XML Schema Part 2: Datatypes

W3C Working Draft 22 September 2000

This version:
http://www.w3.org/TR/2000/WD-xmlschema-2-20000922/
(in XML and HTML, with a schema and DTD including datatype definitions, as well as a schema for built-in datatypes only, in a separate namespace.)
Latest version:
http://www.w3.org/TR/xmlschema-2/
Previous version:
http://www.w3.org/TR/2000/WD-xmlschema-2-20000407/
Editors:
Paul V. Biron (Kaiser Permanente, for Health Level Seven) <Paul.V.Biron@kp.org>
Ashok Malhotra (IBM) <petsa@us.ibm.com>

Abstract

XML Schema: Datatypes is part 2 of a two-part draft of the specification for the XML Schema definition language. This document proposes facilities for defining datatypes to be used in XML Schemas as well as other XML specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs) for specifying datatypes on elements and attributes.

Status of this document

This is an internal working draft for review by members of the Working Group.

It has not been reviewed by the XML Schema Working Group and the Working Group has not agreed to its publication. Note that not that all sections of the draft represent the current consensus of the WG. Different sections of the specification may well command different levels of consensus in the WG. Public comments on this draft will be instrumental in the WG's deliberations.

This working draft incorporates all WG decisions through 2000-08-02, and some decisions taken since then.

Although the Working Group does not anticipate further changes to the functionality described here, this is still a working draft, subject to change. The present version should be implemented only by those interested in providing a check on its design or by those preparing for an implementation of the Candidate Recommendation. *The Schema WG will not allow early implementation to constrain its*

ability to make changes to this specification prior to final release.

During the Candidate Recommendation phase, although feedback based on implementation experience is welcome, there are certain aspects of the design presented herein where the Working Group is particularly interested in feedback. These are designated *priority feedback* aspects of the design, and identified as such in editorial notes throughout this draft.

A list of current W3C working drafts can be found at http://www.w3.org/TR/. They may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress".

Several "note types" are used throughout this draft:

issue [Issue (issue-name):]
: something on which the editors are seeking comment.

editorial note [Ed. Note:]
: something the editors wish to call to the attention of the reader. To be removed prior to the recommendation becoming final.

note [Note:]
: something the editors wish to call to the attention of the reader. To remain in the final recommendation.

Table of contents

Appendices

---

1 Introduction

1.1 Purpose

The [XML 1.0 Recommendation] specification defines limited facilities for applying datatypes to document content in that documents may contain or refer to DTDs that assign types to elements and attributes. However, document authors, including authors of traditional *documents* and those transporting *data* in XML, often require a higher degree of type checking to ensure robustness in document understanding and data interchange.

The table below offers two typical examples of XML instances in which datatypes are implicit: the instance on the left represents a billing invoice, the instance on the right a memo or perhaps an email message in XML.

Data oriented	Document oriented

<invoice>
  <orderDate>1999-01-21</orderDate>
  <shipDate>1999-01-25</shipDate>
  <billingAddress>
  <name>Ashok Malhotra</name>
  <street>123 IBM Ave.</street>
  <city>Hawthorne</city>
  <state>NY</state>
  <zip>10532-0000</zip>
  </billingAddress>
  <voice>555-1234</voice>
  <fax>555-4321</fax>
</invoice>
	<memo importance='high'
	  date='1999-03-23'>
	  <from>Paul V. Biron</from>
	  <to>Ashok Malhotra</to>
	  <subject>Latest draft</subject>
	  <body>
	  We need to discuss the latest
	  draft <emph>immediately</emph>.
	  Either email me at <email>
	  mailto:paul.v.biron@kp.org</email>
	  or call <phone>555-9876</phone>
	  </body>
	</memo>

The invoice contains several dates and telephone numbers, the postal abbreviation for a state (which comes from an enumerated list of sanctioned values), and a ZIP code (which takes a definable regular form). The memo contains many of the same types of information: a date, telephone number, email address and an "importance" value (from an enumerated list, such as "low", "medium" or "high"). Applications which process invoices and memos need to raise exceptions if something that was supposed to be a date or telephone number does not conform to the rules for valid dates or telephone numbers.

In both cases, validity constraints exist on the content of the instances that are not expressible in XML DTDs. The limited datatyping facilities in XML have prevented validating XML processors from supplying the rigorous type checking required in these situations. The result has been that individual applications writers have had to implement type checking in an ad hoc manner. This specification addresses the need of both document authors and applications writers for a robust, extensible datatype system for XML which could be incorporated into XML processors. As discussed below, these datatypes could be used in other XML-related standards as well.

1.2 Requirements

The [XML Schema Requirements] document spells out concrete requirements to be fulfilled by this specification, which state that the XML Schema Language must:

1. provide for primitive data typing, including byte, date, integer, sequence, SQL & Java primitive data types, etc.;
2. define a type system that is adequate for import/export from database systems (e.g., relational, object, OLAP);

3. distinguish requirements relating to lexical data representation vs. those governing an underlying information set;
4. allow creation of user-defined datatypes, such as datatypes that are derived from existing datatypes and which may constrain certain of its properties (e.g., range, precision, length, format).

1.3 Scope

This portion of the XML Schema Language discusses datatypes that can be used in an XML Schema. These datatypes can be specified for element content that would be specified as #PCDATA and attribute values of various types in a DTD. It is the intention of this specification that it be usable outside of the context of XML Schemas for a wide range of other XML-related activities such as [XSL] and [RDF Schema].

1.4 Terminology

The terminology used to describe XML Schema Datatypes is defined in the body of this specification. The terms defined in the following list are used in building those definitions and in describing the actions of a datatype processor:

[Definition:] for compatibility
A feature of this specification included solely to ensure that schemas which use this feature remain compatible with [XML 1.0 Recommendation]

[Definition:] may
Conforming documents and processors are permitted to but need not behave as described.

[Definition:] match
(Of strings or names:) Two strings or names being compared must be character for character the same.

[Definition:] must
Conforming documents and processors are required to behave as described; otherwise they are in error.

[Definition:] error
A violation of the rules of this specification; results are undefined. Conforming software may detect and report an **error** and may recover from it.

1.5 Constraints and Contributions

This specification provides three different kinds of normative statements about schema components, their representations in XML and their contribution to the schema-validation of information items:

[Definition:] Constraint on Schemas
Constraints on the schema components themselves, i.e. conditions components must satisfy to be components at all. Largely to be found in Datatype components (§4).

[Definition:] Schema Representation Constraint
Constraints on the representation of schema components in XML. Some but not all of these are expressed in Schema for Datatype Definitions (normative) (§A) and DTD for Datatype Definitions (non-normative) (§B). Largely to be found in XML representation of datatype definitions (§5).

[Definition:] Validity Contribution
Constraints expressed by schema components which information items must satisfy to be schema-valid. Largely to be found in Datatype components (§4).

2 Type System

This section describes the conceptual framework behind the type system defined in this specification. The framework has been influenced by the [ISO 11404] standard on language-independent datatypes as well as the datatypes for [SQL] and for programming languages such as Java.

The datatypes discussed in this specification are computer representations of well known abstract concepts such as *integer* and *date*. It is not the place of this specification to define these abstract concepts; many other publications provide excellent definitions.

2.1 Datatype

[Definition:] In this specification, a **datatype** is a 3-tuple, consisting of a) a set of distinct values, called its value space, b) a set of lexical representations, called its lexical space, and c) a set of facets that characterize properties of the value space, individual values or lexical items.

2.2 Value space

[Definition:] A **value space** is the set of values for a given datatype. Each value in the **value space** of a datatype is denoted by one or more literals in its lexical space.

The value space of a given datatype can be defined in one of the following ways:

- defined axiomatically from fundamental notions (intensional definition) [see primitive]
- enumerated outright (extensional definition) [see enumeration]
- defined by restricting the value space of an already defined datatype to a particular subset with a given set of properties [see derived]
- defined as a combination of values from one or more already defined value space(s) by a specific construction procedure [see list and union]

value spaces have certain properties. For example, they always have the property of cardinality, some definition of *equality* and may be ordered by which individual values within the value space can be compared to one another. The properties of value spaces that are recognized by this specification are defined in Fundamental facets (§2.5.1).

2.3 Lexical space

In addition to its value space, each datatype also has a lexical space.

[Definition:] A **lexical space** is the set of valid *literals* for a datatype (literals may appear as one or more character information items as defined in [XML Information Set]).

For example, "100" and "1.0E2" are two different literals from the lexical space of float which both denote the same value. The type system defined in this specification provides a mechanism for schema designers to control the set of values and the corresponding set of acceptable literals of those values for a datatype.

2.4 Canonical Lexical Representation

While the dataypes defined in this specification have, for the most part, a single lexical representation i.e. each value in the datatype's value space is denoted by a single literal in its lexical space, this is not always the case. The example in the previous section showed two literals for the datatype float which denote the same value. Similarly, there may be several literals for one of the date or time datatypes that denote the same value using different timezone indicators. In such cases, this specification defines a [Definition:] **canonical lexical representation**, or **canonical representation** for short, which selects a set of literals from among the valid set of literals for the datatype such that each literal maps to a single value in the value space and vice-versa.

2.5 Facets

[Definition:] A **facet** is a single defining aspect of a value space. Generally speaking, each facet characterizes a value space along independent axes or dimensions.

The facets of a datatype serve to distinguish those aspects of one datatype which *differ* from other datatypes. Rather than being defined solely in terms of a prose description the datatypes in this specification are defined in terms of the *synthesis* of facet values which together determine the value space and properties of the datatype.

Facets are of two types: *fundamental* facets that define the datatype and *non-fundamental* or *constraining* facets that constrain the permitted values of a datatype.

2.5.1 Fundamental facets

[Definition:] A **fundamental facet** is an abstract property which serves to semantically characterize the values in a value space.

These properties are discussed in this section.

2.5.1.1 Equal

Every value space supports the notion of equality, with the following rules:

- for any two instances of values from the value space *(a,b)*, either *a* is equal to *b*, denoted *a = b*, or *a* is not equal to *b*, denoted *a != b*;
- there is no pair of instances (*a, b*) of values from the value space such that both *a = b* and *a != b*;
- for every value a from the value space, *a = a*;
- for any two instances *(a, b)* of values from the value space, *a = b* if and only if *b = a*;
- for any three instances *(a, b, c)* of values from the value space, if *a = b* and *b = c*, then *a = c*.

On every datatype, the operation Equal is defined in terms of the equality property of the value space: for any values *a, b* drawn from the value space, *Equal(a,b)* is true if *a = b*, and false otherwise.

By definition, given value space*A* and value space*B* where *A* and *B* are not related by , for every pair of values *a* from *A* and *b* from *B*, *a != b*.

2.5.1.2 Order

[Definition:] An **order relation** on a value space is a mathematical relation which imposes a total order on the members of the value space.

[Definition:] A value space, and hence a datatype, is said to be **ordered** if there exists an order-relation defined for that value space.

order relations have the following rules:

- for every pair *(a, b)* from the value space, either *a* < *b* or *b* < *a*, or *a* = *b*;
- for every triple *(a, b, c)* from the value space, if *a* < *b* and *b* < *c*, then *a* < *c*.

NOTE: The fact that this specification does not define an order-relation for some datatype does not mean that some other application cannot treat that datatype as being ordered.

2.5.1.3 Bounds

[Definition:] A value space is **bounded above** if there exists a unique value *U* in the value space such that, for all values *v* in the value space, *v* <= *U*. [Definition:] The value *U* is said to be an **upper bound** of the value space.

[Definition:] A value space is **bounded below** if there exists a unique value *L* in the space such that, for all values *v* in the value space, *L* <= *v*. [Definition:] The value *L* is then said to be a **lower bound** of the value space.

[Definition:] A datatype is **bounded** if its value space has both an upper bound and a lower bound.

2.5.1.4 Cardinality

[Definition:] Every value space has associated with it the concept of **cardinality**. Some value spaces are finite, some are countably infinite while still others are uncountably infinite. A datatype is said to have the cardinality of its value space.

It is sometimes useful to categorize value spaces (and hence, datatypes) as to their cardinality. There are two significant cases:

- value spaces that are finite
- value spaces that are countably infinite

2.5.1.5 Numeric

[Definition:] A datatype is said to be **numeric** if its values are conceptually quantities (in some mathematical number system).

[Definition:] A datatype whose values are not numeric is said to be **non-numeric**.

2.5.2 Constraining or Non-fundamental facets

[Definition:] A **constraining facet** is an optional property that can be applied to a datatype to constrain its value space.

Constraining the value space consequently constrains the lexical space. Adding constraining facets to a base type is described in Derivation by restriction (§5.1.1).

In this section we define all constraining facets that are available for use when defining derived datatypes.

2.5.2.1 length

[Definition:] **length** is the number of *units of length*, where *units of length* varies depending on the base type. The value of **length** must be a nonNegativeInteger.

For string and datatypes derived from string, **length** is measured in units of [Unicode] code points. For binary and datatypes derived from binary, **length** is measured in octets (8 bits) of binary data. For datatypes derived by list, **length** is measured in list items.

2.5.2.2 minLength

[Definition:] **minLength** is the minimum number of *units of length*, where *units of length* varies depending on the base type. The value of **minLength** must be a nonNegativeInteger.

For string and datatypes derived from string, **minLength** is measured in units of [Unicode] code points. For binary and datatypes derived from binary, **minLength** is measured in octets (8 bits) of binary data. For datatypes derived by list, **length** is measured in list items.

2.5.2.3 maxLength

[Definition:] **maxLength** is the maximum number of *units of length*, where *units of length* varies depending on the base type. The value of **maxLength** must be a nonNegativeInteger.

For string and datatypes derived from string, **maxLength** is measured in units of [Unicode] code points. For binary and datatypes derived from binary, **maxLength** is measured in octets (8 bits) of binary data. For datatypes derived by list, **length** is measured in list items.

2.5.2.4 pattern

[Definition:] **pattern** is a constraint on the value space of a datatype which is achieved by constraining the lexical space to literals which match a specific pattern. The value of **pattern** must be a regular expression.

2.5.2.5 enumeration

[Definition:] **enumeration** constrains the value space to a specified set of values.

enumeration does not impose an order relation on the value space it creates; the ordered property of the datatype involved remains that of the base type.

2.5.2.6 maxInclusive

[Definition:] **maxInclusive** is the upper bound of the value space for a datatype with the ordered property. The value is *inclusive* in the sense that the value is itself included in the value space. The value of **maxInclusive** must be of the same type as the base type.

2.5.2.7 maxExclusive

[Definition:] **maxExclusive** is the upper bound of the value space for a datatype with the ordered

property. The value is *exclusive* in the sense that the value is itself excluded from the value space. The value of **maxExclusive** must be of the same type as the base type.

2.5.2.8 minInclusive

[Definition:] **minInclusive** is the lower bound of the value space for a datatype with the ordered property. The value is *inclusive* in the sense that the value is itself included in the value space. The value of **minInclusive** must be of the same type as the base type.

2.5.2.9 minExclusive

[Definition:] **minExclusive** is the lower bound of the value space for a datatype with the ordered property. The value is *exclusive* in the sense that the value is itself excluded from the value space for the datatype. The value of **minExclusive** must be of the same type as the base type.

2.5.2.10 precision

[Definition:] **precision** is the maximum number of decimal digits in values of datatypes derived from decimal. The value of **precision** must be a positiveInteger.

2.5.2.11 scale

[Definition:] **scale** is the maximum number of decimal digits in the fractional part of values of datatypes derived from decimal. The value of **scale** must be a nonNegativeInteger .

2.5.2.12 encoding

[Definition:] **encoding** is the encoded form of the lexical space of datatypes derived from binary. The value of **encoding** must be one of {hex, base64}.

If the value of **encoding** is *hex* then each binary octet is encoded as a character tuple, consisting the two hexadecimal digits ([0-9a-fA-F]) representing the octet code. For example, "20" is the *hex* encoding for the US-ASCII space character.

If the value of **encoding** is *base64* then the entire binary stream is encoding using the Base64 Content-Transfer-Encoding defined in Section 6.8 [RFC 2045].

2.5.2.13 duration

[Definition:] **duration** is the duration of values for the datatype recurringDuration and datatypes derived from recurringDuration. The value of **duration** must be a timeDuration.

2.5.2.14 period

[Definition:] **period** is the frequency of recurrence for values for the datatype recurringDuration and datatypes derived from recurringDuration. The value of **period** must be timeDuration.

2.6 Datatype dichotomies

It is useful to categorize the datatypes defined in this specification along various dimensions, forming a set of characterization dichotomies.

2.6.1 Atomic vs. list vs. union datatypes

> **Ed. Note:** I know, now this is a trichotomy and not a dichotomy...hopefully no one will be picky enough to complain

The first distinction to be made is that between atomic, list and union datatypes.

- [Definition:] **Atomic** datatypes are those having values which are regarded by this specification as being indivisible.
- [Definition:] **List** datatypes are those having values each of which which consists of a finite-length sequence of values of an atomic datatype.
- [Definition:] **Union** datatypes are those whose value spaces and lexical spaces are the union of the value spaces and lexical spaces of two or more other datatypes.

For example, a single token which matches Nmtoken from [XML 1.0 Recommendation] could be the value of an atomic datatype (NMTOKEN); while a sequence of such tokens could be the value of a list datatype (NMTOKENS).

2.6.1.1 Atomic datatypes

atomic datatypes may be either primitive or derived. The value space of an atomic datatype is a set of "atomic" values, which for the purposes of this specification, are not further decomposable. The lexical space of an atomic datatype is a set of *literals* whose internal structure is specific to the datatype in question.

2.6.1.2 List datatypes

Several type systems (such as the one described in [ISO 11404]) treat list datatypes as special cases of the more general notions of aggregate or collection datatypes.

list datatypes are always derived. The value space of a list datatype is a set of finite-length sequences of atomic values. The lexical space of a list datatype is a set of literals whose internal structure is a whitespace separated sequence of literals of the atomic datatype of the items in the list (where whitespace matches S in [XML 1.0 Recommendation]).

Example

```
<simpleType name='sizes'>
  <list itemType='decimal'/>
</simpleType>
```

```
<cerealSizes xsi:type='sizes'> 8 10.5 12 </cerealSizes>
```

A list datatype can be derived from an atomic datatype whose lexical space allows whitespace. In such a case, regardless of the input, list items will be separated at whitespace boundaries.

Example

```
<simpleType name='listOfString'>
  <list itemType='string'/>
</simpleType>
```

```
<someElement xsi:type='listOfString'>
this is not list item 1
this is not list item 2
this is not list item 3
</someElement>
```

In the above example, the value of the *someElement* element is not a list of length 3; rather, it is a list of length 18.

When a datatype is derived from a list datatype, the following constraining facets may be used:

- length
- maxLength
- minLength
- enumeration

For each of the above facets, the *unit of length* is measured in number of list items.

2.6.1.3 Union datatypes

The value space and lexical space of a union datatype are the union of the value spaces and lexical spaces of its input types. union datatypes are always derived. Currently, there are no built-in union datatypes.

Example

A prototypical example of a union type is the maxOccurs attribute on the element element in XML Schema itself: it is a union of nonNegativeInteger and an enumeration with the single member, the string "unbounded", as shown below.

```
<attributeGroup name="occurs">
  <attribute name="minOccurs" type="nonNegativeInteger" use="default" value="1"/>
  <attribute name="maxOccurs">
    <simpleType>
      <union>
        <simpleType>
          <restriction base='nonNegativeInteger'/>
        </simpleType>
        <simpleType>
          <restriction base='string'>
            <enumeration value='unbounded'/>
          </restriction>
        </simpleType>
      </union>
    </simpleType>
  </attribute>
</attributeGroup>
```

Any number (greater than 1) of atomic or listdatatypes may participate in a union type. The order in

which the participating types are specified in the definition (that is, the order of the <simpleType> children of the <union> element) is significant. During validation, an element or attribute's value is validated against the participating types in the order in which they appear in the definition until a match is found. The evaluation order can be overridden with the use of xsi:type. See Datatype definition (§4.1) and XML representation of datatype definitions (§5) for more details.

Ed. Note: *(PVB)* Do we want to make the restriction that there has to be more than one type in a union? It was in the proposal, but I don't think it should be an error if only one appears.

Example

For example, given the definition below, the first instance of the <size> element validates correctly as an integer (§3.3.8), the second and third as string (§3.2.1).

```
<xsd:element name='size'>
  <xsd:simpleType>
    <xsd:union>
      <xsd:simpleType>
        <xsd:restriction base='integer'/>
      </xsd:simpleType>
      <xsd:simpleType>
        <xsd:restriction base='string'/>
      </xsd:simpleType>
    </xsd:union>
  </xsd:simpleType>
</xsd:element>
```

```
<size>1</size>
<size>large</size>
<size xsi:type='xsd:string'>1</size>
```

NOTE: A datatype which is atomic in this specification need not be an "atomic" datatype in any programming language used to implement this specification. Likewise, a datatype which is a list in this specification need not be a "list" datatype in any programming language used to implement this specification. Furthermore, a datatype which is a union in this specification need not be a "union" datatype in any programming language used to implement this specification.

2.6.2 Primitive vs. derived datatypes

Next, we distinquish between primitive and derived datatypes.

- [Definition:] **Primitive** datatypes are those that are not defined in terms of other datatypes; they exist *ab initio*.
- [Definition:] **Derived** datatypes are those that are defined in terms of other datatypes.

For example, in this specification, float is a well-defined mathematical concept that cannot be defined in terms of other datatypes, while a date is a special case of the more general datatype recurringDuration.

The datatypes defined by this specification fall into both the primitive and derived categories. It is felt that a judiciously chosen set of primitive datatypes will serve the widest possible audience by

providing a set of convenient datatypes that can be used as is, as well as providing a rich enough base from which the variety of datatypes needed by schema designers can be derived.

[Definition:] Every derived datatype is defined in terms of an existing datatype, referred to as the **base type**. **base type**s may be either primitive or derived.

In the example above, date is derived from the base typerecurringDuration.

> **NOTE:** A datatype which is primitive in this specification need not be a "primitive" datatype in any programming language used to implement this specification. Likewise, a datatype which is derived in this specification need not be a "derived" datatype in any programming language used to implement this specification.

2.6.3 Built-in vs. user-derived datatypes

- [Definition:] **Built-in** datatypes are those which are defined in this specification, and may be either primitive or derived;
- [Definition:] **User-derived** datatypes are those derived datatypes that are defined by individual schema designers.

Conceptually there is no difference between the built-in derived datatypes included in this specification and the user-derived datatypes which will be created by individual schema designers. The built-in derived datatypes are those which are believed to be so common that if they were not defined in this specification many schema designers would end up "reinventing" them. Furthermore, including these derived datatypes in this specification serves to demonstrate the mechanics and utility of the datatype generation facilities of this specification.

> **NOTE:** A datatype which is built-in in this specification need not be a "built-in" datatype in any programming language used to implement this specification. Likewise, a datatype which is user-derived in this specification need not be a "user-derived" datatype in any programming language used to implement this specification.

3 Built-in datatypes

3.1 Namespace considerations

The built-in datatypes defined by this specification are designed to be used with the XML Schema definition language as well as other XML specifications. To facilitate such usage the built-in datatypes in this specification have the namespace URI:

- http://www.w3.org/2000/10/XMLSchema-datatypes

This applies to both built-inprimitive and built-inderived datatypes.

Each user-derived datatype is also associated with a unique namespace. However, user-derived datatypes do not come from the namespace defined by this specification; rather, they come from the namespace of the schema in which they are defined (see XML Representation of Schemas in [XML Schema Part 1: Structures]).

As described in more detail in XML representation of datatype definitions (§5.1), each user-derived datatype must be defined in terms of another datatype in one of three ways: 1) by assigning

constraining facets which serve to *restrict* the value space of the user-derived datatype to a subset of the base type; 2) by creating a list datatype whose value space consists of finite-length sequences of values of the base type; or 3) by creating a union datatype whose value space consists of the union of the value space of two or more other datatypes.

3.2 Primitive datatypes

The primitive datatypes defined by this specification are described below. For each datatype, the value space and lexical space are defined, constraining facets which apply to the datatype are listed and any datatypes derived from this datatype are specified.

primitive datatypes can only be added by revisions to this specification.

3.2.1 string

[Definition:] The **string** datatype represents character strings in XML. The value space of **string** is the set of finite-length sequences of UCS characters ([ISO 10646] and [Unicode]). A UCS character (or just character, for short) is an atomic unit of communication; it is not further specified except to note that every UCS character has a corresponding UCS code point, which is an integer.

> **NOTE:** As noted in Order (§2.5.1.2), the fact that this specification does not specify an order-relation for string does not preclude other applications from treating strings as being ordered.

3.2.1.1 Constraining facets

string has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration

3.2.1.2 Derived datatypes

string has the following built-in derived datatypes:

- language
- NMTOKEN
- Name

3.2.2 boolean

[Definition:] **boolean** has the value space required to support the mathematical concept of binary-valued logic: {true, false}.

3.2.2.1 Lexical Representation

An instance of a datatype that is defined as boolean can have the following legal lexical values {true, false}.

3.2.2.2 Constraining facets

boolean has the following constraining facets:

- pattern

3.2.3 float

[Definition:] **float** corresponds to the IEEE single-precision 32-bit floating point type [IEEE 754-1985]. The basic value space of **float** consists of the values *m* × *2^e*, where *m* is an integer whose absolute value is less than *2^24*, and *e* is an integer between -149 and 104, inclusive. In addition to the basic value space described above, the value space of **float** also contains the following *special values*: positive and negative zero, positive negative infinity and not-a-number. The order-relation on **float** is: *x* < *y iff y* - *x* is positive.

A literal in the lexical space representing a decimal number *d* maps to the normalized value in the value space of **float** that is closest to *d*; if *d* is exactly halfway between two such values then the even value is chosen. This is the *best approximation* of *d*[Clinger, WD (1990)] [Gay, DM (1990)], which is more accurate than the mapping required by [IEEE 754-1985].

3.2.3.1 Lexical representation

float values have a lexical representation consisting of a mantissa followed, optionally, by the character "E" or "e", followed by an exponent. The exponent must be an integer. The mantissa must be a decimal number. The representations for exponent and mantissa must follow the lexical rules for integer and decimal. If the "E" or "e" and the following exponent are omitted, an exponent value of 0 is assumed.

The *special values* positive and negative zero, positive and negative infinity and not-a-number have `0`, `-0`, `INF`, `-INF` and `NaN`, respectively.

For example, `-1E4, 1267.43233E12, 12.78e-2, 12 and INF` are all legal literals for **float**.

3.2.3.2 Canonical representation

The canonical representation for **float** is defined by prohibiting certain options from the Lexical representation (§3.2.3.1). Specifically, the preceding optional "+" sign is prohibited from the mantissa. The exponent must be indicated by "E" ande number representations must be normalized such that for non-zero numbers there is a single non-zero digit to the left of the decimal point. Leading and trailing zeroes are disallowed in the mantissa and leading zeroes are disallowed in the exponent.

3.2.3.3 Constraining facets

float has the following constraining facets:

- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.4 double

[Definition:] The **double** datatype corresponds to IEEE double-precision 64-bit floating point type [IEEE 754-1985]. The basic value space of **double** consists of the values *m* × *2^e*, where *m* is an integer whose absolute value is less than *2^53*, and *e* is an integer between -1075 and 970, inclusive. In addition to the basic value space described above, the value space of **double** also contains the following *special values*: positive and negative zero, positive negative infinity and not-a-number. The order-relation on **double** is: *x* < *y iff y - x* is positive.

A literal in the lexical space representing a decimal number *d* maps to the normalized value in the value space of **double** that is closest to *d*; if *d* is exactly halfway between two such values then the even value is chosen. This is the *best approximation* of *d* ([Clinger, WD (1990)], [Gay, DM (1990)]), which is more accurate than the mapping required by [IEEE 754-1985].

3.2.4.1 Lexical representation

double values have a lexical representation consisting of a mantissa followed, optionally, by the character "E" or "e", followed by an exponent. The exponent must be an integer. The mantissa must be a decimal number. The representations for exponent and mantissa must follow the lexical rules for integer and decimal. If the "E" or "e" and the following exponent are omitted, an exponent value of 0 is assumed.

The *special values* positive and negative zero, positive and negative infinity and not-a-number have `0`, `-0`, `INF`, `-INF` and `NaN`, respectively.

For example, `-1E4, 1267.43233E12, 12.78e-2, 12 and INF` are all legal literals for **double**.

3.2.4.2 Canonical representation

The canonical representation for **double** is defined by prohibiting certain options from the Lexical representation (§3.2.4.1). Specifically, the preceding optional "+" sign is prohibited from the mantissa. The exponent must be indicated by "E" ande number representations must be normalized such that for non-zero numbers there is a single non-zero digit to the left of the decimal point. Leading and trailing zeroes are disallowed in the mantissa and leading zeroes are disallowed in the exponent.

3.2.4.3 Constraining facets

double has the following constraining facets:

- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.5 decimal

[Definition:] **decimal** represents arbitrary precision decimal numbers. The value space of **decimal** is the set of the values *i* × *10^-n*, where *i* and *n* are integers such that *n* >= *0*. The order-relation on

decimal is: *x* < *y iff y - x* is positive.

[Definition:] The value space of types derived from **decimal** with a value for precision of *p* is the set of values *i* × *10*^-*n*, where *n* and *i* are integers such that *p* >= *n* >= *0* and the number of significant decimal digits in *i* is less than or equal to *p*.

[Definition:] The value space of types derived from **decimal** with a value for scale of *s* is the set of values *i* × *10*^-*n*, where *i* and *n* are integers such that *0* <= *n* <= *s*.

> **Ed. Note:** Priority Feedback Request
> The use of arbitrary precision decimal numbers, including all datatypes derived from decimal (e.g., integer) in this design impacts the implementation of schema processors in a number of places: checking maxLength constraints on strings, for example. It may impact interchange between XML schemas and programming languages, databases, etc. Our design discussions did not reveal convincing evidence of undue burden because of arbitrary precision decimal numbers in this design, but we welcome further input from implementors.

3.2.5.1 Lexical representation

decimal has a lexical representation consisting of a finite-length sequence of decimal digits separated by a period as a decimal indicator, in accordance with the scale and precision facets, with an optional leading sign. If the sign is omitted, "+" is assumed. Leading and trailing zeroes are optional. If the fractional part is zero, the period and following zero(es) can be omitted. For example: `-1.23, 12678967.543233, +100000.00.`

3.2.5.2 Canonical representation

The canonical representation for **decimal** is defined by prohibiting certain options from the Lexical representation (§3.2.5.1). Specifically, the preceding optional "+" sign is prohibited. Leading zeroes are prohibited. Trailing zeroes to the right of the decimal point are also prohibited.

3.2.5.3 Constraining facets

decimal has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.5.4 Derived datatypes

decimal has the following built-in derived datatypes:

- integer

3.2.6 timeDuration

[Definition:] **timeDuration** represents a duration of time. The value space of **timeDuration** is a six-dimensional space where the coordinates designate the year, month, day, hour, minute, and second components defined in § 5.5.3.2 of [ISO 8601], respectively. These components are ordered in their significance by their order of appearance i.e. as year, month, day, hour, minute, and second. The order-relation on timeDuration is stated as follows: for instances x and y of timeDuration, x-y is defined as component-wise subtraction. x > y iff all the components of x-y are non-negative and at least one component is positive. x < y iff all the components of x-y are non-positive and at least one component is negative. x = y iff all components of x-y are zero.

3.2.6.1 Lexical representation

A single lexical representation, similar to the representations allowed by [ISO 8601], is allowed for **timeDuration**. This lexical representation is the [ISO 8601] extended format P*n*Y*n* M*n*DT*n*H *n*M*n*S, where *n*Y represents the number of years, *n*M the number of months, *n*D the number of days, 'T' is the date/time separator, *n*H the number of hours, *n*M the number of minutes and *n*S the number of seconds. The number of seconds can include decimal digits to arbitrary precision.

The values of the Year, Month, Day, Hour and Minutes components are not restricted but allow an arbitrary integer. Similarly, the value of the Seconds component allows an arbitrary decimal, that is they do not follow the alternative format of § 5.5.3.2.1 of [ISO 8601].

An optional preceding minus sign ('-') is allowed, to indicate a negative duration. If the sign is omitted a positive duration is indicated. See also ISO 8601 Date and Time Formats (§D).

For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes, one would write: `P1Y2M3DT10H30M`. One could also indicate a duration of minus 120 days as: `-P120D`.

Reduced precision and truncated representations of this format are allowed provided they conform to the following:

- The lowest order items may be omitted. If omitted their value is assumed to be zero.
- The lowest order item may have a decimal fraction.
- If the number of years, months, days, hours, minutes, or seconds in any expression equals zero, the number and its corresponding designator may be omitted. However, at least one number and its designator must be present.
- The designator 'T' shall be absent if all of the time items are absent. The designator 'P' must always be present.

For example, P1347Y, P1347M are P1Y2MT2H are all allowed; P0Y1347M and P0Y1347M0D are allowed. P-1347M is not allowed although -P1347M is allowed. P1Y2MT is not allowed.

3.2.6.2 Constraining facets

timeDuration has the following constraining facets:

- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.7 recurringDuration

[Definition:] **recurringDuration** represents a timeDuration that recurs with a specific timeDuration starting from a specific origin. The order-relation on **recurringDuration** is: *x* < *y iff y* - *x* is positive.

Recurring duration has two constraining facets duration and period whose values must be specified when the datatype is defined. These facets specify the length of the duration and after what duration it recurs. The lexical format used to specify these facet values is the lexical format for timeDuration. A value of 0 for the facet period means that the duration does not recur i.e. there is but a single occurrence. A value of 0 for the facet duration means that the duration is, in fact, a single instant of time.

recurringDuration is a conceptual datatype which serves as a basetype from which the other date and time datatypes are generated. It can also be used as a basetype for user-derived datatypes. A user-derived datatype can be generated from **recurringDuration** by specifying the values for duration and period. The value that appears in an instance document is the value of the origin when the recurrence begins.

Constraint: duration and period required for recurringDuration
It is an error for **recurringDuration** to be used directly in a schema. Only datatypes that are derived from **recurringDuration** by specifying a value for duration and period can be used in a schema.

3.2.7.1 Lexical representation

A single lexical representation, which is a subset of the lexical representations allowed by [ISO 8601], is allowed for **recurringDuration**. This lexical representation is the [ISO 8601] extended format CCYY-MM-DDThh:mm:ss.sss where "CC" represents the century, "YY" the year, "MM" the month and "DD" the day, preceded by an optional leading sign to indicate a negative number. If the sign is omitted, "+" is assumed. The letter "T" is the date/time separator and "hh", "mm", "ss.sss" represent hour, minute and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired. To accommodate year values greater than 9999 additional digits can be added to the left of this representation. The year 0000 is prohibited.

This representation can be immediately followed by a "Z" to indicate Coordinated Universal Time (UTC). To indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, the difference immediately follows the time and consists of a sign, + or -, followed by hh:mm. See also ISO 8601 Date and Time Formats (§D).

The derived datatype timeInstant uses the same lexical representation. Other derived datatypes date, time, timePeriod and recurringDate use truncated versions of this lexical representation.

3.2.7.2 Canonical representation

The canonical representation for **recurringDuration** is defined by prohibiting certain options from the Lexical representation (§3.2.7.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.2.7.3 Constraining facets

recurringDuration has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.7.4 Derived datatypes

recurringDuration has the following built-in derived datatypes:

- timeInstant
- time
- timePeriod
- recurringDate
- recurringDay

3.2.8 binary

[Definition:] **binary** represents arbitrary binary data. The value space of **binary** is the set of finite-length sequences of binary octets.

Constraint: encoding required for binary
It is an error for **binary** to be used directly in a schema. Only datatypes that arederived from **binary** by minimally specifying a value forencoding can be used in a schema.

3.2.8.1 Constraining facets

binary has the following constraining facets:

- encoding
- length
- minLength
- maxLength
- pattern
- enumeration

> **Ed. Note:** What does the pattern facet on binary really mean? Since pattern operates on the lexical space, one would have to give a regex for the base64 or hex that would result for a specific binary sequence that one wanted to constrain...this is not too far fetched for hex, but almost impossible for base64, isn't it?

3.2.9 uriReference

[Definition:] **uriReference** represents a Uniform Resource Identifier (URI) Reference as defined in Section 4 of [RFC 2396]. A **uriReference** may be absolute or relative, and may have an optional fragment identifier.

[Definition:] An **absolute uriReference** refers to a resource in a manner which is independent of the context in which the uriReference occurs.

[Definition:] A **relative uriReference** refers to a resource by describing the difference within a hierarchy of resources between the context in which the **relative uriReference** occurs and the absolute uriReference of the resource.

3.2.9.1 Lexical representation

The lexical space of **uriReference** is the set of strings that match the URI-reference production in Section 4 of [RFC 2396].

3.2.9.2 Constraining facets

uriReference has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration

3.2.10 ID

[Definition:] **ID** represents the ID attribute type from [XML 1.0 Recommendation]. The value space of **ID** is the set of all strings that match the NCName production in [Namespaces in XML] and have been used in an XML document. The lexical space of **ID** is the set of all strings that match the NCName production in [Namespaces in XML].

> **NOTE:** The value space of **ID** is scoped to a specific instance document.

For compatibility (see Terminology (§1.4)) **ID** should be used only on attributes.

Constraint: ID Unique
An **ID** must not appear more than once in an XML document as a value of this type; i.e., **ID** values must uniquely identify the elements which bear them.

3.2.10.1 Constraining facets

ID has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.11 IDREF

[Definition:] **IDREF** represents the IDREF attribute type from [XML 1.0 Recommendation]. The value space of **IDREF** is the set of all strings that match the NCName production in [Namespaces in XML] and have been used in an XML document as the value of an element or attribute of type ID. The lexical space of **IDREF** is the set of strings that match the NCName production in [Namespaces in XML].

> **NOTE:** The value space of **IDREF** is scoped to a specific instance document.

For compatibility (see Terminology (§1.4)) this datatype should be used only on attributes.

Constraint: IDREF
An **IDREF** must match the value of an ID in the XML document in which it occurs.

3.2.11.1 Constraining facets

IDREF has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.11.2 Derived datatypes

IDREF has the following built-in derived datatypes:

- IDREFS

3.2.12 ENTITY

[Definition:] **ENTITY** represents the ENTITY attribute type from [XML 1.0 Recommendation]. The value space of **ENTITY** is the set of all strings that match the NCName production in [Namespaces in XML] and have been declared as an unparsed entity in a document type definition. The lexical space of **ENTITY** is the set of all strings that match the NCName production in [Namespaces in XML].

> **NOTE:** The value space of **ENTITY** is scoped to a specific instance document.

Constraint: ENTITY declared
ENTITY values must match an unparsed entity name that is declared in the schema.

For compatibility (see Terminology (§1.4)) **ENTITY** should be used only on attributes.

3.2.12.1 Constraining facets

ENTITY has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.12.2 Derived datatypes

ENTITY has the following built-in derived datatypes:

- ENTITIES

3.2.13 NOTATION

[Definition:] **NOTATION** represents the NOTATION attribute type from [XML 1.0 Recommendation]. The value space of **NOTATION** is the set of all notations declared in a schema. The lexical space of **NOTATION** is the set of all strings that match the NCName production in [Namespaces in XML].

> **NOTE:** The value space of **NOTATION** is scoped to a specific instance document.

Constraint: NOTATION declared
NOTATION values must match a notation name that is declared in the schema.

For compatibility (see Terminology (§1.4)) **NOTATION** should be used only on attributes.

3.2.13.1 Constraining facets

NOTATION has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.14 QName

[Definition:] **QName** represents XML qualified names. The value space of **QName** is the set of tuples {namespace name, local part}, where namespace name is a uriReference and local part is an NCName. The lexical space of **QName** is the set of strings that match the QName production of

[Namespaces in XML].

3.2.14.1 Constraining facets

QName has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3 Derived datatypes

This section gives conceptual definitions for all built-in derived datatypes defined by this specification. The XML Representation used to define derived datatypes (whether built-in or user-derived) is given in section XML representation of datatype definitions (§5.1) and the complete definitions of the built-in derived datatypes are provided in Appendix Schema for Datatype Definitions (normative) (§A).

3.3.1 language

[Definition:] **language** represents natural language identifiers as defined by [RFC 1766]. The value space of **language** is the set of all strings that match the LanguageID production in [XML 1.0 Recommendation]. The lexical space of **language** is the set of all strings that match the LanguageID production in [XML 1.0 Recommendation]. The base type of **language** is string.

3.3.1.1 Constraining facets

language has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration

3.3.2 IDREFS

[Definition:] **IDREFS** represents the IDREFS attribute type from [XML 1.0 Recommendation]. The value space of **IDREFS** is the set of finite-length sequences of IDREFs that have been used in an XML document. The lexical space of **IDREFS** is the set of whitespace separated tokens, each of which is in the lexical space of IDREF. The base type of **IDREFS** is IDREF.

> **NOTE:** The value space of **IDREFS** is scoped to a specific instance document.

For compatibility (see Terminology (§1.4)) **IDREFS** should be used only on attributes.

3.3.2.1 Constraining facets

IDREFS has the following constraining facets:

- length
- minLength
- maxLength
- enumeration

3.3.3 ENTITIES

[Definition:] **ENTITIES** represents the ENTITIES attribute type from [XML 1.0 Recommendation]. The value space of **ENTITIES** is the set of finite-length sequences of ENTITYs that have been declared as unparsed entities in a document type definition. The lexical space of **ENTITIES** is the set of whitespace separated tokens, each of which is in the lexical space of NMTOKEN. The base type of **ENTITIES** is ENTITY.

> **NOTE:** The value space of **ENTITIES** is scoped to a specific instance document.

For compatibility (see Terminology (§1.4)) **ENTITIES** should be used only on attributes.

3.3.3.1 Constraining facets

ENTITIES has the following constraining facets:

- length
- minLength
- maxLength
- enumeration

3.3.4 NMTOKEN

[Definition:] **NMTOKEN** represents the NMTOKEN attribute type from [XML 1.0 Recommendation]. The value space of **NMTOKEN** is the set of tokens that match the Nmtoken production in [XML 1.0 Recommendation]. The lexical space of **NMTOKEN** is the set of strings that match the Nmtoken production in [XML 1.0 Recommendation]. The base type of **NMTOKEN** is string.

For compatibility (see Terminology (§1.4)) **NMTOKEN** should be used only on attributes.

3.3.4.1 Constraining facets

NMTOKEN has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration

3.3.4.2 Derived datatypes

NMTOKEN has the following built-in derived datatypes:

- NMTOKENS

3.3.5 NMTOKENS

[Definition:] **NMTOKENS** represents the NMTOKENS attribute type from [XML 1.0 Recommendation]. The value space of **NMTOKENS** is the set of finite-length sequences of NMTOKENs. The lexical space of **NMTOKENS** is the set of whitespace separated tokens, each of which is in the lexical space of NMTOKEN. The base type of **NMTOKENS** is NMTOKEN.

For compatibility (see Terminology (§1.4)) **NMTOKENS** should be used only on attributes.

3.3.5.1 Constraining facets

NMTOKENS has the following constraining facets:

- length
- minLength
- maxLength
- enumeration

3.3.6 Name

[Definition:] **Name** represents XML Names. The value space of **Name** is the set of all strings which match the Name production of [XML 1.0 Recommendation]. The lexical space of **Name** is the set of all strings which match the Name production of [XML 1.0 Recommendation]. The base type of **Name** is string.

3.3.6.1 Constraining facets

Name has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration

3.3.6.2 Derived datatypes

Name has the following built-in derived datatypes:

- NCName

3.3.7 NCName

[Definition:] **NCName** represents XML "non-colonized" Names. The value space of **NCName** is the set of all strings which match the NCName production of [Namespaces in XML]. The lexical space of **NCName** is the set of all strings which match the NCName production of [Namespaces in XML]. The base type of **NCName** is Name.

3.3.7.1 Constraining facets

NCName has the following constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration

3.3.8 integer

[Definition:] **integer** is derived from decimal by fixing the value of scale to be 0. This results in the standard mathematical concept of the integer numbers. The value space of **integer** is the infinite set {...,-2,-1,0,1,2,...}. The base type of **integer** is decimal.

3.3.8.1 Lexical representation

integer has a lexical representation consisting of a finite-length sequence of decimal digits with an optional leading sign. If the sign is omitted, "+" is assumed. For example: -1, 0, 12678967543233, +100000.

3.3.8.2 Canonical representation

The canonical representation for **integer** is defined by prohibiting certain options from the Lexical representation (§3.3.8.1). Specifically, the preceding optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.8.3 Constraining facets

integer has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.8.4 Derived datatypes

integer has the following built-in derived datatypes:

- nonPositiveInteger
- long
- nonNegativeInteger

3.3.9 nonPositiveInteger

[Definition:] **nonPositiveInteger** is derived from integer by setting the value of maxInclusive to be 0. This results in the standard mathematical concept of the non-positive integers. The value space of

nonPositiveInteger is the infinite set {...,-2,-1,0}. The base type of **nonPositiveInteger** is integer.

3.3.9.1 Lexical representation

nonPositiveInteger has a lexical representation consisting of a negative sign ("-") followed by a finite-length sequence of decimal digits. If the sequence of digits consists of all zeros then the sign is optional. For example: -1, 0, -12678967543233, -100000.

3.3.9.2 Canonical representation

The canonical representation for **nonPositiveInteger** is defined by prohibiting certain options from the Lexical representation (§3.3.9.1). Specifically, the negative sign ("-") is required with the token "0" and leading zeroes are prohibited.

3.3.9.3 Constraining facets

nonPositiveInteger has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.9.4 Derived datatypes

nonPositiveInteger has the following built-in derived datatypes:

- negativeInteger

3.3.10 negativeInteger

[Definition:] **negativeInteger** is derived from nonPositiveInteger by setting the value of maxInclusive to be -1. This results in the standard mathematical concept of the negative integers. The value space of **negativeInteger** is the infinite set {...,-2,-1}. The base type of **negativeInteger** is nonPositiveInteger.

3.3.10.1 Lexical representation

negativeInteger has a lexical representation consisting of a negative sign ("-") followed by a finite-length sequence of decimal digits. For example: -1, -12678967543233, -100000.

3.3.10.2 Canonical representation

The canonical representation for **negativeInteger** is defined by prohibiting certain options from the Lexical representation (§3.3.10.1). Specifically, leading zeroes are prohibited.

3.3.10.3 Constraining facets

negativeInteger has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.11 long

[Definition:] **long** is derived from integer by setting the value of maxInclusive to be 9223372036854775807 and minInclusive to be -9223372036854775808. The base type of **long** is integer.

3.3.11.1 Lexical representation

long has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits. If the sign is omitted, "+" is assumed. For example: -1, 0, 12678967543233, +100000.

3.3.11.2 Canonical representation

The canonical representation for **long** is defined by prohibiting certain options from the Lexical representation (§3.3.11.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.11.3 Constraining facets

long has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.11.4 Derived datatypes

long has the following built-in derived datatypes:

- int

3.3.12 int

[Definition:] **int** is derived from long by setting the value of maxInclusive to be 2147483647 and minInclusive to be -2147483648. The base type of **int** is long.

3.3.12.1 Lexical representation

int has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits. If the sign is omitted, "+" is assumed. For example: -1, 0, 126789675, +100000.

3.3.12.2 Canonical representation

The canonical representation for **int** is defined by prohibiting certain options from the Lexical representation (§3.3.12.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.12.3 Constraining facets

int has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.12.4 Derived datatypes

int has the following built-in derived datatypes:

- short

3.3.13 short

[Definition:] **short** is derived from int by setting the value of maxInclusive to be 32767 and minInclusive to be -32768. The base type of **short** is int.

3.3.13.1 Lexical representation

short has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits. If the sign is omitted, "+" is assumed. For example: -1, 0, 12678, +10000.

3.3.13.2 Canonical representation

The canonical representation for **short** is defined by prohibiting certain options from the Lexical representation (§3.3.13.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.13.3 Constraining facets

short has the following constraining facets:

- precision
- scale
- pattern
- enumeration

- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.13.4 Derived datatypes

short has the following built-in derived datatypes:

- byte

3.3.14 byte

[Definition:] **byte** is derived from short by setting the value of maxInclusive to be 127 and minInclusive to be -128. The base type of **byte** is short.

3.3.14.1 Lexical representation

byte has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits. If the sign is omitted, "+" is assumed. For example: -1, 0, 126, +100.

3.3.14.2 Canonical representation

The canonical representation for **byte** is defined by prohibiting certain options from the Lexical representation (§3.3.14.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.14.3 Constraining facets

byte has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.15 nonNegativeInteger

[Definition:] **nonNegativeInteger** is derived from integer by setting the value of minInclusive to be 0. This results in the standard mathematical concept of the non-negative integers. The value space of **nonNegativeInteger** is the infinite set {0,1,2,...}. The base type of **nonNegativeInteger** is integer.

3.3.15.1 Lexical representation

nonNegativeInteger has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits. If the sign is omitted, "+" is assumed. For example: 1, 0, 12678967543233, +100000.

3.3.15.2 Canonical representation

The canonical representation for **nonNegativeInteger** is defined by prohibiting certain options from the Lexical representation (§3.3.15.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.15.3 Constraining facets

nonNegativeInteger has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.15.4 Derived datatypes

nonNegativeInteger has the following built-in derived datatypes:

- unsignedLong
- positiveInteger

3.3.16 unsignedLong

[Definition:] **unsignedLong** is derived from nonNegativeInteger by setting the value of maxInclusive to be 18446744073709551615. The base type of **unsignedLong** is nonNegativeInteger.

3.3.16.1 Lexical representation

unsignedLong has a lexical representation consisting of a finite-length sequence of decimal digits. For example: 0, 12678967543233, 100000.

3.3.16.2 Canonical representation

The canonical representation for **unsignedLong** is defined by prohibiting certain options from the Lexical representation (§3.3.16.1). Specifically, leading zeroes are prohibited.

3.3.16.3 Constraining facets

unsignedLong has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.16.4 Derived datatypes

unsignedLong has the following built-in derived datatypes:

- unsignedInt

3.3.17 unsignedInt

[Definition:] **unsignedInt** is derived from unsignedLong by setting the value of maxInclusive to be 4294967295. The base type of **unsignedInt** is unsignedLong.

3.3.17.1 Lexical representation

unsignedInt has a lexical representation consisting of a finite-length sequence of decimal digits. For example: 0, 1267896754, 100000.

3.3.17.2 Canonical representation

The canonical representation for **unsignedInt** is defined by prohibiting certain options from the Lexical representation (§3.3.17.1). Specifically, leading zeroes are prohibited.

3.3.17.3 Constraining facets

unsignedInt has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.17.4 Derived datatypes

unsignedInt has the following built-in derived datatypes:

- unsignedShort

3.3.18 unsignedShort

[Definition:] **unsignedShort** is derived from unsignedInt by setting the value of maxInclusive to be 65535. The base type of **unsignedShort** is unsignedInt.

3.3.18.1 Lexical representation

unsignedShort has a lexical representation consisting of a finite-length sequence of decimal digits. For example: 0, 12678, 10000.

3.3.18.2 Canonical representation

The canonical representation for **unsignedShort** is defined by prohibiting certain options from the

Lexical representation (§3.3.18.1). Specifically, the leading zeroes are prohibited.

3.3.18.3 Constraining facets

unsignedShort has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.18.4 Derived datatypes

unsignedShort has the following built-in derived datatypes:

- unsignedByte

3.3.19 unsignedByte

[Definition:] **unsignedByte** is derived from unsignedShort by setting the value of maxInclusive to be 255. The base type of **unsignedByte** is unsignedShort.

3.3.19.1 Lexical representation

unsignedByte has a lexical representation consisting of a finite-length sequence of decimal digits. For example: 0, 126, 100.

3.3.19.2 Canonical representation

The canonical representation for **unsignedByte** is defined by prohibiting certain options from the Lexical representation (§3.3.19.1). Specifically, leading zeroes are prohibited.

3.3.19.3 Constraining facets

unsignedByte has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.20 positiveInteger

[Definition:] **positiveInteger** is derived from nonNegativeInteger by setting the value of minInclusive to be 1. This results in the standard mathematical concept of the positive integer numbers. The value

space of **positiveInteger** is the infinite set {1,2,...}. The base type of **positiveInteger** is nonNegativeInteger.

3.3.20.1 Lexical representation

positiveInteger has a lexical representation consisting of an optional positive sign ("+") followed by a finite-length sequence of decimal digits. For example: 1, 12678967543233, +100000.

3.3.20.2 Canonical representation

The canonical representation for **positiveInteger** is defined by prohibiting certain options from the Lexical representation (§3.3.20.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.20.3 Constraining facets

positiveInteger has the following constraining facets:

- precision
- scale
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.21 timeInstant

[Definition:] **timeInstant** represents a specific instant of time. The value space of **timeInstant** is the space of *Combinations of date and time of day* values as defined in § 5.4 of [ISO 8601]. The base type of **timeInstant** is recurringDuration. **timeInstant** is generated from recurringDuration by fixing the value of the duration facet equal to "P0Y" and the value of the period facet equal to "P0Y" (no recurrence).

3.3.21.1 Lexical representation

A single lexical representation, which is a subset of the lexical representations allowed by [ISO 8601], and is the same lexical representation as its basetype recurringDuration is allowed for **timeInstant**.

For example, to indicate 1:20 pm on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write: `1999-05-31T13:20:00-05:00`.

3.3.21.2 Canonical representation

The canonical representation for **timeInstant** is defined by prohibiting certain options from the Lexical representation (§3.3.21.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.21.3 Constraining facets

timeInstant has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.22 time

[Definition:] **time** represents an instant of time that recurs every day. The value space of **time** is the space of *time of day* values as defined in § 5.3 of [ISO 8601]. Specifically, it is a set of zero-duration daily instances e.g. lexical 12:30:24 to represent T12:30:24 on any day. The base type of **time** is recurringDuration. **time** is generated from recurringDuration by fixing the value of the duration facet equal to "P0Y" and the value of the period facet equal to "P1D".

3.3.22.1 Lexical representation

The lexical representation for **time** is the left truncated lexical representation for timeInstant: hh:mm:ss.sss with optional following time zone indicator. For example, to indicate 1:20 pm for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write: 13:20:00-05:00. See also ISO 8601 Date and Time Formats (§D).

3.3.22.2 Canonical representation

The canonical representation for **time** is defined by prohibiting certain options from the Lexical representation (§3.3.22.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.22.3 Constraining facets

time has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.23 timePeriod

[Definition:] **timePeriod** represents a specific period of time with a given start and end. The value space of **timePeriod** is the value space of *Periods of time* as defined in § 5.5 of [ISO 8601]. The base type of **timePeriod** is recurringDuration. **timePeriod** is generated from recurringDuration by fixing the value of the period facet equal to "P0Y" (no recurrence).

The value in the instance specifies the start of the **time period** while the value of duration facet,

specified when subtypes are defined, gives the duration of the **time period**.

3.3.23.1 Lexical representation

The lexical representation for **timePeriod** is the same as that of its basetype, recurringDuration.

3.3.23.2 Canonical representation

The canonical representation for **timePeriod** is defined by prohibiting certain options from the Lexical representation (§3.3.23.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.23.3 Constraining facets

timePeriod has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

Constraint: period facet value required for timePeriod
It is an error for **timePeriod** to be used directly in a schema. Only datatypes that arederived from **timePeriod** by specifying a value forduration can be used in a schema.

3.3.23.4 Derived datatypes

timePeriod has the following built-in derived datatypes:

- date
- month
- year
- century

3.3.24 date

[Definition:] **date** represents a timePeriod that starts at midnight of a specified day and lasts until midnight the following day. The value space of **date** is the set of Gregorian calendar dates as defined in § 5.2.1 of [ISO 8601]. Specifically, it is a set of one-day long, non-periodic instances e.g. lexical 1999-10-26 to represent the whole day of 1999-10-26, independent of how many hours this day has. The base type of **date** is timePeriod. **date** is generated from timePeriod by fixing the value of the duration facet equal to "P1D".

3.3.24.1 Lexical representation

The lexical representation for **date** is the reduced (right truncated) lexical representation for timePeriod: CCYY-MM-DD. No left truncation is allowed. An optional following time zone qualifier is allowed as for timePeriod. To accommodate year values outside the range from 0001 to 9999,

additional digits can be added to the left of this representation and a preceding "-" is allowed.

For example, to indicate May the 31st, 1999, one would write: 1999-05-31. See also ISO 8601 Date and Time Formats (§D).

3.3.24.2 Canonical representation

The canonical representation for **date** is defined by prohibiting certain options from the Lexical representation (§3.3.24.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.24.3 Constraining facets

date has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.25 month

[Definition:] **month** represents a timePeriod that starts at midnight on the first day of the month and lasts until the midnight that ends the last day of the month. The value space of **month** is the set of Gregorian calendar months as defined in § 5.2.1 of [ISO 8601]. Specifically, it is a set of one-month long, non-periodic instances e.g. 1999-10 to represent the whole month of 1999-10, independent of how many days this month has. The base type of **month** is timePeriod. **month** is generated from timePeriod by fixing the value of the duration facet equal to "P1M".

3.3.25.1 Lexical representation

The lexical representation for **month** is the reduced (right truncated) lexical representation for timePeriod: CCYY-MM. No left truncation is allowed. An optional following time zone qualifier is allowed as for timePeriod. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" is allowed.

For example, to indicate the month of May 1999, one would write: 1999-05. See also ISO 8601 Date and Time Formats (§D).

3.3.25.2 Canonical representation

The canonical representation for **month** is defined by prohibiting certain options from the Lexical representation (§3.3.25.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.25.3 Constraining facets

month has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.26 year

[Definition:] **year** represents a timePeriod that starts at the midnight that starts the first day of the year and ends at the midnight that ends the last day of the year. The value space of **year** is the set of Gregorian calendar years as defined in § 5.2.1 of [ISO 8601]. Specifically, it is a set of one-year long, non-periodic instances e.g. lexical 1999 to represent the whole year 1999, independent of how many months and days this year has. The base type of **year** is timePeriod. **year** is generated from timePeriod by fixing the value of the duration facet equal to "P1Y".

3.3.26.1 Lexical representation

The lexical representation for **year** is the reduced (right truncated) lexical representation for timePeriod: CCYY. No left truncation is allowed. An optional following time zone qualifier is allowed as for timePeriod. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" is allowed.

For example, to indicate 1999, one would write: 1999. See also ISO 8601 Date and Time Formats (§D).

3.3.26.2 Canonical representation

The canonical representation for **year** is defined by prohibiting certain options from the Lexical representation (§3.3.26.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.26.3 Constraining facets

year has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.27 century

[Definition:] **century** represents a timePeriod that starts at the midnight that starts the first day of the century and ends at the midnight that ends that last day of the century. The value space of **century** is the set of Gregorian calendar centuries as defined in § 5.2.1 of [ISO 8601]. Specifically, it is a set

of one-century long, non-periodic instances e.g. lexical 20 to represent the whole of the 19th century. The base type of **century** is timePeriod. **century** is generated from timePeriod by fixing the value of the duration facet equal to "P100Y".

3.3.27.1 Lexical representation

The lexical representation for **century** is the reduced (right truncated) lexical representation for timePeriod: CC. No left truncation is allowed. An optional following time zone qualifier is allowed as for timePeriod. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" is allowed.

For example, to indicate the 20th century, one would write: 19. See also ISO 8601 Date and Time Formats (§D).

3.3.27.2 Canonical representation

The canonical representation for **century** is defined by prohibiting certain options from the Lexical representation (§3.3.27.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.27.3 Constraining facets

century has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.28 recurringDate

[Definition:] **recurringDate** is a date that recurs, specifically a day of the year such as the third of May. Arbitrary recurring dates are not supported by this datatype. The value space of **recurringDate** is the set of *calendar dates*, as defined in § 3 of [ISO 8601]. Specidically, it is a set of one-day long, annually periodic instances. The base type of **recurringDate** is recurringDuration. **recurringDate** is generated from recurringDuration by fixing the value of the duration facet equal to "P1D" and the value of the period facet to "P1Y" (one year).

3.3.28.1 Lexical representation

The lexical representation for **recurringDate** is the left truncated lexical representation for date: --MM-DD. No other formats are allowed. See also ISO 8601 Date and Time Formats (§D).

3.3.28.2 Canonical representation

The canonical representation for **recurringDate** is defined by prohibiting certain options from the Lexical representation (§3.3.28.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.28.3 Constraining facets

recurringDate has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.29 recurringDay

[Definition:] **recurringDay** is a day that recurs, specifically a day of the month such as the 5th of the month. Arbitrary recurring days are not supported by this datatype. The value space of **recurringDay** is the space of a set of *calendar dates* as defined in § 3 of [ISO 8601]. Specifically, it is a set of one-day long, monthly periodic instances. The base type of **recurringDay** is recurringDuration. **recurringDay** is generated from recurringDuration by fixing the value of the duration facet equal to "P1D" and the value of the period facet to "P1M".

3.3.29.1 Lexical representation

The lexical representation for **recurringDay** is the left truncated lexical representation for date: ---DD. No other formats are allowed. See also ISO 8601 Date and Time Formats (§D).

3.3.29.2 Canonical representation

The canonical representation for **recurringDay** is defined by prohibiting certain options from the Lexical representation (§3.3.29.1). Specifically, the preceding optional "+" sign is prohibited and the time zone must be Coordinated Universal Time (UTC) and be indicated by a "Z".

3.3.29.3 Constraining facets

recurringDay has the following constraining facets:

- duration
- period
- pattern
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

4 Datatype components

The following sections provide full details on the properties and significance of each kind of schema component involved in datatype definitions. For each property, the kinds of values it may have is specified. Any property not identified as optional is required to be present; optional properties which are absent are taken to have the null value.

Throughout the following sections, the [value] of an attribute information item means a string composed of, in order, the [character code] of each character information item in the [children] of that attribute information item.

For more information on the notion of datatype (schema) components, see Schema Component Details

> **NOTE:** Readers whose primary interest is in the XML representation of datatype definitions may wish to skip this section on the first reading, concentrating instead on XML representation of datatype definitions (§5) .

4.1 Datatype definition

Datatype definitions provide for:

- Establishing the value space of a datatype, through the combined set of constraining facets specified in the definition;
- Attaching a unique name (actually a QName) to the value space

The datatype definition schema component has the following properties:

Schema Component: Datatype Definition
[name] Optional. An NCName as defined by [Namespaces in XML]. **[target namespace]** Either null or a namespace URI, as defined in [Namespaces in XML]. **[variety]** One of {atomic, list, union}. **[base type definition]** If {variety} is atomic or list then a datatype definition, else if {variety} is union then a list of datatype definitions. **[facets]** A possibly empty set of Constraining facets (§4.2). **[fundamental facets]** A set of Fundamental facets (§2.5.1) **[annotation]** Optional. An annotation.

Datatypes are identified by their {name} and {target namespace}. Except for anonymous datatypes (those with no {name}), datatype definitions must be uniquely identified within an schema.

If {variety} is atomic then the value space of the datatype defined will be a subset of the value space of {base type definition}. If {variety} is list then the value space of the datatype defined will be a finite-length sequence of values from the value space of {base type definition}. If {variety} is union then the value space of the datatype defined will be the union of the value spaces of each datatype in {base type definition}.

If {variety} is atomic then {base type definition} must itself be atomic. If {variety} is list then {base type

definition} must be either atomic or union. If {variety} is union then {base type definition} must be a list of atomic, list or union types.

> **Ed. Note:** *(PVB)* List of union makes sense, allowing different items in the list to be of the different types in the union. There is no way to override the lexical resolution strategy, i.e., you can't use xsi:type to say that item 3 of the list is really a string when the resolution strategy would say it is an int. This is OK, but it just dawned on my and I wanted to call it out to make sure I'm correct.

The value of {facets} consists of the set of facets specified directly in the datatype definition unioned with the possibly empty set of {facets} of {base type definition}.

The value of {fundamental facets} consists of the set of fundamental facets and their values. If {variety} is atomic then {fundamental facets} are inherited from {base type definition} with the possibility that bounded could become *true* if one of facets providing bounds is given a value (e.g., maxInclusive, maxExclusive, minInclusive or minExclusive). If {variety} is list then numeric is *false* and the other fundamental facets are valued the same as when {variety} is atomic. If {variety} is union then ordered, bounded and numeric are *false* and cardinality is *countably infinite* if any datatype in {base type definition} is *countably infinite* and *finite* otherwise.

> **Ed. Note:** *(PVB)* The above is slightly wrong for unions, but I don't have time to think thru all of the cases. For instance, if all the types in the union are bounded and the VS of the union is the union of the participating types, then shouldn't the union be bounded?

Constraint on Schemas: applicable facets

The constraining facets which are allowed the be members of{facets} are dependent on {base type definition} as specified in the following table:

	{base type definition}	**applicable {facets}**
If {variety} is list, then		
	[all datatypes]	length, minLength, maxLength, enumeration
If {variety} is union, then		
	[all datatypes]	pattern, enumeration
else if {variety} is atomic, then		
primitive	string	length, minLength, maxLength, pattern, enumeration
	boolean	pattern
	float	pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	double	pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	decimal	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	timeDuration	pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	recurringDuration	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	binary	encoding, length, minLength, maxLength, pattern, enumeration

	uriReference	length, minLength, maxLength, pattern, enumeration
	ID	length, minLength, maxLength, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	IDREF	length, minLength, maxLength, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	ENTITY	length, minLength, maxLength, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	NOTATION	length, minLength, maxLength, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	QName	length, minLength, maxLength, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
derived	language	length, minLength, maxLength, pattern, enumeration
	IDREFS	length, minLength, maxLength, enumeration
	ENTITIES	length, minLength, maxLength, enumeration
	NMTOKEN	length, minLength, maxLength, pattern, enumeration
	NMTOKENS	length, minLength, maxLength, enumeration
	Name	length, minLength, maxLength, pattern, enumeration
	NCName	length, minLength, maxLength, pattern, enumeration
	integer	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	nonPositiveInteger	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	negativeInteger	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	long	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	int	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	short	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	byte	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	nonNegativeInteger	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	unsignedLong	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	unsignedInt	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	unsignedShort	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	unsignedByte	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive

	positiveInteger	precision, scale, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	timeInstant	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	time	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	timePeriod	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	date	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	month	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	year	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	century	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	recurringDate	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
	recurringDay	duration, period, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive

Constraint on Schemas: list of atomic
If {variety} is list, then {variety} of {base type definition} must be atomic.

Validation Contribution: datatype valid
A sequence of character information items appearing in an instance document is schema-valid with respect to a datatype definition if: 1. It is a member of the lexical space of the {base type definition}; 2. The member of the value space of the {base type definition} denoted by the sequence is facet valid (§4.2) with respect to each member of {facets}.

4.2 Constraining facets

This section provides the details of each constraining facet component.

constraining facets provide for:

- Constraining the value space of a datatype by specifying optional properties which serve to semantically characterize the values in the value space.

Validation Contribution: facet valid
A member of a value space is schema-valid with respect to a constraining facet component if: 1. the member is schema-valid with respect to the particular constraining facet as specified below.

4.2.1 length

length provides for:

- Constraining a value space to values with a specific number of *units of length*, where *units of length* varies depending on {base type definition}.

Schema Component: length
[value] A nonNegativeInteger. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Constraint on Schemas: length and minLength
It is an error for both length and minLength to be members of {facets}.

Validation Contribution: length valid
A member of a value space is schema-valid with respect to length if: 1. {base type definition} is string, then the length of the value, as measured in [Unicode] code points, is equal to {value}; 2. {base type definition} is binary, then the length of the value, as measured in octets of the binary data, is equal to {value}; 3. {variety} is list, then the length of the value, as measured in list items, is equal to {value}

4.2.2 minLength

minLength provides for:

- Constraining a value space to values with at least a specific number of *units of length*, where *units of length* varies depending on {base type definition}.

Schema Component: minLength
[value] A nonNegativeInteger. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Constraint on Schemas: minLength <= maxLength
If both minLength and maxLength are members of {facets}, then the {value} of minLength must be less than or equal to the {value} of maxLength.

Validation Contribution: minLength valid
A member of a value space is schema-valid with respect to minLength if: 1. {base type definition} is string, then the length of the value, as measured in [Unicode] code points, is greater than or equal to {value}; 2. {base type definition} is binary, then the length of the value, as measured in octets of binary data, is greater than or equal to {value}; 3. {variety} is list, then the length of the value, as measured in list items, is greater than or equal to {value}

4.2.3 maxLength

maxLength provides for:

- Constraining a value space to values with at most a specific number of *units of length*, where *units of length* varies depending on {base type definition}.

Schema Component: maxLength
[value] A nonNegativeInteger. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Constraint on Schemas: length and maxLength
It is an error for both length and maxLength to be specified for the same datatype.

Validation Contribution: maxLength valid
A member of a value space is schema-valid with respect to maxLength if: 1. {base type definition} is string, then the length of the value, as measured in [Unicode] code points, is less than or equal to {value}; 2. {base type definition} is binary, then the length of the value, as measured in octets of binary data, is less than or equal to {value}; 3. {variety} is list, then the length of the value, as measured in list items, is less than or equal to {value}

4.2.4 pattern

pattern provides for:

- Constraining a value space to values that are denoted by literals which match a specific

regular expression.

Schema Component: pattern
[value] A regular expression. **[annotation]** Optional. An annotation.

Ed. Note: *(PVB)* Fixing the pattern component makes sense (and would result in no further restriction of the value space using any other facet), but the concept of "fixing" one occurance of pattern facet doesn't seem to make sense. So, I'm not sure how this should be handled at the XML representation level.

Ed. Note: *(AM)* I have removed the "fixed" property from enumeration and pattern pending resolution on how to handle these two cases.

Validation Contribution: pattern valid
A member of a value space is schema-valid with respect to pattern if: 1. the value is among the set of values denoted by literals which are among the set of strings denoted by the regular expression specified in {value}.

4.2.5 enumeration

enumeration provides for:

- Constraining a value space to a specified set of values.

Schema Component: enumeration
[value] A set of values from the value space of the {base type definition}. **[annotation]** Optional. An annotation.

Ed. Note: *(PVB)* Fixing the enumeration component makes sense (and would result in no further restriction of the value space using any other facet), but the concept of "fixing" one occurance of enumeration facet doesn't seem to make sense. So, I'm not sure how this should be handled at the XML representation level.

Ed. Note: *(AM)* I have removed the "fixed" property from enumeration and pattern pending resolution on how to handle these two cases.

Validation Contribution: enumeration valid
A member of a value space is schema-valid with respect to enumeration if: 1. the value is one of the values specified in {value}.

4.2.6 maxInclusive

maxInclusive provides for:

- Constraining a value space to values with a specific *inclusive*upper bound.

Schema Component: maxInclusive
[value] A value from the value space of the {base type definition}. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Constraint on Schemas: minInclusive <= maxInclusive
It is an error for the value specified for minInclusive to be greater than the value specified for maxInclusive for the same datatype.

Validation Contribution: maxInclusive valid
A member of a value space is schema-valid with respect to maxInclusive if: 1. the numeric property in {fundamental facets} is *true*, then the value is numerically less than or equal to {value}; 2. the numeric property of {base type definition} is *false* (i.e., {base type definition} is one of the date and time related datatypes), then the value must be chronologically less than or equal to{value};

4.2.7 maxExclusive

maxExclusive provides for:

- Constraining a value space to values with a specific *exclusive*upper bound.

Schema Component: maxExclusive
[value] A value from the value space of the {base type definition}. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Constraint on Schemas: maxInclusive and maxExclusive
It is an error for both maxInclusive and maxExclusive to be specified for the same datatype.

Constraint on Schemas: minExclusive <= maxExclusive
It is an error for the value specified for minExclusive to be greater than the value specified for maxExclusive for the same datatype.

Validation Contribution: maxExclusive valid
A member of a value space is schema-valid with respect to maxExclusive if: 1. the numeric property of {base type definition} is *true*, then the value must be numerically less than {value}; 2. the numeric property of {base type definition} is *false* (i.e., {base type definition} is one of the date and time related datatypes), then the value must be chronologically less than {value};

4.2.8 minExclusive

minExclusive provides for:

- Constraining a value space to values with a specific *exclusive* lower bound.

Schema Component: minExclusive
[value] A value from the value space of the {base type definition}. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Constraint on Schemas: minInclusive and minExclusive
It is an error for both minInclusive and minExclusive to be specified for the same datatype.

Validation Contribution: minExclusive valid
A member of a value space is schema-valid with respect to minExclusive if: 1. the numeric property of {base type definition} is *true*, then the value must be numerically greater than {value}; 2. the numeric property of {base type definition} is *false* (i.e., {base type definition} is one of the date and time related datatypes), then the value must be chronologically greater than {value};

4.2.9 minInclusive

minInclusive provides for:

- Constraining a value space to values with a specific *inclusive* lower bound.

Schema Component: minInclusive
[value] A value from the value space of the {base type definition}. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Validation Contribution: minInclusive valid
A member of a value space is schema-valid with respect to minInclusive if: 1. the numeric property of {base type definition} is *true*, then the value must be numerically greater than or equal to {value}; 2. the numeric property of {base type definition} is *false* (i.e., {base type definition} is one of the date and time related datatypes), then the value must be chronologically greater than or equal to {value};

4.2.10 precision

precision provides for:

- Constraining a value space to values with a specific maximum number of decimal digits.

Schema Component: precision
[value] A positiveInteger. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Validation Contribution: precision valid
A member of a value space is schema-valid with respect to precision if: 1. the number of decimal digits in the value is less than or equal to {value};

4.2.11 scale

scale provides for:

- Constraining a value space to values with a specific maximum number of decimal digits in the fractional part.

Schema Component: scale
[value] A nonNegativeInteger. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

Constraint on Schemas: scale less than or equal to precision
It is an error for scale to be greater than precision.

Validation Contribution: scale valid
A member of a value space is schema-valid with respect to precision if: 1. the number of decimal digits in the fractional part of the value is less than or equal to {value};

4.2.12 encoding

encoding provides for:

- Constraining the lexical space of datatypes derived from binary to a specified form.

Schema Component: encoding
[value] One of {*hex*, *base64*}. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

NOTE: There are no Validity Contributions associated with encoding.

4.2.13 duration

duration provides for:

- Constraining a value space to values of a specific duration of time.

Schema Component: duration
[value] A timeDuration. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

NOTE: There are no Validity Contributions associated duration.

4.2.14 period

period provides for:

- Constraining a value space to values of a specific frequency of recurrence.

Schema Component: period
[value] A timeDuration. **[fixed]** A boolean. **[annotation]** Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} may not specify a value for other than {value}.

NOTE: There are no Validity Contributions associated period.

5 XML representation of datatype definitions

The sections below define correspondences between element information items and datatype definition components. All the element information items in the XML representation of a datatype definition are in the XML Schema namespace, that is their [namespace URI] is `http://www.w3.org/2000/10/XMLSchema`.

Throughout the following sections, the [value] of an attribute information item or the [children] of an element information item means a string composed of, in order, the [character code] of each character information item in the [children] of that attribute information item or in the [children] of that element information item respectively.

5.1 XML representation of datatype definitions

The XML representation for a Datatype definition schema component is a simpleType element

information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `simpleType` Element Information Item

```
<simpleType
  id = ID
  name = NCName>
  Content: (annotation? , (restriction | list | union))
</simpleType>
```

```
<restriction
  base = QName>
  Content: ((annotation?) , ( | simpleType? | (minExclusive | minInclusive |
maxExclusive | maxInclusive | precision | scale | length | minLength |
maxLength | encoding | period | duration | enumeration | pattern)*))
</restriction>
```

```
<list
  itemType = QName>
  Content: (annotation?)
</list>
```

```
<union
  memberType = QName>
  Content: (annotation? , (simpleType+))
</union>
```

Datatype Definition Schema Component

Property	Representation
{name}	The value of the `name`[attribute], if present, otherwise null
{base type definition}	If {variety} is list then the value of the `type`[attribute]; else if {variety} is union, then the list of {base type definition} from the `simpleType`[children]; else the value of the `base`[attribute] of the `restriction`[child]
{variety}	*list* if the immediate[child] of `simpleType` (or the {variety} of {base type definition} or any of its ancestors) is `list` is *list*; else *union* if the immediate [child] of `simpleType` (or the {variety} of {base type definition} or any of its ancestors) is `union`; otherwise *atomic*
{target namespace}	The value of the `targetNamespace`[attribute] of the parent `schema` element information item.
{facets}	The union of the set of facet components corresponding to the facet [children] of the `restriction`.
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

A derived datatype can be derived from a primitive datatype or another derived datatype by one of three means: by *restriction*, by *list* or by *union*.

5.1.1 Derivation by restriction

An atomic datatype can be derived from another atomic datatype by restricting its value space and, consequently, lexical space.

Example

An electronic commerce schema might define a datatype called *Sku* (the barcode number that appears on products) from the built-in datatype string by supplying a value for the pattern facet.

```
<simpleType name='Sku'>
    <restriction base='string'>
      <pattern value='\d{3}-[A-Z]{2}'/>
    </restriction>
</simpleType>
```

In this case, *Sku* is the name of the new user-derived datatype, string is its base type and pattern is the facet.

5.1.2 Derivation by list

A list datatype must be derived from an atomic datatype. This yields a list datatype that can contain whitespace separated lists of values of the base type.

Example

A system may want to store lists of floating point values.

```
<simpleType name='listOfFloat'>
  <list itemType='float'/>
</simpleType>
```

In this case, *listOfFloat* is the name of the new user-derived datatype, float is its base type and list is the derivation method.

As mentioned in List datatypes (§2.6.1.2), when a datatype is derived from a list datatype, the following constraining facets may be used:

- length
- maxLength
- minLength
- enumeration

regardless of the constraining facets that are applicable to the atomic datatype that serves as the base type of the list.

For each of the above facets, the *unit of length* is measured in number of list items.

5.1.3 Derivation by union

A union datatype may be derived from two or more atomic, list or other union datatypes.

Example

An example, taken from a typical display oriented text markup language, might want to express font sizes as an integer between 8 and 72, or with one of the tokens "small", "medium" or "large". The union type definition below would accomplish that.

```
<xsd:attribute name="size">
  <xsd:simpleType>
    <xsd:union>
      <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
          <xsd:minInclusive="8"/>
          <xsd:maxInclusive="72"/>
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="small"/>
          <xsd:enumeration value="medium"/>
          <xsd:enumeration value="large"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:union>
  </xsd:simpleType>
</xsd:attribute>
```

```
<p>
<font size='large'>A header</font>
</p>
<p>
<font size='12'>this is a test</font>
</p>
```

As mentioned in Union datatypes (§2.6.1.3), when a datatype is derived from a union datatype, the only following constraining facets may be used:

- pattern
- enumeration

regardless of the constraining facets that may be applicable to the datatypes that participate in the union

5.2 Constraining facets

This section discusses the details of the XML Representation for specifying constraining facets in a datatype definition.

5.2.1 length

The XML representation for a length schema component is a length element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `length` Element Information Item

```
<length
  id = ID
  value = nonNegativeInteger
  fixed = boolean : false>
  Content: (annotation?)
</length>
```

length Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of a user-derived datatype to represent product codes which must be exactly 8 characters in length. By fixing the value of the **length** facet we ensure that types derived from productCode may change or set the values of other facets, such as **pattern**, but cannot change the length.

```
<simpleType name='productCode'>
   <restriction base='string'>
     <length value='8' fixed='true'/>
   </restriction>
</simpleType>
```

5.2.2 minLength

The XML representation for a minLength schema component is a minLength element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `minLength` Element Information Item

```
<minLength
  id = ID
  value = nonNegativeInteger
  fixed = boolean : false>
  Content: (annotation?)
</minLength>
```

minLength Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of a user-derived datatype which requires strings to have at least one character (i.e., the empty string is not in the value space of this datatype).

```
<simpleType name='non-empty-string'>
  <restriction base='string'>
    <minLength value='1'/>
  </restriction>
</simpleType>
```

5.2.3 maxLength

The XML representation for a maxLength schema component is a maxLength element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `maxLength` Element Information Item

```
<maxLength
  id = ID
  value = nonNegativeInteger
  fixed = boolean : false>
  Content: (annotation?)
</maxLength>
```

maxLength Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of a user-derived datatype which might be used to accept form input with an upper limit to the number of characters that are acceptable.

```
<simpleType name='form-input'>
  <restriction base='string'>
    <maxLength value='50'/>
  </restriction>
</simpleType>
```

5.2.4 pattern

The XML representation for a pattern schema component is a pattern element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `pattern` Element Information Item

```
<pattern
  id = ID
  value = string
  fixed = boolean : false>
  Content: (annotation?)
</pattern>
```

{value} must be a valid regular expression.

pattern Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Schema Representation Constraint: Multiple patterns

If multiple pattern element information items appear as [children] of a simpleType, the [value]s should be combined as if they appeared in a single regular expression as separate branches.

Example

The following is the definition of a user-derived datatype which is a better representation of postal codes in the United States, by limiting strings to those which are matched by a specific regular expression.

```
<simpleType name='better-us-zipcode'>
  <restriction base='string'>
    <pattern value='[0-9]{5}(-[0-9]{4})?'/>
  </restriction>
</simpleType>
```

5.2.5 enumeration

The XML representation for a enumeration schema component is a enumeration element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `enumeration` Element Information Item

```
<enumeration
  id = ID
  value = string
  fixed = boolean : false>
  Content: (annotation?)
</enumeration>
```

{value} must be a valid value of the {base type definition}.

enumeration Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Schema Representation Constraint: Multiple patterns

If multiple enumeration element information items appear as [children] of a simpleType the {value} of the enumeration component should be the set of all such [value]s.

Example

The following example is a datatype definition for a user-derived datatype which limits the values of dates to the three US holidays enumerated. This datatype definition would appear in a schema authored by an "end-user" and shows how to define a datatype by enumerating the values in its value space. The enumerated values must be type-valid literals for the base type.

```
<simpleType name='holidays'>
    <annotation>
        <documentation>some US holidays</documentation>
    </annotation>
    <restriction base='recurringDate'>
      <enumeration value='--01-01'>
        <annotation>
            <documentation>New Year's day</documentation>
        </annotation>
      </enumeration>
      <enumeration value='--07-04'/>
        <annotation>
            <documentation>4th of July</documentation>
        </annotation>
      </enumeration>
      <enumeration value='--12-25'/>
        <annotation>
            <documentation>Christmas</documentation>
        </annotation>
      </enumeration>
    </restriction>
</simpleType>
```

5.2.6 maxInclusive

The XML representation for a maxInclusive schema component is a maxInclusive element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `maxInclusive` Element Information Item

```
<maxInclusive
  id = ID
  value = string
  fixed = boolean : false>
  Content: (annotation?)
</maxInclusive>
```

{value} must be a valid value of the {base type definition}.

maxInclusive Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false, if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of a user-derived datatype which limits values to integers less than or equal to 100, using maxInclusive.

```
<simpleType name='one-hundred-or-less'>
  <restriction base='integer'>
    <maxInclusive value='100'/>
  </restriction>
</simpleType>
```

5.2.7 maxExclusive

The XML representation for a maxExclusive schema component is a maxExclusive element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `maxExclusive` Element Information Item

```
<maxExclusive
  id = ID
  value = string
  fixed = boolean : false>
  Content: (annotation?)
</maxExclusive>
```

{value} must be a valid value of the {base type definition}.

maxExclusive Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of a user-derived datatype which limits values to integers less than or equal to 100, using maxExclusive.

```
<simpleType name='less-than-one-hundred-and-one'>
  <restriction base='integer'>
    <maxExclusive value='101'/>
  </restriction>
</simpleType>
```

Note that the value space of this datatype is identical to the previous one (named 'one-hundred-or-less').

5.2.8 minInclusive

The XML representation for a minInclusive schema component is a minInclusive element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `minInclusive` Element Information Item

```
<minInclusive
  id = ID
  value = string
  fixed = boolean : false>
  Content: (annotation?)
</minInclusive>
```

{value} must be a valid value of the {base type definition}.

minInclusive Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of a user-derived datatype which limits values to integers greater than or equal to 100, using minInclusive.

```
<simpleType name='one-hundred-or-more'>
  <restriction base='integer'>
    <minInclusive value='100'/>
  </restriction>
</simpleType>
```

5.2.9 minExclusive

The XML representation for a minExclusive schema component is a minExclusive element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `minExclusive` Element Information Item

```
<minExclusive
  id = ID
  value = string
  fixed = boolean : false>
  Content: (annotation?)
</minExclusive>
```

{value} must be a valid value of the {base type definition}.

minExclusive Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of a user-derived datatype which limits values to integers greater than or equal to 100, using minExclusive.

```
<simpleType name='more-than-ninety-nine'>
  <restriction base='integer'>
    <minExclusive value='99'/>
  </restriction>
</simpleType>
```

Note that the value space of this datatype is identical to the previous one (named 'one-hundred-or-more').

5.2.10 precision

The XML representation for a precision schema component is a precision element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `precision` Element Information Item

```
<precision
  id = ID
  value = nonNegativeInteger
  fixed = boolean : false>
  Content: (annotation?)
</precision>
```

precision Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of a user-derived datatype which could be used to represent monetary amounts, such as in a financial management application which does not have figures above $1M and only allows whole cents. This definition would appear in a schema authored by an "end-user" and shows how to define a datatype by specifying facet values which constrain the range of the base type in a manner specific to the base type (different than specifying max/min values as before).

```
<simpleType name='amount'>
  <restriction base='decimal'>
    <precision value='8'/>
    <scale value='2' fixed='true'/>
  </restriction>
</simpleType>
```

5.2.11 scale

The XML representation for a scale schema component is a scale element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `scale` Element Information Item

```
<scale
  id = ID
  value = nonNegativeInteger
  fixed = boolean : false>
  Content: (annotation?)
</scale>
```

scale Schema Component

Property	Representation
{value}	The value of the `value`[attribute]
{fixed}	The value of the `fixed`[attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following is the definition of auser-derived datatype which could be used to represent the magnitude of a person's body temperature on the farenheit scale. This definition would appear in a schema authored by an "end-user" and shows how to define a datatype by specifying facet values which constrain the range of the base type.

```
<simpleType name='farenheitBodyTemp'>
  <restriction base='decimal'>
    <precision value='4'/>
    <scale value='1'/>
    <minInclusive value='97.5'/>
    <maxInclusive value='105.0'/>
  </restriction>
</simpleType>
```

5.2.12 encoding

The XML representation for a encoding schema component is an encoding element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `encoding` Element Information Item

```
<encoding
  id = ID
  value = hex | base64
  fixed = boolean : false>
  Content: (annotation?)
</encoding>
```

encoding Schema Component

Property	Representation
{value}	The value of the `value` [attribute]
{fixed}	The value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

The following example is a datatype definition for a user-derived datatype whose value space is the set of binary streams of length 4 octets (32 bits) and whose lexical space is the set of base64 encodings of such binary streams. This datatype definition would appear in a schema authored by an "end-user" and shows how to define a datatype by specifying multiple constraining facets.

```
<simpleType name='myBinary'>
  <restriction base='binary'>
    <length value='4'/>
    <encoding value='base64'/>
  </restriction>
</simpleType>
```

5.2.13 duration

The XML representation for a duration schema component is a duration element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `duration` Element Information Item

```
<duration
  id = ID
  value = timeDuration
  fixed = boolean : false>
  Content: (annotation?)
</duration>
```

duration Schema Component

Property	Representation
{value}	The value of the `value`[attribute]
{fixed}	The value of the `fixed`[attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

Suppose an health insurance company wanted to define the length of a hospital stay permitted for the birth of a baby as two days. They may want to define a datatype as below.

```
<simpleType name='birthingStay'>
  <restriction base='timePeriod'>
    <duration value='P2D'/>
  </restriction>
</simpleType>
```

5.2.14 period

The XML representation for a period schema component is a period element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `period` Element Information Item

```
<period
  id = ID
  value = timeDuration
  fixed = boolean : false>
  Content: (annotation?)
</period>
```

period Schema Component

Property	Representation
{value}	The value of the `value`[attribute]
{fixed}	The value of the `fixed`[attribute], if present, otherwise false
{annotation}	The annotation corresponding to the annotation element information item in the [children], if present, otherwise null

Example

Suppose we wanted to define a day of the week i.e. a duration of one day that recurs every 7 days. This could be done as follows:

```
<simpleType name='dayOfWeek'>
  <restriction base='recurringDuration'>
    <duration value='P1D'/>
    <period value='P7D'/>
  </restriction>
</simpleType>
```

6 Conformance

This specification describes two levels of conformance for datatype processors. The first is required of all processors. Support for the other will depend on the application environments for which the processor is intended.

[Definition:] **Minimally conforming** processors must completely and correctly implement the Constraint on Schemas and Validity Contribution.

[Definition:] Processors which accept schemas in the form of XML documents as described in XML representation of datatype definitions (§5.1) are additionally said to provide **conformance to the XML Representation of Schemas**, and must, when processing schema documents, completely and correctly implement all Schema Representation Constraints in this specification, and must adhere exactly to the specifications in XML representation of datatype definitions (§5.1) for mapping the contents of such documents to schema components for use in validation.

> **NOTE:** By separating the conformance requirements relating to the concrete syntax of XML schema documents, this specification admits processors which validate using schemas stored in optimised binary representations, dynamically created schemas represented as programming language data structures, or implementations in which particular schemas are compiled into executable code such as C or Java. Such processors can be said to be minimally conforming but not necessarily in conformance to the XML Representation of Schemas.

A Schema for Datatype Definitions (normative)

```
<?xml version='1.0'?>
<!-- XML Schema schema for XML Schemas: Part 2: Datatypes -->
<!DOCTYPE schema PUBLIC "-//W3C//DTD XMLSCHEMA 200010//EN" "XMLSchema.dtd" [
<!--
     keep this schema XML1.0 DTD valid
  -->
       <!ENTITY % elementAttrs 'xmlns:x CDATA #IMPLIED'>
       <!ENTITY % schemaAttrs 'xmlns:hfp CDATA #IMPLIED'>

       <!ELEMENT hfp:hasFacet EMPTY>
       <!ATTLIST hfp:hasFacet
```

```
                name NMTOKEN #REQUIRED
                >
        <!ELEMENT hfp:hasProperty EMPTY>
        <!ATTLIST hfp:hasProperty
                name NMTOKEN #REQUIRED
                value CDATA #REQUIRED
                >
        ]>
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://www.w3.org/2000/10/XMLSchema"
        version="$Id: datatypes.html,v 1.4 2000/09/22 22:45:23 ht Exp $"
        xmlns:hfp="http://www.w3.org/2000/10/XMLSchema-hasFacetAndProperty">

 <annotation>
   <documentation source="http://www.w3.org/TR/2000/WD-xmlschema-2-20000922/datatypes.htm
   The schema corresponding to this document is normative,
   with respect to the syntactic constraints it expresses in the
   XML Schema language.  The documentation (within &lt;documentation> elements)
   below, is not normative, but rather highlights important aspects of
   the W3C Recommendation of which this is a part</documentation>
 </annotation>

        <import namespace="http://www.w3.org/XML/1998/namespace"
                schemaLocation="http://www.w3.org/2000/10/xml.xsd">
                <annotation>
                        <documentation>
                                Get access to the xml: attribute groups for xml:lang
                        </documentation>
                </annotation>
        </import>

        <annotation>
                <documentation>
                        First the builtin primitive datatypes.  These definitions are for
                        information only, the real builtin definitions are magic.  Note i
                        particular that there is no type named 'anySimpleType'.  The
                        primitives should really be derived from no type at all, and
                        anySimpleType should be derived as a union of all the primitives.
                </documentation>
        </annotation>

        <simpleType name="string">
                <annotation>
                        <appinfo>
                                <hfp:hasFacet name="length"/>
                                <hfp:hasFacet name="minLength"/>
                                <hfp:hasFacet name="maxLength"/>
                                <hfp:hasFacet name="pattern"/>
                                <hfp:hasFacet name="enumeration"/>
                                <hfp:hasProperty name="ordered" value="false"/>
                                <hfp:hasProperty name="bounded" value="false"/>
                                <hfp:hasProperty name="cardinality" value="countably infi
                                <hfp:hasProperty name="numeric" value="false"/>
                        </appinfo>
                </annotation>
                <restriction base="anySimpleType"/>
        </simpleType>

        <simpleType name="boolean">
                <annotation>
                        <appinfo>
                                <hfp:hasFacet name="pattern"/>
                                <hfp:hasProperty name="ordered" value="false"/>
                                <hfp:hasProperty name="bounded" value="false"/>
                                <hfp:hasProperty name="cardinality" value="finite"/>
                                <hfp:hasProperty name="numeric" value="false"/>
                        </appinfo>
                </annotation>
```

```
                <restriction base="anySimpleType"/>
        </simpleType>

        <simpleType name="float">
                <annotation>
                        <appinfo>
                                <hfp:hasFacet name="pattern"/>
                                <hfp:hasFacet name="enumeration"/>
                                <hfp:hasFacet name="maxInclusive"/>
                                <hfp:hasFacet name="maxExclusive"/>
                                <hfp:hasFacet name="minInclusive"/>
                                <hfp:hasFacet name="minExclusive"/>
                                <hfp:hasProperty name="ordered" value="true"/>
                                <hfp:hasProperty name="bounded" value="true"/>
                                <hfp:hasProperty name="cardinality" value="finite"/>
                                <hfp:hasProperty name="numeric" value="true"/>
                        </appinfo>
                </annotation>
                <restriction base="anySimpleType"/>
         </simpleType>

         <simpleType name="double">
                <annotation>
                        <appinfo>
                                <hfp:hasFacet name="pattern"/>
                                <hfp:hasFacet name="enumeration"/>
                                <hfp:hasFacet name="maxInclusive"/>
                                <hfp:hasFacet name="maxExclusive"/>
                                <hfp:hasFacet name="minInclusive"/>
                                <hfp:hasFacet name="minExclusive"/>
                                <hfp:hasProperty name="ordered" value="true"/>
                                <hfp:hasProperty name="bounded" value="true"/>
                                <hfp:hasProperty name="cardinality" value="finite"/>
                                <hfp:hasProperty name="numeric" value="true"/>
                        </appinfo>
                </annotation>
                <restriction base="anySimpleType"/>
         </simpleType>

         <simpleType name="decimal">
                <annotation>
                        <appinfo>
                                <hfp:hasFacet name="precision"/>
                                <hfp:hasFacet name="scale"/>
                                <hfp:hasFacet name="pattern"/>
                                <hfp:hasFacet name="enumeration"/>
                                <hfp:hasFacet name="maxInclusive"/>
                                <hfp:hasFacet name="maxExclusive"/>
                                <hfp:hasFacet name="minInclusive"/>
                                <hfp:hasFacet name="minExclusive"/>
                                <hfp:hasProperty name="ordered" value="true"/>
                                <hfp:hasProperty name="bounded" value="false"/>
                                <hfp:hasProperty name="cardinality" value="countably infi
                                <hfp:hasProperty name="numeric" value="true"/>
                        </appinfo>
                </annotation>
                <restriction base="anySimpleType"/>
         </simpleType>

         <simpleType name="timeDuration">
                <annotation>
                        <appinfo>
                                <hfp:hasFacet name="pattern"/>
                                <hfp:hasFacet name="enumeration"/>
                                <hfp:hasFacet name="maxInclusive"/>
                                <hfp:hasFacet name="maxExclusive"/>
                                <hfp:hasFacet name="minInclusive"/>
                                <hfp:hasFacet name="minExclusive"/>
```

```
                        <hfp:hasProperty name="ordered" value="true"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <restriction base="anySimpleType"/>
</simpleType>

<simpleType name="recurringDuration">
        <annotation>
                <appinfo>
                        <hfp:hasFacet name="duration"/>
                        <hfp:hasFacet name="period"/>
                        <hfp:hasFacet name="pattern"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasFacet name="maxInclusive"/>
                        <hfp:hasFacet name="maxExclusive"/>
                        <hfp:hasFacet name="minInclusive"/>
                        <hfp:hasFacet name="minExclusive"/>
                        <hfp:hasProperty name="ordered" value="false"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <restriction base="anySimpleType"/>
</simpleType>

<simpleType name="binary">
        <annotation>
                <appinfo>
                        <hfp:hasFacet name="encoding"/>
                        <hfp:hasFacet name="length"/>
                        <hfp:hasFacet name="minLength"/>
                        <hfp:hasFacet name="maxLength"/>
                        <hfp:hasFacet name="pattern"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasProperty name="ordered" value="false"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <restriction base="anySimpleType"/>
</simpleType>

<simpleType name="uriReference">
        <annotation>
                <appinfo>
                        <hfp:hasFacet name="length"/>
                        <hfp:hasFacet name="minLength"/>
                        <hfp:hasFacet name="maxLength"/>
                        <hfp:hasFacet name="pattern"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasProperty name="ordered" value="false"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <restriction base="anySimpleType"/>
</simpleType>

<simpleType name="ID">
        <annotation>
                <appinfo>
                        <hfp:hasFacet name="length"/>
```

```
                        <hfp:hasFacet name="minLength"/>
                        <hfp:hasFacet name="maxLength"/>
                        <hfp:hasFacet name="pattern"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasFacet name="maxInclusive"/>
                        <hfp:hasFacet name="maxExclusive"/>
                        <hfp:hasFacet name="minInclusive"/>
                        <hfp:hasFacet name="minExclusive"/>
                        <hfp:hasProperty name="ordered" value="true"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <restriction base="anySimpleType"/>
 </simpleType>

 <simpleType name="IDREF">
        <annotation>
                <appinfo>
                        <hfp:hasFacet name="length"/>
                        <hfp:hasFacet name="minLength"/>
                        <hfp:hasFacet name="maxLength"/>
                        <hfp:hasFacet name="pattern"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasFacet name="maxInclusive"/>
                        <hfp:hasFacet name="maxExclusive"/>
                        <hfp:hasFacet name="minInclusive"/>
                        <hfp:hasFacet name="minExclusive"/>
                        <hfp:hasProperty name="ordered" value="true"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <restriction base="anySimpleType"/>
 </simpleType>

<simpleType name="ENTITY">
        <annotation>
                <appinfo>
                        <hfp:hasFacet name="length"/>
                        <hfp:hasFacet name="minLength"/>
                        <hfp:hasFacet name="maxLength"/>
                        <hfp:hasFacet name="pattern"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasFacet name="maxInclusive"/>
                        <hfp:hasFacet name="maxExclusive"/>
                        <hfp:hasFacet name="minInclusive"/>
                        <hfp:hasFacet name="minExclusive"/>
                        <hfp:hasProperty name="ordered" value="true"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <restriction base="anySimpleType"/>
</simpleType>

<simpleType name="NOTATION">
        <annotation>
                <appinfo>
                        <hfp:hasFacet name="length"/>
                        <hfp:hasFacet name="minLength"/>
                        <hfp:hasFacet name="maxLength"/>
                        <hfp:hasFacet name="pattern"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasFacet name="maxInclusive"/>
```

```
                        <hfp:hasFacet name="maxExclusive"/>
                        <hfp:hasFacet name="minInclusive"/>
                        <hfp:hasFacet name="minExclusive"/>
                        <hfp:hasProperty name="ordered" value="true"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <restriction base="anySimpleType"/>
</simpleType>

<simpleType name="QName">
        <annotation>
            <appinfo>
                        <hfp:hasFacet name="length"/>
                        <hfp:hasFacet name="minLength"/>
                        <hfp:hasFacet name="maxLength"/>
                        <hfp:hasFacet name="pattern"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasFacet name="maxInclusive"/>
                        <hfp:hasFacet name="maxExclusive"/>
                        <hfp:hasFacet name="minInclusive"/>
                        <hfp:hasFacet name="minExclusive"/>
                        <hfp:hasProperty name="ordered" value="true"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
      </annotation>
        <restriction base="anySimpleType"/>
</simpleType>

<annotation>
        <documentation>
                Now the derived primitive types
        </documentation>
</annotation>

<simpleType name="language">
        <restriction base="string">
                <pattern value="([a-zA-Z]{2}|[iI]-[a-zA-Z]+|[xX]-[a-zA-Z]+)(-[a-z
                        <annotation>
                                <documentation source="http://www.w3.org/TR/REC-x
                                        pattern matches production 33 from the XM
                                </documentation>
                        </annotation>
                </pattern>
        </restriction>
</simpleType>

<simpleType name="IDREFS">
        <annotation>
                <appinfo>
                        <hfp:hasFacet name="length"/>
                        <hfp:hasFacet name="minLength"/>
                        <hfp:hasFacet name="maxLength"/>
                        <hfp:hasFacet name="enumeration"/>
                        <hfp:hasProperty name="ordered" value="false"/>
                        <hfp:hasProperty name="bounded" value="false"/>
                        <hfp:hasProperty name="cardinality" value="countably infi
                        <hfp:hasProperty name="numeric" value="false"/>
                </appinfo>
        </annotation>
        <list itemType="IDREF"/>
</simpleType>

<simpleType name="ENTITIES">
```

```
                <annotation>
                        <appinfo>
                                <hfp:hasFacet name="length"/>
                                <hfp:hasFacet name="minLength"/>
                                <hfp:hasFacet name="maxLength"/>
                                <hfp:hasFacet name="enumeration"/>
                                <hfp:hasProperty name="ordered" value="false"/>
                                <hfp:hasProperty name="bounded" value="false"/>
                                <hfp:hasProperty name="cardinality" value="countably infi
                                <hfp:hasProperty name="numeric" value="false"/>
                        </appinfo>
                </annotation>
                <list itemType="ENTITY"/>
        </simpleType>

        <simpleType name="NMTOKEN">
                <restriction base="string">
                        <pattern value="\c+">
                                <annotation>
                                        <documentation source="http://www.w3.org/TR/REC-x
                                                pattern matches production 7 from the XML
                                        </documentation>
                                </annotation>
                        </pattern>
                </restriction>
        </simpleType>

        <simpleType name="NMTOKENS">
                <annotation>
                        <appinfo>
                                <hfp:hasFacet name="length"/>
                                <hfp:hasFacet name="minLength"/>
                                <hfp:hasFacet name="maxLength"/>
                                <hfp:hasFacet name="enumeration"/>
                                <hfp:hasProperty name="ordered" value="true"/>
                                <hfp:hasProperty name="bounded" value="false"/>
                                <hfp:hasProperty name="cardinality" value="countably infi
                                <hfp:hasProperty name="numeric" value="false"/>
                        </appinfo>
                </annotation>
                <list itemType="NMTOKEN"/>
        </simpleType>

        <simpleType name="Name">
                <restriction base="string">
                        <pattern value="\i\c*">
                                <annotation>
                                        <documentation source="http://www.w3.org/TR/REC-x
                                                pattern matches production 5 from the XML
                                        </documentation>
                                </annotation>
                        </pattern>
                </restriction>
        </simpleType>

        <simpleType name="NCName">
                <restriction base="Name">
                        <pattern value="[\i-[:]][\c-[:]]*">
                                <annotation>
                                        <documentation source="http://www.w3.org/TR/REC-x
                                                pattern matches production 4 from the Nam
                                        </documentation>
                                </annotation>
                        </pattern>
                </restriction>
        </simpleType>

        <simpleType name="integer">
```

```
                <restriction base="decimal">
                        <scale value="0" fixed="true"/>
                </restriction>
        </simpleType>

        <simpleType name="nonPositiveInteger">
                <restriction base="integer">
                        <maxInclusive value="0"/>
                </restriction>
        </simpleType>

        <simpleType name="negativeInteger">
                <restriction base="nonPositiveInteger">
                        <maxInclusive value="-1"/>
                </restriction>
        </simpleType>

        <simpleType name="long">
                <annotation>
                        <appinfo>
                                <hfp:hasProperty name="bounded" value="true"/>
                                <hfp:hasProperty name="cardinality" value="finite"/>
                        </appinfo>
                </annotation>
                <restriction base="integer">
                        <minInclusive value="-9223372036854775808"/>
                        <maxInclusive value="9223372036854775807"/>
                </restriction>
        </simpleType>

        <simpleType name="int">
                <restriction base="long">
                        <minInclusive value="-2147483648"/>
                        <maxInclusive value="2147483647"/>
                </restriction>
        </simpleType>

        <simpleType name="short">
                <restriction base="int">
                        <minInclusive value="-32768"/>
                        <maxInclusive value="32767"/>
                </restriction>
        </simpleType>

        <simpleType name="byte">
                <restriction base="short">
                        <minInclusive value="-128"/>
                        <maxInclusive value="127"/>
                </restriction>
        </simpleType>

        <simpleType name="nonNegativeInteger">
                <restriction base="integer">
                        <minInclusive value="0"/>
                </restriction>
        </simpleType>

        <simpleType name="unsignedLong">
                <annotation>
                        <appinfo>
                                <hfp:hasProperty name="bounded" value="true"/>
                                <hfp:hasProperty name="cardinality" value="finite"/>
                        </appinfo>
                </annotation>
                <restriction base="nonNegativeInteger">
                        <maxInclusive value="18446744073709551615"/>
                </restriction>
        </simpleType>
```

```
        <simpleType name="unsignedInt">
                <restriction base="unsignedLong">
                        <maxInclusive value="4294967295"/>
                </restriction>
        </simpleType>

        <simpleType name="unsignedShort">
                <restriction base="unsignedInt">
                        <maxInclusive value="65535"/>
                </restriction>
        </simpleType>

        <simpleType name="unsignedByte">
                <restriction base="unsignedShort">
                        <maxInclusive value="255"/>
                </restriction>
        </simpleType>

        <simpleType name="positiveInteger">
                <restriction base="nonNegativeInteger">
                        <minInclusive value="1"/>
                </restriction>
        </simpleType>

        <simpleType name="timeInstant">
                <restriction base="recurringDuration">
                        <duration value="P0Y" fixed="true"/>
                        <period value="P0Y" fixed="true"/>
                </restriction>
        </simpleType>

        <simpleType name="time">
                <restriction base="recurringDuration">
                        <period value="PT24H" fixed="true"/>
                        <duration value="P0Y" fixed="true"/>
                </restriction>
        </simpleType>

        <simpleType name="timePeriod">
                <restriction base="recurringDuration">
                        <period value="P0Y" fixed="true"/>
                </restriction>
        </simpleType>

        <simpleType name="date">
                <restriction base="timePeriod">
                        <duration value="PT24H" fixed="true"/>
                </restriction>
        </simpleType>

        <simpleType name="month">
                <restriction base="timePeriod">
                        <duration value="P1M" fixed="true"/>
                </restriction>
        </simpleType>

        <simpleType name="year">
                <restriction base="timePeriod">
                        <duration value="P1Y" fixed="true"/>
                </restriction>
        </simpleType>

        <simpleType name="century">
                <restriction base="timePeriod">
                        <period value="P100Y" fixed="true"/>
                </restriction>
        </simpleType>
```

```
        <simpleType name="recurringDate">
                <restriction base="recurringDuration">
                        <duration value="P24H" fixed="true"/>
                        <period value="P1Y" fixed="true"/>
                </restriction>
        </simpleType>

        <simpleType name="recurringDay">
                <restriction base="recurringDuration">
                        <duration value="P24H" fixed="true"/>
                        <period value="P1M" fixed="true"/>
                </restriction>
        </simpleType>

  <complexType name="openAttrs">
   <annotation>
    <documentation>This type is extended by almost all schema types
                   to allow attributes from other namespaces to be
                   added to user schemas.</documentation>
    </annotation>
   <complexContent>
    <restriction base="anyType">
     <anyAttribute namespace="##other" processContents="lax"/>
    </restriction>
   </complexContent>
  </complexType>

  <complexType name="annotated">
<annotation>
    <documentation>This type is extended by all types which allow annotation
          other than &lt;schema&gt; itself</documentation>
    </annotation>
    <complexContent>
      <extension base="openAttrs">
       <sequence>
        <element ref="annotation" minOccurs="0"/>
       </sequence>
       <attribute name="id" type="ID"/>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="simpleType" abstract="true">
    <complexContent>
      <extension base="annotated">
    <sequence>
     <element ref="simpleDerivation"/>
    </sequence>
    <attribute name="name" type="NCName">
     <annotation>
      <documentation>Can be restricted to required or forbidden</documentation>
     </annotation>
    </attribute>
        </extension>
    </complexContent>
  </complexType>

  <complexType name="topLevelSimpleType">
    <complexContent>
      <restriction base="simpleType">
   <sequence>
    <element ref="annotation" minOccurs="0"/>
    <element ref="simpleDerivation"/>
   </sequence>
   <attribute name="name" use="required" type="NCName">
     <annotation>
       <documentation>Required at the top level</documentation>
```

```
    </annotation>
   </attribute>
        </restriction>
    </complexContent>
  </complexType>

  <complexType name="localSimpleType">
    <complexContent>
      <restriction base="simpleType">
   <sequence>
    <element ref="annotation" minOccurs="0"/>
    <element ref="simpleDerivation"/>
   </sequence>
   <attribute name="name" use="prohibited">
     <annotation>
       <documentation>Forbidden when nested</documentation>
     </annotation>
   </attribute>
        </restriction>
    </complexContent>
  </complexType>

  <element name="simpleType" substitutionGroup="redefinable" type="topLevelSimpleType"/>

  <element name="simpleDerivation" abstract="true" type="annotated"/>

  <group name="simpleRestrictionModel">
     <choice>
      <element ref="facet" minOccurs="0" maxOccurs="unbounded"/>
     </choice>
  </group>

  <element name="restriction" substitutionGroup="simpleDerivation">
   <complexType>
<annotation>
     <documentation>base attribute and simpleType child are mutually
                    exclusive, but one or other is required</documentation>
    </annotation>
      <complexContent>
        <extension base="annotated">

    <sequence>
     <element name="simpleType" type="localSimpleType" minOccurs="0"/>
     <group ref="simpleRestrictionModel"/>
    </sequence>
    <attribute name="base" type="QName" use="optional"/>
          </extension>
      </complexContent>
    </complexType>
  </element>

  <element name="list" substitutionGroup="simpleDerivation">
   <complexType>
<annotation>
     <documentation>type attribute and simpleType child are mutually
                    exclusive, but one or other is required</documentation>
    </annotation>
      <complexContent>
        <extension base="annotated">

    <sequence>
     <element name="simpleType" type="localSimpleType" minOccurs="0"/>
    </sequence>
    <attribute name="itemType" type="QName" use="optional"/>
          </extension>
      </complexContent>
    </complexType>
  </element>
```

```
  <element name="union" substitutionGroup="simpleDerivation">
   <complexType>
<annotation>
      <documentation>types attribute must be non-empty or there must be
                     at least one simpleType child</documentation>
    </annotation>
      <complexContent>
        <extension base="annotated">

    <sequence>
     <element name="simpleType" type="localSimpleType" minOccurs="0" maxOccurs="unbounded
    </sequence>
    <attribute name="memberTypes" use="optional">
     <simpleType>
      <list itemType="QName"/>
     </simpleType>
    </attribute>
          </extension>
      </complexContent>
    </complexType>
  </element>

  <complexType name="facet">
    <complexContent>
      <extension base="annotated">
    <attribute name="value" use="required"/>
    <attribute name="fixed" type="boolean" use="optional"/>
       </extension>
    </complexContent>
  </complexType>

       <element name="facet" type="facet" abstract="true"/>

       <element name="minBound" abstract="true" substitutionGroup="facet"/>

       <element name="minExclusive" substitutionGroup="minBound"/>
       <element name="minInclusive" substitutionGroup="minBound"/>

       <element name="maxBound" abstract="true" substitutionGroup="facet"/>

       <element name="maxExclusive" substitutionGroup="maxBound"/>
       <element name="maxInclusive" substitutionGroup="maxBound"/>

  <complexType name="numFacet">
    <complexContent>
      <restriction base="facet">
       <sequence>
        <element ref="annotation" minOccurs="0"/>
       </sequence>
    <attribute name="value" type="nonNegativeInteger"/>
       </restriction>
    </complexContent>
  </complexType>


       <element name="precision" substitutionGroup="facet">
              <complexType>
               <complexContent>
                <restriction base="numFacet">
                 <sequence>
                  <element ref="annotation" minOccurs="0"/>
                 </sequence>
                 <attribute name="value" type="positiveInteger"/>
                </restriction>
               </complexContent>
              </complexType>
       </element>
```

```
      <element name="scale" type="numFacet" substitutionGroup="facet"/>

      <element name="length" type="numFacet" substitutionGroup="facet"/>
      <element name="minLength" type="numFacet" substitutionGroup="facet"/>
      <element name="maxLength" type="numFacet" substitutionGroup="facet"/>

      <element name="encoding" substitutionGroup="facet">
  <complexType>
     <complexContent>
       <restriction base="facet">
      <sequence>
       <element ref="annotation" minOccurs="0"/>
      </sequence>
                     <attribute name="value">
                             <simpleType>
                                     <restriction base="NMTOKEN">
                                             <enumeration value="hex">
                                                     <annotation>
                                                             <documentation>
                                                                     each (8-bit) byte
                                                                     of 2 hexidecimal
                                                             </documentation>
                                                     </annotation>
                                             </enumeration>
                                             <enumeration value="base64">
                                                     <annotation>
                                                             <documentation>
                                                                     value is encoded
                                                                     in the MIME RFC
                                                             </documentation>
                                                     </annotation>
                                             </enumeration>
                                     </restriction>
                             </simpleType>
                     </attribute>
          </restriction>
     </complexContent>
               </complexType>
       <s/element>

      <element name="period" substitutionGroup="facet">
   <complexType>
     <complexContent>
       <restriction base="facet">
      <sequence>
       <element ref="annotation" minOccurs="0"/>
      </sequence>
                     <attribute name="value" type="timeDuration"/>
          </restriction>
     </complexContent>
               </complexType>
       </element>

      <element name="duration" substitutionGroup="facet">
   <complexType>
     <complexContent>
       <restriction base="facet">
      <sequence>
       <element ref="annotation" minOccurs="0"/>
      </sequence>
                     <attribute name="value" type="timeDuration"/>
          </restriction>
     </complexContent>
               </complexType>
       </element>

      <element name="enumeration" substitutionGroup="facet"/>
```

```
        <element name="pattern" substitutionGroup="facet"/>

        <element name="appinfo">
    <complexType mixed="true">
                        <sequence minOccurs="0" maxOccurs="unbounded">
                                <any processContents="lax"/>
                        </sequence>
                        <attribute name="source" type="uriReference"/>
                </complexType>
        </element>

        <element name="documentation" xmlns:x="http://www.w3.org/XML/1998/namespace">
    <complexType mixed="true">
                        <sequence minOccurs="0" maxOccurs="unbounded">
                                <any processContents="lax"/>
                        </sequence>
                        <attribute name="source" type="uriReference"/>
                        <attribute ref="x:lang"/>
                </complexType>
        </element>

        <element name="annotation">
                <complexType>
                        <choice minOccurs="0" maxOccurs="unbounded">
                                <element ref="appinfo"/>
                                <element ref="documentation"/>
                        </choice>
                </complexType>
        </element>

 <simpleType name="anySimpleType">
  <annotation>
   <documentation>Not the real thing, provided just for
                  completeness</documentation>
  </annotation>
  <restriction base="anyType"/>
 </simpleType>

</schema>
```

B DTD for Datatype Definitions (non-normative)

```
<!-- DTD for XML Schemas: Part 2: Datatypes -->
<!-- $Id: datatypes.html,v 1.4 2000/09/22 22:45:23 ht Exp $ -->
<!-- Define all the element names, with optional prefix -->
<!ENTITY % simpleType "%p;simpleType">
<!ENTITY % restriction "%p;restriction">
<!ENTITY % list "%p;list">
<!ENTITY % union "%p;union">
<!ENTITY % maxExclusive "%p;maxExclusive">
<!ENTITY % minExclusive "%p;minExclusive">
<!ENTITY % maxInclusive "%p;maxInclusive">
<!ENTITY % minInclusive "%p;minInclusive">
<!ENTITY % precision "%p;precision">
<!ENTITY % scale "%p;scale">
<!ENTITY % length "%p;length">
<!ENTITY % minLength "%p;minLength">
<!ENTITY % maxLength "%p;maxLength">
<!ENTITY % enumeration "%p;enumeration">
<!ENTITY % pattern "%p;pattern">
<!ENTITY % encoding "%p;encoding">
<!ENTITY % period "%p;period">
```

```
<!ENTITY % duration "%p;duration">

<!-- Customisation entities for the ATTLIST of each element type.
     Define one of these if your schema takes advantage of the
     anyAttribute='##other' in the schema for schemas -->

<!ENTITY % simpleTypeAttrs ''>
<!ENTITY % restrictionAttrs ''>
<!ENTITY % listAttrs ''>
<!ENTITY % unionAttrs ''>
<!ENTITY % simpleTypeAttrs ''>
<!ENTITY % maxExclusiveAttrs ''>
<!ENTITY % minExclusiveAttrs ''>
<!ENTITY % maxInclusiveAttrs ''>
<!ENTITY % minInclusiveAttrs ''>
<!ENTITY % precisionAttrs ''>
<!ENTITY % scaleAttrs ''>
<!ENTITY % lengthAttrs ''>
<!ENTITY % minLengthAttrs ''>
<!ENTITY % maxLengthAttrs ''>
<!ENTITY % enumerationAttrs ''>
<!ENTITY % patternAttrs ''>
<!ENTITY % encodingAttrs ''>
<!ENTITY % periodAttrs ''>
<!ENTITY % durationAttrs ''>
<!ENTITY % appinfoAttrs ''>
<!ENTITY % documentationAttrs ''>


<!-- annotation elements -->
<!ENTITY % annotation "%p;annotation">
<!ENTITY % appinfo "%p;appinfo">
<!ENTITY % documentation "%p;documentation">


<!-- Define some entities for informative use as attribute types -->
<!ENTITY % URIref "CDATA">
<!ENTITY % XPathExpr "CDATA">
<!ENTITY % QName "NMTOKEN">
<!ENTITY % QNames "NMTOKENS">
<!ENTITY % NCName "NMTOKEN">
<!ENTITY % nonNegativeInteger "NMTOKEN">
<!ENTITY % boolean "(true|false)">

<!-- Note that the use of 'facet' below is less restrictive than is
     really intended:  There should in fact be no more than one of each of
     minInclusive, minExclusive, maxInclusive, maxExclusive,
     precision, scale,
     length, maxLength, minLength, encoding, period within datatype,
     and the min- and max- variants of Inclusive and Exclusive are
     mutually exclusive.
     On the other hand,  pattern and enumeration may repeat -->
<!ENTITY % minBound '(%minInclusive; | %minExclusive;)'>
<!ENTITY % maxBound '(%maxInclusive; | %maxExclusive;)'>
<!ENTITY % bounds '%minBound; | %maxBound;'>
<!ENTITY % numeric '%precision; | %scale;'>
<!ENTITY % ordered '%bounds; | %numeric;'>
<!ENTITY % unordered
   '%pattern; | %enumeration; | %length; | %maxLength; | %minLength;
    | %encoding; | %period; | %duration;'>
<!ENTITY % facet '%ordered; | %unordered;'>
<!ENTITY % facetAttr 'value CDATA #REQUIRED'>
<!ENTITY % fixedAttr 'fixed %boolean; #IMPLIED'>
<!ENTITY % facetModel '(%annotation;)?'>
<!ELEMENT %simpleType; ((%annotation;)?, (%restriction; | %list; | %union;))>
<!ATTLIST %simpleType;
    name      %NCName; #IMPLIED
    %simpleTypeAttrs;>
```

```
<!-- name is required at top level -->
<!ELEMENT %restriction; ((%annotation;)?,
                         (%restriction1; |
                          ((%simpleType;)?,(%facet;)*)),
                         (%attrDecls;))>
<!ATTLIST %restriction;
    base      %QName;                  #IMPLIED
    %restrictionAttrs;>
<!-- base and simpleType child are mutually exclusive, one is required -->
<!-- restriction is shared between simpleType and simpleContent and -->
<!-- complexContent (in XMLSchema.xsd). restriction1 is for the latter -->
<!-- cases, when this is restricting a complex type, as is attrDecls -->
<!ELEMENT %list; ((%annotation;)?,(%simpleType;)?)>
<!ATTLIST %list;
    itemType      %QName;             #IMPLIED
    %listAttrs;>
<!-- itemType and simpleType child are mutually exclusive, one is required -->
<!ELEMENT %union; ((%annotation;)?,(%simpleType;)*)>
<!ATTLIST %union;
    memberTypes   %QNames;            #IMPLIED
    %unionAttrs;>
<!-- At least one item in memberTypes or one simpleType child is required -->

<!ELEMENT %maxExclusive; %facetModel;>
<!ATTLIST %maxExclusive;
        %facetAttr;
        %fixedAttr;
    %maxExclusiveAttrs;>
<!ELEMENT %minExclusive; %facetModel;>
<!ATTLIST %minExclusive;
        %facetAttr;
        %fixedAttr;
        %minExclusiveAttrs;>

<!ELEMENT %maxInclusive; %facetModel;>
<!ATTLIST %maxInclusive;
        %facetAttr;
        %fixedAttr;
        %maxInclusiveAttrs;>
<!ELEMENT %minInclusive; %facetModel;>
<!ATTLIST %minInclusive;
        %facetAttr;
        %fixedAttr;
        %minInclusiveAttrs;>

<!ELEMENT %precision; %facetModel;>
<!ATTLIST %precision;
        %facetAttr;
        %fixedAttr;
        %precisionAttrs;>
<!ELEMENT %scale; %facetModel;>
<!ATTLIST %scale;
        %facetAttr;
        %fixedAttr;
        %scaleAttrs;>

<!ELEMENT %length; %facetModel;>
<!ATTLIST %length;
        %facetAttr;
        %fixedAttr;
        %lengthAttrs;>
<!ELEMENT %minLength; %facetModel;>
<!ATTLIST %minLength;
        %facetAttr;
        %fixedAttr;
        %minLengthAttrs;>
<!ELEMENT %maxLength; %facetModel;>
<!ATTLIST %maxLength;
```

```
        %facetAttr;
        %fixedAttr;
        %maxLengthAttrs;>

<!-- This one can be repeated -->
<!ELEMENT %enumeration; %facetModel;>
<!ATTLIST %enumeration;
        %facetAttr;
        %enumerationAttrs;>

<!-- This one can be repeated -->
<!ELEMENT %pattern; %facetModel;>
<!ATTLIST %pattern;
        %facetAttr;
        %patternAttrs;>

<!ELEMENT %encoding; %facetModel;>
<!ATTLIST %encoding;
        %facetAttr;
        %fixedAttr;
        %encodingAttrs;>
<!ELEMENT %period; %facetModel;>
<!ATTLIST %period;
        %facetAttr;
        %fixedAttr;
        %periodAttrs;>
<!ELEMENT %duration; %facetModel;>
<!ATTLIST %duration;
        %facetAttr;
        %fixedAttr;
        %durationAttrs;>

<!-- Annotation is either application information or documentation -->
<!-- By having these here they are available for datatypes as well
     as all the structures elements -->

<!ELEMENT %annotation; (%appinfo; | %documentation;)*>

<!-- User must define annotation elements in internal subset for this
     to work -->
<!ELEMENT %appinfo; ANY>   <!-- too restrictive -->
<!ATTLIST %appinfo;
          source     %URIref;      #IMPLIED
          %appinfoAttrs;>
<!ELEMENT %documentation; ANY>   <!-- too restrictive -->
<!ATTLIST %documentation;
          source     %URIref;   #IMPLIED
          xml:lang   CDATA      #IMPLIED
          %documentationAttrs;>
```

C Datatypes and Facets

C.1 Fundamental Facets

The following table shows the values of the fundamental facets for each built-in datatype.

	Datatype	ordered	bounded	cardinality	numeric
	string	false	false	countably infinite	false
	boolean	false	false	finite	false

primitive	float	true	true	finite	true
	double	true	true	finite	true
	decimal	true	false	countably infinite	true
	timeDuration	true	false	countably infinite	false
	recurringDuration	false	false	countably infinite	false
	binary	false	false	countably infinite	false
	uriReference	false	false	countably infinite	false
	ID	true	false	countably infinite	false
	IDREF	true	false	countably infinite	false
	ENTITY	true	false	countably infinite	false
	NOTATION	true	false	countably infinite	false
	QName	true	false	countably infinite	false
derived	language	false	false	countably infinite	false
	IDREFS	false	false	countably infinite	false
	ENTITIES	false	false	countably infinite	false
	NMTOKEN	false	false	countably infinite	false
	NMTOKENS	true	false	countably infinite	false
	Name	false	false	countably infinite	false
	NCName	false	false	countably infinite	false
	integer	true	false	countably infinite	true
	nonPositiveInteger	true	false	countably infinite	true
	negativeInteger	true	false	countably infinite	true
	long	true	true	finite	true
	int	true	true	finite	true
	short	true	true	finite	true
	byte	true	true	finite	true
	nonNegativeInteger	true	false	countably infinite	true
	unsignedLong	true	true	finite	true
	unsignedInt	true	true	finite	true
	unsignedShort	true	true	finite	true
	unsignedByte	true	true	finite	true
	positiveInteger	true	false	countably infinite	true
	timeInstant	false	false	countably infinite	false
	time	false	false	countably infinite	false
	timePeriod	false	false	countably infinite	false
	date	false	false	countably infinite	false
	month	false	false	countably infinite	false
	year	false	false	countably infinite	false
	century	false	false	countably infinite	false

	recurringDate	false	false	countably infinite	false
	recurringDay	false	false	countably infinite	false

C.2 Constraining Facets

The constraining facets are listed below with all the primitive and derived datatypes that they apply to.

length applies to the following datatypes:

- string
- language
- NMTOKEN
- Name
- NCName
- binary
- uriReference
- ID
- IDREF
- ENTITY
- NOTATION
- QName
- IDREFS
- ENTITIES
- NMTOKENS

minLength applies to the following datatypes:

- string
- language
- NMTOKEN
- Name
- NCName
- binary
- uriReference
- ID
- IDREF
- ENTITY
- NOTATION
- QName
- IDREFS
- ENTITIES
- NMTOKENS

maxLength applies to the following datatypes:

- string
- language
- NMTOKEN
- Name
- NCName

- binary
- uriReference
- ID
- IDREF
- ENTITY
- NOTATION
- QName
- IDREFS
- ENTITIES
- NMTOKENS

pattern applies to the following datatypes:

- string
- language
- NMTOKEN
- Name
- NCName
- boolean
- float
- double
- decimal
- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger
- timeDuration
- recurringDuration
- timeInstant
- time
- timePeriod
- date
- month
- year
- century
- recurringDate
- recurringDay
- binary
- uriReference
- ID
- IDREF
- ENTITY
- NOTATION

- QName

enumeration applies to the following datatypes:

- string
- language
- NMTOKEN
- Name
- NCName
- float
- double
- decimal
- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger
- timeDuration
- recurringDuration
- timeInstant
- time
- timePeriod
- date
- month
- year
- century
- recurringDate
- recurringDay
- binary
- uriReference
- ID
- IDREF
- ENTITY
- NOTATION
- QName
- IDREFS
- ENTITIES
- NMTOKENS

maxInclusive applies to the following datatypes:

- float
- double
- decimal

- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger
- timeDuration
- recurringDuration
- timeInstant
- time
- timePeriod
- date
- month
- year
- century
- recurringDate
- recurringDay
- ID
- IDREF
- ENTITY
- NOTATION
- QName

maxExclusive applies to the following datatypes:

- float
- double
- decimal
- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger
- timeDuration
- recurringDuration
- timeInstant
- time

- timePeriod
- date
- month
- year
- century
- recurringDate
- recurringDay
- ID
- IDREF
- ENTITY
- NOTATION
- QName

minInclusive applies to the following datatypes:

- float
- double
- decimal
- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger
- timeDuration
- recurringDuration
- timeInstant
- time
- timePeriod
- date
- month
- year
- century
- recurringDate
- recurringDay
- ID
- IDREF
- ENTITY
- NOTATION
- QName

minExclusive applies to the following datatypes:

- float
- double

- decimal
- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger
- timeDuration
- recurringDuration
- timeInstant
- time
- timePeriod
- date
- month
- year
- century
- recurringDate
- recurringDay
- ID
- IDREF
- ENTITY
- NOTATION
- QName

precision applies to the following datatypes:

- decimal
- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger

scale applies to the following datatypes:

- decimal
- integer

- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger

encoding applies to the following datatypes:

- binary

duration applies to the following datatypes:

- recurringDuration
- timeInstant
- time
- timePeriod
- date
- month
- year
- century
- recurringDate
- recurringDay

period applies to the following datatypes:

- recurringDuration
- timeInstant
- time
- timePeriod
- date
- month
- year
- century
- recurringDate
- recurringDay

D ISO 8601 Date and Time Formats

D.1 ISO 8601 Conventions

The two primitive datatypes described above, timeDuration, recurringDuration, and the five derived dataypes timeInstant, date, time, timePeriod, and recurringDate use lexical formats inspired by [ISO 8601]. This appendix provides more detail on the ISO formats and discusses some deviations from them for the datatypes defined in this specification.

[ISO 8601] "specifies the representation of dates in the Gregorian calendar and times and representations of periods of time". It should be pointed out that the datatypes described in this specification do not cover all the types of data covered by [ISO 8601], nor do they support all the lexical representations for those types of data.

[ISO 8601] lexical formats are described using "pictures" in which characters are used in place of digits. These characters have the following meanings:

- C -- represents a digit used in the thousands and hundreds components, the "century" component, of the time element "year". Legal values are from 0 to 9.
- Y -- represents a digit used in the tens and units components of the time element "year". Legal values are from 0 to 9.
- M -- represents a digit used in the time element "month". The two digits in a MM format can have values from 1 to 12.
- D -- represents a digit used in the time element "day". The two digits in a DD format can have values from 1 to 28 if the month value equals 2, 1 to 29 if the month value equals 2 and the year is a leap year, 1 to 30 if the month value equals 4, 6, 9 or 11, and 1 to 31 if the month value equals 1, 3, 5, 7, 8, 10 or 12.
- h -- represents a digit used in the time element "hour". The two digits in a hh format can have values from 0 to 12.
- m -- represents a digit used in the time element "minute". The two digits in a mm format can have values from 0 to 60.
- s -- represents a digit used in the time element "second". The two digits in a ss format can have values from 0 to 60. In the formats described in this specification the whole number of seconds may be followed by decimal seconds to an arbitrary level of precision. This is represented in the picture by "ss.sss".

For all the information items indicated by the above characters, leading zeros are required where indicated.

In addition to the above, certain characters are used as designators and appear as themselves in lexical formats.

- T -- is used as time designator to indicate the start of the representation of the time of day in recurringDuration and timeInstant
- Z -- is used as time-zone designator, immediately (without a space) following a data element expressing the time of day in Coordinated Universal Time (UTC) in recurringDuration, timeInstant and time

D.2 Truncated and Reduced Formats

[ISO 8601] supports a variety of "truncated" formats in which some of the characters on the left of specific formats, for example, the century, can be omitted. Truncated formats are, in general, not permitted for the datatypes defined in this specification with three exceptions. The time datatype uses a truncated format for timeInstant. By truncating the date information we represent an instant of time that recurs every day. Similarly, the recurringDate and recurringDay datatypes use left-truncated formats for date.

[ISO 8601] also supports a variety of "reduced" or right-truncated formats in which some of the characters to the right of specific formats, such as the time specification, can be omitted. Right truncated formats are also, in general, not permitted for the datatypes defined in this specification

with the following exceptions: right-truncated representations of timePeriod are used as lexical representations for date, month, year and century.

D.3 Deviations from ISO 8601 Formats

D.3.1 Sign Allowed

An optional minus sign is allowed immediately preceding, without a space, the lexical representations for timeInstant and date.

D.3.2 No Year Zero

The year "0000" is an illegal year value.

D.3.3 More Than 9999 Years

To accommodate year values greater than 9999, more than four digits are allowed in the year representations of timeInstant, timeInstant and time. This follows the [ISO 8601 Draft Revision].

E Regular Expressions

A regular expression *R* is a sequence of characters that denote a **set of strings** *L(R)*. When used to constrain a lexical space, a **regular expression** *R* asserts that only strings in *L(R)* are valid literals for values of that type.

[Definition:] A **regular expression** is composed from zero or more branches, separated by | characters.

For all branches *S*, and for all regular expressions *T*, valid regular expressions *R* are:	Denoting the set of strings *L(R)* containing:
(empty string)	the set containing just the empty string
S	all strings in *L(S)*
S|*T*	all strings in *L(S)* and all strings in *L(T)*

[Definition:] A **branch** consists of zero or more pieces, concatenated together.

For all pieces *S*, and for all branches *T*, valid branches *R* are:	Denoting the set of strings *L(R)* containing:
S	all strings in *L(S)*
ST	all strings *st* with *s* in *L(S)* and *t* in *L(T)*

[Definition:] A **piece** is an atom, possibly followed by a quantifier.

For all atoms *S* and non-negative integers *n*, *m* such that *n* <= *m*, valid pieces *R* are:	Denoting the set of strings *L(R)* containing:
S	all strings in *L(S)*
S?	the empty string, and all strings in *L(S)*.
*S**	All strings in *L(S?)* and all strings *st* with *s* in *L(S*)* and *t* in *L(S)*. *(all concatenations of zero or more strings from L(S))*
S+	All strings *st* with *s* in *L(S)* and *t* in *L(S*)*. *(all concatenations of one or more strings from L(S))*
S{n,m}	All strings *st* with *s* in *L(S)* and *t* in *L(S{n-1,m-1})*. *(All sequences of at least n, and at most m, strings from L(S))*
S{0,m}	All strings *st* with *s* in *L(S?)* and *t* in *L(S{0,m-1})*. *(All sequences of at most m, strings from L(S))*
S{n,}	All strings in L(S{n,n}S*)
S{0,0}	The set containing only the empty string

NOTE: The regular expression language in the Perl Programming Language [Perl] does not include a quantifier of the form `S{,m}`, since it is logically equivalent to `S{0,m}`. We have, therefore, left this logical possibility out of the regular expression language defined by this specification. We welcome further input from implementors and schema authors on this issue.

[Definition:] A **quantifier** is one of `?`, `*`, `+`, `{n,m}` or `{n,}`, which have the meanings defined in the table above.

[Definition:] An **atom** is either a normal character, a character class, or a parenthesized regular expression.

For all normal characters *c*, character classes *C*, and regular expressions *S*, valid atoms *R* are:	Denoting the set of strings *L(R)* containing:
c	the single string consisting only of *c*
C	all strings in *L(C)*
(*S*)	all strings in *L(S)*

[Definition:] A **metacharacter** is either `.`, `\`, `?`, `*`, `+`, `{`, `}`, `(`, `)`, `[` or `]`. These characters have special meanings in regular expressions, but can be escaped to form atoms that denote the sets of strings containing only themselves, i.e., an escaped metacharacter behaves like a normal character.

[Definition:] A **normal character** is any XML character that is not a metacharacter. In regular expressions, a normal character is an atom that denotes the singleton set of strings containing only itself.

Note that a normal character can be represented either as itself, or with a character reference.

E.1 Character Classes

[Definition:] A **character class** is an atom *R* that identifies a **set of characters** *C(R)*. The set of

strings *L(R)* denoted by a character class *R* contains one single-character string "*c*" for each character *c* in *C(R)*.

A character class is either a character class escape or a character class expression.

[Definition:] A **character class expression** is a character group surrounded by `[` and `]` characters. For all character groups *G*, [*G*] is a valid **character class expression**, identifying the set of characters *C*([*G*]) = *C*(*G*).

[Definition:] A **character group** is either positive character group, a negative character group, or a character class subtraction.

[Definition:] A **positive character group** consists of one or more character ranges or character class escapes, concatenated together. A **positive character group** identifies the set of characters containing all of the characters in all of the sets identified by its constituent ranges or escapes.

For all character ranges ***R***, all character class escapes ***E***, and all positive character groups ***P***, valid positive character groups ***G*** are:	Identifying the set of characters *C(G)* containing:
R	all characters in *C(R)*.
E	all characters in *C(E)*.
RP	all characters in *C(R)* and all characters in *C(P)*.
EP	all characters in *C(E)* and all characters in *C(P)*.

[Definition:] A **negative character group** is a positive character group preceded by the `^` character. For all positive character groups *P*, *^P* is a valid **negative character group**, and *C(^P)* contains all XML characters that are **not** in *C(P)*.

[Definition:] A **character class subtraction** is a character class expression subtracted from a positive character group or negative character group, using the `-` character.

For any positive character group or negative character group *G*, and any character class expression *C*, *G-C* is a valid character class subtraction, identifying the set of all characters in *C(G)* that are not also in *C(C)*.

[Definition:] A **character range** *R* identifies a set of characters *C(R)* containing all XML characters with Unicode code points in a specified range.

A single XML character is a character range that identifies the set of characters containing only itself. All XML characters are valid character ranges, except as follows:

- The `[`, `]`, and `\` characters are not valid character ranges;
- The `^` character is only valid at the beginning of a positive character group if it is part of a negative character group; and
- The `-` character is a valid character range only at the beginning or end of a positive character group.

A character range may also be written in the form *s-e*, identifying the set that contains all XML

characters with Unicode code points greater than or equal to the code point of *s*, but not greater than the code point of *e*.

s-*e* is a valid character range iff:

- *s* is a single character escape, or an XML character;
- *s* is not `\`
- If s is the first character in a character class expression, then *s* is not `^`
- *e* is a single character escape, or an XML character;
- *e* is not `\` or `[`; and
- The code point of *e* is greater than or equal to the code point of *s*;

  NOTE: The code point of a single character escape is the code point of the single character in the set of characters that it identifies.

E.1.1 Character Class Escapes

[Definition:] A **character class escape** is a short sequence of characters that identifies predefined character class. The valid character class escapes include the single character escapes, the multi-character escapes, and the category escapes.

[Definition:] A **single character escape** identifies a set containing a only one character -- usually because that character is difficult or impossible to write directly into a regular expression.

The valid single character escapes are:	Identifying the set of characters *C(R)* containing:
`\n`	the newline character (
)
`\r`	the return character (
)
`\t`	the tab character ( )
`\\`	\
`|`	|
`\.`	.
`\-`	-
`\^`	^
`\?`	?
`\*`	*
`\+`	+
`\{`	{
`\}`	}
`\(`	(
`\)`	)
`\[`	[
`\]`	]

[Definition:] The Unicode Standard [Unicode] defines a number of character properties and provides mappings from code points to specific character properties. The set containing of all characters that have property `X`, may be identified with a **category escape** `\p{X}`. The complement of this set may be

specified with the **category escape** `\P{X}`. (`[\P{X}]` = `[^\p{X}]`).

NOTE: The syntax `\p{X}` is the same as that used in [Perl 5.6].

The following table specifies the main character properties (for more information, see Chapter 4 of [Unicode]).

Category	Property	Meaning
Letters	L	All Letters
	Lu	Uppercase
	Ll	Lowercase
	Lt	Titlecase
	Lm	Modifier
	Lo	Other
Marks	M	All Marks
	Mn	Non-Spacing
	Mc	Spacing Combining
	Me	Enclosing
Numbers	N	All Numbers
	Nd	Decimal Digit
	Nl	Letter
	No	Other
Punctuation	P	All Punctuation
	Pc	Connector
	Pd	Dash
	Ps	Open
	Pe	Close
	Pi	Initial quote (may behave like Ps or Pe depending on usage)
	Pf	Final quote (may behave like Ps or Pe depending on usage)
	Po	Other
Separators	Z	All Separators
	Zs	Space
	Zl	Line
	Zp	Paragraph
	S	All Symbols
	Sm	Math

Symbols	Sc	Currency
	Sk	Modifier
	So	Other
Other	C	All Others
	Cc	Control
	Cf	Format
	Cs	Surrogate
	Co	Private Use
	Cn	Not Assigned

[Definition:] The Unicode Standard [Unicode] groups code points into a number of blocks such as Basic Latin (i.e., ASCII), Latin-1 Supplement, Hangul Jamo, CJK Compatibility, etc. The set containing of all characters that have block name `X` (with all whitespace stripped out), may be identified with a **block escape** `\p{IsX}`. The compliment of this set may be specified with the **block escape** `\P{IsX}`. (`[\P{IsX}]` = `[^\p{IsX}]`).

NOTE: The syntax `\p{IsX}` is the same as that used in [Perl 5.6].

For example, the block escape for identifying the ASCII characters is `\p{IsBasicLatin}`.

[Definition:] A **multi-character escape** provides a simple way to identify a commonly used set of characters:

Character sequence	Equivalent character class
.	[^\n\r]
\s	[\t\n\r]
\S	[^\s]
\i	[\p{L}\p{Nl}:_]
\I	[^\i]
\c	the set of characters matched by NameChar
\C	[^\c]
\d	\p{Nd}
\D	[^\d]
\w	[&#x0000;-&#xFFFF;]-[\p{P}\p{S}\p{C}] (*all characters except the set of "punctuation", "separator" and "control" characters*)
\W	[^\w]

F References

F.1 Normative

Clinger, WD (1990)

William D Clinger. *How to Read Floating Point Numbers Accurately.* In *Proceedings of Conference on Programming Language Design and Implementation*, pages 92-101. Available at: ftp://ftp.ccs.neu.edu/pub/people/will/howtoread.ps

IEEE 754-1985

IEEE. *IEEE Standard for Binary Floating-Point Arithmetic.* See http://standards.ieee.org/reading/ieee/std_public/description/busarch/754-1985_desc.html

ISO 10646

ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology --- Universal Multiple-Octet Coded Character Set (UCS) --- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

Namespaces in XML

World Wide Web Consortium. *Namespaces in XML*. Available at: http://www.w3.org/TR/REC-xml-names/

RFC 1766

H. Alvestrand, ed. *RFC 1766: Tags for the Identification of Languages* 1995. Available at: http://www.ietf.org/rfc/rfc1766.txt

RFC 2045

N. Freed and N. Borenstein. *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. 1996 Available at: http://www.ietf.org/rfc/rfc2045.txt

RFC 2396

Tim Berners-Lee, et. al. *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax.*. 1998 Available at: http://www.ietf.org/rfc/rfc2396.txt

Unicode

The Unicode Consortium. *The Unicode Standard, Version 3.0.* Addison-Wesley, 2000. ISBN 0-201-61633-5.

Unicode Database

The Unicode Consortium. *The Unicode Character Database*. Available at: http://www.unicode.org/Public/3.0-Update/UnicodeCharacterDatabase-3.0.0.html

XML 1.0 Recommendation

World Wide Web Consortium. *Extensible Markup Language (XML) 1.0.* Available at: http://www.w3.org/TR/REC-xml

XML Information Set

World Wide Web Consortium. XML Information Set (public WD) Available at: http://www.w3.org/TR/xml-infoset

XML Schema Part 1: Structures

XML Schema Part 1: Structures. Available at: http://www.w3.org/TR/2000/WD-xmlschema-1-20000922/

XML Schema Requirements

XML Schema Requirements. Available at: http://www.w3.org/TR/NOTE-xml-schema-req

F.2 Non-normative

Gay, DM (1990)

David M. Gay. *Correctly Rounded Binary-Decimal and Decimal-Binary Conversions.* AT&T Bell Laboratories Numerical Analysis Manuscript 90-10, November 1990. Available at: http://cm.bell-labs.com/cm/cs/doc/90/4-10.ps.gz

ISO 11404

ISO (International Organization for Standardization). *Language-independent Datatypes.* See http://www.iso.ch/cate/d19346.html

ISO 8601

ISO (International Organization for Standardization). *Representations of dates and times, 1988-06-15.* Available at: http://www.iso.ch/markete/8601.pdf

ISO 8601 Draft Revision

ISO (International Organization for Standardization). *Representations of dates and times, draft revision, 1998.*

Perl

The Perl Programming Language. See http://www.perl.com

Perl 5.6

The Perl Programming Language, Version 5.6. See http://www.perl.com/language/misc/ann58/index.html

RDF Schema

World Wide Web Consortium. *RDF Schema Specification.* Available at: http://www.w3.org/TR/rdf-schema/

SQL

ISO (International Organization for Standardization). *ISO/IEC 9075-2:1999, Information technology --- Database languages --- SQL --- Part 2: Foundation (SQL/Foundation)* . [Geneva]: International Organization for Standardization, 1999. See http://www.iso.ch/cate/d26197.html

Unicode Regular Expression Guidelines

Mark Davis. *Unicode Regular Expression Guidelines*, 1988. Available at: http://www.unicode.org/unicode/reports/tr18/

XSL

World Wide Web Consortium. *Extensible Stylesheet Language (XSL).* Available at: http://www.w3.org/TR/xsl/

G Acknowledgments (non-normative)

The editors acknowledge the members of the XML Schema Working Group, the members of other W3C Working Groups, and industry experts in other forums who have contributed directly or indirectly to the process or content of creating this document. The Working Group is particularly grateful to Lotus Development Corp. and IBM for providing teleconferencing facilities.

The current members of the XML Schema Working Group are:

Jim Barnette, Defense Information Systems Agency (DISA); David Beech, Oracle Corp.; Paul V. Biron, Health Level Seven; Don Box, DevelopMentor; Allen Brown, Microsoft; Lee Buck, Extensibility; Charles E. Campbell, Informix; Peter Chen, Bootstrap Alliance and LSU; David Cleary, Progress Software; Dan Connolly, W3C (staff contact); Roger L. Costello, MITRE; Ugo Corda, Xerox; Andrew Eisenberg, Progress Software; David Ezell, Hewlett Packard Company; David Fallside, IBM; Matthew Fuchs, Commerce One; Andrew Goodchild, Distributed Systems Technology Centre (DSTC Pty Ltd); Paul Grosso, ArborText, Inc; Martin Gudgin, DevelopMentor; Dave Hollander, CommerceNet (co-chair); Mary Holstege, Calico Commerce; Jane Hunter, Distributed Systems Technology Centre (DSTC Pty Ltd); Rick Jelliffe, Academia Sinica; Andrew Layman, Microsoft; Dmitry Lenkov, Hewlett Packard Company; Eve Maler, Sun Microsystems; Ashok Malhotra, IBM; Murray Maloney, Commerce One; John McCarthy, Lawrence Berkeley National Laboratory; Noah Mendelsohn, Lotus Development Corporation; Don Mullen, Extensibility; Frank Olken, Lawrence Berkeley National Laboratory; Dave Peterson, Graphic Communications Association; Mark Reinhold, Sun Microsystems; Jonathan Robie, Software AG; John C. Schneider, MITRE; Lew Shannon, NCR; C. M. Sperberg-McQueen, W3C (co-chair); Bob Streich, Calico Commerce; Henry S. Thompson, University of Edinburgh; Matt Timmermans, Microstar; Jim Trezzo, Oracle Corp.; Steph Tryphonas, Microstar; Mark Tucker, Health Level Seven; Asir S. Vedamuthu, webMethods, Inc; Priscilla Walmsley, XMLSolutions; Norm Walsh, ArborText, Inc; Aki Yoshida, SAP AG

The XML Schema Working Group has benefited in its work from the participation and contributions of a number of people not currently members of the Working Group, including in particular those named below. Affiliations given are those current at the time of their work with the WG.

Paula Angerstein, Vignette Corporation; Gabe Beged-Dov, Rogue Wave Software; Greg Bumgardner, Rogue Wave Software; Dean Burson, Lotus Development Corporation; Rob Ellman, Calico Commerce; George Feinberg, Object Design; Charles Frankston, Microsoft; Ernesto Guerrieri, Inso; Michael Hyman, Microsoft; Renato Iannella, Distributed Systems Technology Centre (DSTC Pty Ltd); Dianne Kennedy, Graphic Communications Association; Janet Koenig, Sun Microsystems; Setrag Khoshafian, Technology Deployment International (TDI); Ara Kullukian, Technology Deployment International (TDI); Murata Makoto, Xerox; Chris Olds, Wall Data; Shriram Revankar, Xerox; William Shea, Merrill Lynch; Ralph Swick, W3C; Tony Stewart, Rivcom

H Revisions from Previous Draft

1. 2000-07-12: pvb: removed note from DTD/Schema for datatypes included in Appendices A&B which says they aren't normative but that they ones included in Appednices A&B are:-)
2. 2000-07-12: pvb: added \| as a single character escape in the regex language
3. 2000-07-12: pvb: changed all wording of the form "X is derived from Y by fixing the value of facet Z to a" to be "X is derived from Y by setting the value of facet Z to a", to avoid confusion (since we can't [yet] "fix" a facet value).
4. 2000-07-13: pvb: updated the status of this document section for internal point release
5. 2000-07-13: pvb: added note to section on order relations, to the effect that just because this spec doesn't say that a type is ordered doesn't mean that down-stream apps can't specify some order relation.
6. 2000-07-13: pvb: modified stylesheet to make "priority feedback" issues more prevalent
7. 200007-13: pvb: modified markup around PFI for decimal to take advantage of the new stylesheet template for PFIs
8. 2000-07-13: pvb: removed the order relation from string, and hence, the min/max facets
9. 2000-07-13: pvb: turned the <note> in decimal about wanting feedback about arbitrary precision into an <ednote role='pf'>, which displays specially with new stylesheet
10. 2000-07-14: pvb: fixed the stylesheet so that it put a space between the links "built-in" and "derived" in the auto-generated "Derived types" subsection of each type definition.
11. 2000-07-14: pvb: created a schema for has-facet and has-property used in the appinfo of type definitions in the schema for datatypes
12. 2000-07-14: pvb: modified stylesheet to generate the spec from the modified has-facet and has-property appinfo items
13. 2000-07-15 and 2000-07-16: pvb: my allergies had me in bed all day and couldn't get anything done
14. 2000-07-17: pvb: almost fixed the bugs introduced by the stylesheet modifications for has-facet and has-property. Appendix C still contains a few type names duplicated under some facets...I'll get that later.
15. 2000-07-18: AM: Fixed typos caught by Susan Lesch in her note to schema-comments of May 12.
16. 2000-07-18: AM: Changed line in date formats to say year 0 not allowed.
17. 2000-07-18: AM: Changed value space for decimal.
18. 2000-07-18: AM: Changed text for recurringDuration.
19. 2000-07-18: AM: Fixed typos in "time".
20. 2000-07-18: pvb: changed has-facet and has-property to hasFacet and hasProperty
21. 2000-07-18: pvb: changed definition of decimal again, to give separate defs of value space

without any facet being valued, with only precision and with only scale. This is intended to clarify what is and is not meant by precision and scale. Also fixed long standing typo in the equation for the value space of decimal: i x 10^n corrected to i x 10^-n.

22. 2000-08-07: pvb: finally found error in stylesheet which was causing XT to have a stackOverflow, preventing the release of this version.
23. 2000-08-15: pvb: added a fixed property to each facet component
24. 2000-08-15: pvb: removed redundant "if"s in many of the Validation Contributions in section 4
25. 2000-08-15: pvb: removed mention of string from the Validation Contributions of the order-related facets (min/max inc/exc) in section 4. This should have been done in a previous draft when string became unordered.
26. 2000-08-16: pvb: added fixed property to each facet component; added fixed attribute to each facet element. Possible problems with the XML repr for pattern and enumeration still to be worked out.
27. 2000-08-21: pvb: fixed schema dump file, so that stylesheet correctly formats the value attribute of all facets as being required.
28. 2000-08-21: pvb: fixed stylesheet so that "hex | base64" in the XML Rep for encoding no longer formated as "| hex | base64"...this also fixed a long standing bug in the stylesheet such surrounding properly formating of <choice> in content models
29. 2000-08-22: pvb: added union types
30. 2000-08-23: pvb: changed defn syntax to conform to union proposal, including changes to stylesheet to get autogenerated text from datatypes.xsd to format correctly
31. 2000-08-24: pvb: cleaned up a few sections so that they are consistent with the (now) 3 forms of derivation (where there used to be only 2)
32. 2000-08-24: pvb: marked app B (DTD) as non-normative
33. 2000-08-30: AM: added definition of canonical form as 2.4.
34. 2000-08-30: AM: added canonical forms for all built-in datatypes.
35. 2000-08-30: AM: changed lex space for boolean to {true, false}.
36. 2000-08-31: AM: removed fixed property from pattern and enumeration pending resolution of how to handle these two cases.
37. 2000-08-31: AM: fixed syntax for examples. Added "fixed" for 2 examples.
38. 2000-08-31: AM: removed pattern facet from binary.
39. 2000-08-31: AM: changed value space for timeDuration. Some bug fixes to Appendix D.