal indicator;
*float*: IEEE single-precision 32-bit floating-point type. Floats have a lexical representation of a mantissa (decimal number) followed by the character 'E' or 'e' followed by an exponent (integer), for example, 12.78E-2;
*double*: IEEE double-precision 64-bit floating-point type. Doubles have a lexical representation of a mantissa (decimal number) followed by the character 'E' or 'e' followed by an exponent (integer), for example, 12.78E-2;
*duraion*: a duration of time represented in the [ISO8601] [15] format PnYnDTnHnMnSnS, where nY represents the number of years, nM the number of months, nD the number of days, 'T' is the date or time separator, nH the number of hours, nM the number of minutes and nS the number of seconds. For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours and 30 minutes, one would write: P1Y2M3DT10H30M;
*dateTime*: a specific instant of time represented in the [ISO 8601] [15] extended format CCYY-MM-DDThh:mm:ss, where 'CC' represents the century, 'YY' the year, 'MM' the month and 'DD' the day, 'T' is the date or time separator and 'hh', 'mm', 'ss' represent hour, minute and second, respectively;
*time*: an instant of time that recurs everyday, represented using the format: hh:mm:ss.sss with optional following time zone (TZ) indicator. For example, to indicate 1:20 p.m. for Eastern Standard Time, which is five hours behind Coordinated Universal Time (UTC), one would write: 13:20:00-05:00;
*date*: a calendar date, represented using the CCYY-MM-DD. For example, to indicate May the 31st, 1999, one would write: 1999-05-31;
*gYearMonth*: a specific Gregorian month in a specific Gregorian year, represented using the format CCYY-MM;
*gYear*: a Gregorian calendar year, specified using the format CCYY;
*gMonthDay*: a Gregorian date that recurs, specifically a day of the year such as the third of May, and which is represented using the format – MM-DD;
*gDay*: a Gregorian day that recurs, specifically a day of the month such as the 5th of the month. It is represented using the format: – DD;
*gMonth*: a Gregorian month that recurs every year. It is represented using the format: – MM–;
*base64Binary*: base64-encoded arbitrary binary data represented by finite-length sequences of binary octets;
*hexBinary*: arbitrary hex-encoded binary data represented by finite-length sequences of binary octets in which each binary octet is encoded as a character tuple, consisting
of two hexadecimal digits ([0-9a-fA-F]) representing the octet code. For example, '0FB7' is a hex encoding for the 16-bit integer 4023 (whose binary representation is 111101101111);

anyURI: a Uniform Resource Identifier Reference (URI), which can be absolute or relative and may have an optional fragment [16];

Qname: represents XML qualified names that consist of tuples [namespace name, local part], where namespace name is an any URI and local part is an NCName;

NOTATION: the NOTATION attribute type from [XML 1.0 [7]], which is the set of all names of notations declared in the current schema. NOTATION should not be used directly in a schema. Only data types that are derived from NOTATION by specifying a value for enumeration can be used in a schema and NOTATION should be used only on attributes.

4.4.2 Built-in Derived Data Types

A number of built-in derived data types (derived by applying facets to the primitive data types) are provided within XML Schema Data types. The complete definitions for the built-in derived data types, which are listed below, can be found in [5]:

normalizedString: white space normalized strings that do not contain the carriage return (#xD), line feed (#xA) nor tab (#x9) characters;

token: tokenized strings that do not contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces;

language - the set of all strings that are valid language identifiers as defined in RFC, 1766 [17];

NMTOKEN: the NMTOKEN attribute type from XML 1.0 (Second Edition) [7] - the set of tokens that match the Nmtoken production shown in Reference 7;

NMTOKENS: the NMTOKENS attribute type from Reference 7 - the set of finite, nonzero-length sequences of NMTOKENs;

Name - the set of all strings that match the Name production of XML 1.0 [7];

NCName: XML 'non-colonized' Names - the set of all strings that match the NCName production of Namespaces in XML [14];

ID: the ID attribute type from XML 1.0 [7];

IDREF, IDREFS: the IDREF and IDREFS attribute types from [XML 1.0] [7];

ENTITY, ENTITIES: the ENTITY and ENTITIES attribute types from XML 1.0 [7];

integer: a finite-length sequence of decimal digits with an optional leading sign. If the sign is omitted, '+1' is assumed. For example: -1, 0, 12678 967 543 233, +100,000.

The value space of integer is the infinite set \{..., -2, -1, 0, 1, 2, ...\};

nonPositiveInteger: derived from integer by setting maxInclusive to 0, the value space is the infinite set \{..., -2, -1, 0\};

negativeInteger: derived from integer by setting maxInclusive to -1, the value space is the infinite set \{..., -2, -1\};

nonNegativeInteger: derived from integer by setting minInclusive to 0, the value space is the infinite set \{0, 1, 2, ...\};

positiveInteger: derived from integer by setting minInclusive to 1, the value space is the infinite set \{1, 2, ...\};
long: derived from integer by setting the value of maxInclusive to be 9223372036854775807 and minInclusive to be –9223372036854775808;
int: derived from long by setting the value of maxInclusive to be 2147483647 and minInclusive to be –2147483648;
short: derived from int by setting the value of maxInclusive to be 32767 and minInclusive to be –32768;
byte: derived from short by setting the value of maxInclusive to be 127 and minInclusive to be –128;
unsignedLong: derived from nonNegativeInteger by setting the value of maxInclusive to be 18446744073709551615;
unsignedInt: derived from unsignedLong by setting the value of maxInclusive to be 4294967295;
unsignedShort: derived from unsignedInt by setting the value of maxInclusive to be 65535;
unsignedByte: derived from unsignedShort by setting the value of maxInclusive to be 255.

4.4.3 Facets

A derived data type is defined by applying constraining facets to a primitive data type or another derived data type. The table below lists the facets, which are provided to generate customized data types.

<table>
<thead>
<tr>
<th>Facets</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounds facets</td>
<td>minInclusive, minExclusive maxInclusive, maxExclusive.</td>
</tr>
<tr>
<td>Numeric facets</td>
<td>totalDigits, fractionDigits</td>
</tr>
<tr>
<td>Pattern facet</td>
<td>Pattern</td>
</tr>
<tr>
<td>Enumeration facet</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Length facets</td>
<td>Length, minLength, maxLength</td>
</tr>
<tr>
<td>White space facet</td>
<td>White space</td>
</tr>
</tbody>
</table>

The example below illustrates the application of the minInclusive and maxInclusive facets to a float data type to restrict elements of type height to between 0.0 and 120.0:

```xml
<simpleType name="height">
  <restriction base="float">
    <minInclusive value="0.0"/>
    <maxInclusive value="120.0"/>
  </restriction>
</simpleType>
```

The example below illustrates the use of the pattern facet. The data type is constrained to strings that match a specific pattern defined using a regular expression. The following example restricts elements of type PhoneNum to strings of three digits followed by a dash followed by four digits. The precise syntax of regular expressions can be found in Appendix F of [5]:

```xml
<simpleType name="PhoneNum">
  <restriction base="string">
```
The example below illustrates the application of the enumeration facet to a string to restrict the possible values of 'temporal relation' to: before, during, after or contains. The enumeration facet can be applied to almost every simple type (except the boolean type) to limit the type to a set of distinct values:

```xml
<simpleType name="temporal_relation"/>
<restriction base="string">
  <enumeration value="before"/>
  <enumeration value="during"/>
  <enumeration value="after"/>
  <enumeration value="contains"/>
</restriction>
</simpleType>
```

4.4.4 The List Data Type

List types are composed of sequences of atomic types, separated by white space. XML Schema has three built-in list types, NMTOKENS, IDREFS and ENTITIES. In addition, new list types can be created by derivation from existing atomic types. List types cannot be created from existing list types or from complex types but facets (length, minLength, maxLength and enumeration) can be applied to derive new list types.

Examples of list type definitions and a valid instance:

```xml
<simpleType name="integerVector">
  <list itemType="integer"/>
</simpleType>

<simpleType name="integerVector4">
  <restriction base="integerVector">
    <length value="4"/>
  </restriction>
</simpleType>

<integerVector4>5 11 2</integerVector4>
```

4.4.5 The Union Data Type

Union types enable element or attribute values to be one or more instances of one type drawn from the union of multiple atomic and list types. In the example below, the element Unsigned6OrDirection can have a value that is of either type Unsigned6 or Direction, because it is defined as the union of these two types.

```xml
<element name="Unsigned6OrDirection">
  <simpleType>
    <union memberTypes="unsigned6 directionType"/>
  </simpleType>
</element>
```
The examples below are both valid instances of this element:

```xml
<Unsigned6OrDirection> 45 </Unsigned6OrDirection>
<Unsigned6OrDirection> left </Unsigned6OrDirection>
```

### 4.5 MPEG-7-SPECIFIC EXTENSIONS

In order to satisfy the MPEG-7 DDL requirements, it has been necessary to add the following features to XML Schema:

- Array and matrix data types;
- Built-in derived data types `basicTimePoint` and `basicDuration`.

#### 4.5.1 Array and Matrix Data Types

MPEG-7 requires DDL mechanisms to:

- restrict the size of multidimensional matrices to a predefined facet value in a schema definition;
- restrict the size of one-dimensional arrays or multidimensional matrices to an attribute at time of instantiation.

Using the `list` data type, two methods are provided for specifying sizes of (1D) arrays and multidimensional matrices. A new `mpeg7:dimension` facet, which is a list of positive integers, is provided to enable the specification of the dimensions of a fixed-size matrix. Because the `dimension` facet is an MPEG-7 extension that is not compliant with XML Schema Language, it needs to be wrapped inside the `<annotation>`<appinfo> to ensure XML Schema parsers ignore it and MPEG-7 parsers will validate (and possibly process) it.

The example below illustrates the definition and instantiation of an integer matrix with 3 rows and 4 columns:

```xml
<complexType name="IntMatrix2D">
    <list itemType="integer">
        <annotation><appinfo>
            <mpeg7:dimension value="unbounded unbounded"/>
        </appinfo></annotation>
    </list>
</complexType>

<complexType name="IntMatrix3x4">
    <restriction base="IntMatrix2D">
        <annotation><appinfo>
            <mpeg7:dimension value="3 4"/>
        </appinfo></annotation>
    </restriction>
</complexType>

<element name="IntMat3x4" type="IntMatrix3x4"/>

<IntMat3x4>
  5 8 9 4
  7 6 1 2
  1 3 5 8
</IntMat3x4>
```
The special `mpeg7:dim` attribute is also provided to support parameterized array and matrix sizes. It specifies the dimensions to be applied to a list type at the time of instantiation and is defined in the mpeg-7 namespace as a list of positive integers.

```xml
<complexType name="dim">
  <list itemType="integer"/>
</complexType>

<complexType name="NDimIntegerArray">
  <simpleContent>
    <extension base="listOfInteger">
      <attribute ref="mpeg7:dim"/>
    </extension>
  </simpleContent>
</complexType>

<element name="IntegerMatrix" type="NDimIntegerArray"/>
```

In the following example, a matrix with 2 rows and 4 columns is specified at the time of instantiation using `mpeg7:dim`.

```xml
<IntegerMatrix mpeg7:dim="2 4">
  1 2 3 4
  5 6 7 8
</IntegerMatrix>
```

### 4.5.2 Built-in Derived Data Types

In addition to the built-in derived types provided by XML Schema Datatypes, the following built-in data types are also provided by MPEG-7 to explicitly satisfy the requirements of MPEG-7 implementers:

The `basicTimePoint` data type, specifies a time point according to the Gregorian dates, day time and the TZ. The format is based on the ISO 8601 [15] standard. To reduce conversion problems only a subset of the ISO 8601 formats is used [2].

The `basicDuration` data type specifies the duration of a time period according to days and time of day. The format is based on the ISO 8601 [15] standard. To reduce conversion problems, only a subset of the ISO 8601 formats is used [2]. Fractions of a second are specified according to the `basicTimePoint` datatype.

### 4.6 CONCLUSION

This chapter has provided an overview of the MPEG-7 DDL. XML Schema has been chosen as the basis for the DDL, because of its widespread adoption as a schema language for constraining the structure and content of XML documents, its ability to satisfy the MPEG-7 DDL requirements and the ready availability of XML Schema tools and parsers. Consequently this chapter has primarily provided an overview of the XML Schema Language. In time we hope that the MPEG-7-specific extensions to XML Schema will be incorporated as part of a Type library provided within future versions of XML Schema – this will make the MPEG-7 DDL fully XML Schema-compliant.
REFERENCES


   http://www.w3.org/TR/xmlschema-0/.

   http://www.w3.org/TR/xmlschema-1/.

   http://www.w3.org/TR/xmlschema-2/.


---

1 Although MPEG documents are in principle nonpublic documents, many of them are made public and can be accessed at the MPEG Home Page, http://mpeg.telecomitalia.it/ . Nonpublic MPEG documents may be obtained through the MPEG Head of Delegation of the respective country.