Lecture 6 - XSD

XML Validation

- An XML document with correct syntax is called **Well Formed**. As described in the previous lecture:
  - XML documents must have a root element
  - XML elements must have a closing tag
  - XML tags are case sensitive
  - XML elements must be properly nested
  - XML attribute values must be quoted
- A **Valid** XML document must be "Well Formed". In addition, it must conform to a document type definition:
  - *DTD* - The original Document Type Definition
  - *XSD* - XML Schema Definition, an XML-based alternative to DTD
- DTD and XSD define the rules and the legal elements and attributes for an XML document
- DTD and XSD can be used to validate XML documents

DTD Example

Valid XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE note SYSTEM "Note.dtd">
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Note.dtd file:

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

XSD Example

Valid XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>

<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="note.xsd">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Fragment of note.xsd:

```xml
<element name="note">
  <complexType>
    <sequence>
      <element name="to" type="string"/>
      <element name="from" type="string"/>
      <element name="heading" type="string"/>
      <element name="body" type="string"/>
    </sequence>
  </complexType>
</element>
```

XML Schema Definition

- Standard released by W3C in May 2001
- XML Schema model is an XML document
  - It must be enclosed by a root element named schema
    - Any number and combination of inclusions
    - Any number and combination of imports
    - Any number and combination of re-definitions
    - Any number and combination of annotations
    - Any number and combination of simple and complex data type definitions
    - Any number and combination of element, attributes and model group definitions
- XML Schemas are more powerful than DTD
- XML Schemas support Data Types
- XML Schemas support namespaces
- XML Schemas use XML syntax

XML Schema

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <!-- ANY NUMBER OF FOLLOWING -->

  <include ... />
  <import> ... </import>
  <redefine> ... </redefine>
  <annotation> ... </annotation>

  <!-- ANY NUMBER OF FOLLOWING DEFINITIONS -->

  <simpleType> ... </simpleType>
  <complexType> ... </complexType>
  <element> ... </element>
  <attribute />
  <attributeGroup> ... </attributeGroup>
  <group> ... </group>
  <annotation> ... </annotation>
</schema>
```

The `<schema>` Element

- The `<schema>` element is the root element of every XML Schema
- XML Schema standard belongs to the http://www.w3.org/2001/XMLSchema namespace

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  ...
</schema>
```

or with not-mandatory attributes

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.example.org"
           xmlns="http://www.example.org"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
  ...
</xs:schema>
```

- A single XML Schema defines a new namespace (targetNamespace) to be used in other XML documents
- Element names and attribute names within a particular element must be unambiguous
- The form *qualified* means both global and local elements or attributes are in the targetNamespace
  - All elements and attributes from the targetNamespace must be qualified with the namespace prefix
- The form *unqualified* means that only global elements or attributes are in the targetNamespace (default)
  - Only global elements or attributes from the targetNamespace must be qualified with the namespace prefix

Referencing a Schema in an XML Document

- A XML Schema can be referenced from an XML document by defining the schemaLocation from the XML Schema Instance namespace

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="example.xsd" />
```

or if the xml defines a namespace

```
<root xmlns="http://www.example.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.example.org example.xsd" />
```

- The schemaLocation attribute can contain a list of namespaces and schema-locations, separated by white-space.
- You can define multiple namespaces/schema-location pairs to validate elements and attributes in different namespaces.

```
<root xmlns="http://www.example.org"
      xmlns:ns2="http://www.example2.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.example.org example.xsd
                          http://www.example2.org example2.xsd" />
```

Elements

The **element** keyword is used to define an element.
The **name** attribute specifies the name of the element to be defined.
The **type** attribute specifies the data type of the element to be defined.

Definition:

```
<element name="book" type="string">
  ...
</element>
```

Usage:

```
<book>War and Peace</book>
```

Simple Elements may have a default value ...

```
<element name="color" type="string" default="red" />
```

or a fixed value

```
<element name="color" type="string" fixed="red" />
```

Attributes (1)

- Attributes are designed to contain data related to a specific element
- Created via attribute element
- Created and referenced like the elements
- The type attribute specifies the content
- Only elements of type complexType can have attributes, (Simple Elements does not have attributes)

```
<element name="chapter">
    <complexType>
        ...
        <attribute name="security" type="string" />
    </complexType>
</element>
```

Attributes (2)

- By default attributes are optional
- Behaviour modified via the *use* attribute: required, prohibited, optional (default)

```
<attribute name="security" type="string" use="required"/>
```

- Attribute can be declared with a *default* value specified with the default attribute and referenced like the elements

```
<attribute name="security" type="string" default="secret" />
```

- A value for the attribute that can never be changed can be specified via the *fixed* attribute

```
<attribute name="security" type="string" fixed="secret"/>
```

Simple and Complex Elements

Simple element: Element with no attributes, no sub-elements.
Generally, this means that the element is just a container for a simple type value, such as a number, word or text string.
The type attribute is used to specify what kind of data the element can hold.

Definition in XSD:

```
<element name="para" type="string" />
```

Usage in XML:

```
<para> This is a text string. </para>
```

Complex element: Anything more sophisticated than simple value content.
`<complexType>` is needed to specify the complex content model of the element being defined, and/or any attributes that can be attached to this element.

Definition in XSD:

```
<element name="book">
  <complexType>...</complexType>
</element>
```

Simple and Complex Content (1)

Simple Content: Text only, but *can have attributes*.

- Defined using `<simpleContent>` with `<extension>` or `<restriction>`.
- When using `<simpleContent>`, you select a simple data type and you must define an extension OR a restriction within the `<simpleContent>` element.
  - This rule applies also to `<complexContent>`, see example at Redefine slide

Definition in XSD:

```
<element name="price">
  <complexType>
    <simpleContent>
      <extension base="decimal">
        <attribute name="currency" type="string"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
```

Usage in XML:

```
<price currency="USD">10.00</price>
```

Note: An element with simple content can be a simple element or a complex element.

Simple and Complex Content (2)

Complex Content: Contains *child elements* (and optional attributes).

- Defined using `<sequence>`, `<choice>`, or `<all>`.

Definition in XSD:

```
<element name="book">
  <complexType>
    <sequence>
      <element name="title" type="string"/>
    </sequence>
  </complexType>
</element>
```

Usage in XML:

```
<book>
  <title>XML Guide</title>
</book>
```

Note: An element with complex content is always a complex element.

Complex Elements (1)

Empty element

Definition in XSD:

```
<element name="product">
  <complexType>
    <attribute name="prodid" type="positiveInteger"/>
  </complexType>
</element>
```

Usage in XML:

```
<product prodid="1345" />
```

Complex Elements (2)

Elements Only Content

Definition in XSD:

```
<element name="person">
  <complexType>
    <sequence>
      <element name="firstname" type="string"/>
      <element name="lastname" type="string"/>
    </sequence>
  </complexType>
</element>
```

Usage in XML:

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

Complex Elements (3)

Text Only Content with Attributes

Definition in XSD:

```
<element name="shoesize">
  <complexType>
    <simpleContent>
      <extension base="integer">
        <attribute name="country" type="string" />
      </extension>
    </simpleContent>
  </complexType>
</element>
```

Usage in XML:

```
<shoesize country="france">35</shoesize>
```

- Instead if you use a simple element `<element name="shoesize" type="integer" />` you cannot add attributes.

Complex Elements (4)

Mixed Content

Definition in XSD:

```
<element name="letter">
  <complexType mixed="true">
    <sequence>
      <element name="name" type="string"/>
      <element name="orderid" type="positiveInteger"/>
      <element name="shipdate" type="date"/>
    </sequence>
    <attribute name="language" type="string"/>
  </complexType>
</element>
```

Usage in XML:

```
<letter language="english">
  Dear Mr. <name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

Extension (1)

- The `<extension>` element defines extensions on a `<simpleContent>`, or `<complexContent>` definition.

Extension on `<simpleContent>` definition to add attributes:

```
<element name="shoesize">
  <complexType>
    <simpleContent>
      <extension base="integer">
        <attribute name="country" type="string" />
      </extension>
    </simpleContent>
  </complexType>
</element>
```

- Note: when you use `<extension>` you define always a new `<complexType>`.

Extension (2)

- The `<extension>` element defines extensions on a `<simpleContent>`, or `<complexContent>` definition.

Extension on `<complexContent>` definition to add elements and attributes:

```
<!-- Base Type: A generic Research Project -->
<complexType name="ProjectType">
  <sequence>
    <element name="title" type="string"/>
  </sequence>
</complexType>

<!-- Extended Type: A Funded Project adds a budget element and a grant attribute -->
<complexType name="FundedProjectType">
  <complexContent>
    <extension base="ProjectType">
      <sequence>
        <element name="budget" type="decimal"/>
      </sequence>
      <attribute name="grantID" type="string" use="required"/>
    </extension>
  </complexContent>
</complexType>

<element name="project" type="FundedProjectType" />
```

- Note: you can define `<simpleType>` and `<complexType>` globally (outside of specific element declarations) to create named types that can be reused across multiple elements throughout your schema, promoting modularity and consistency.

Restriction

- The `<restriction>` element defines restrictions on a `<simpleType>`, `<simpleContent>`, or `<complexContent>` definition.

Restriction on string definition:

```
<element name="initials">
  <simpleType>
    <restriction base="string">
      <pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </restriction>
  </simpleType>
</element>
```

Restriction on number definition:

```
<element name="age”>
  <simpleType>
    <restriction base="integer">
      <minInclusive value="0"/>
      <maxInclusive value="100"/>
    </restriction>
  </simpleType>
</element>
```

Simple Element Types

Simple Element has always a *simpleType*:

- no attributes
- no child elements

You have two ways to define a simple element:

1. Use a built-in simpleType

```
<element name="para" type="string" />
```

2. Restrict a simpleType

```
<element name="initials">
  <simpleType>
    <restriction base="string">
      <pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </restriction>
  </simpleType>
</element>
```

Complex Element Types

Complex Element has always a *complexType*:

You have two ways to define a complex element:

1. Extend a simpleContent with attributes

```
<element name="shoesize">
  <complexType>
    <simpleContent>
      <extension base="integer">
        <attribute name="country" type="string" />
      </extension>
    </simpleContent>
  </complexType>
</element>
```

2. Use any other complexType or complexContent: empty, mixed, child elements only

Element and Type References

Elements that contain other elements as children can do so by referring to elements that have already been defined elsewhere with the **ref** attribute

```
<element name="firstname" type="string"/>
<element name="lastname" type="string"/>

<element name="person">
  <complexType>
    <sequence>
      <element ref="firstname"/>
      <element ref="lastname"/>
    </sequence>
  </complexType>
</element>
```

Several elements can refer to the same complex type

```
<complexType name="persontype">
  <sequence>
    <element name="firstname" type="string"/>
    <element name="lastname" type="string"/>
  </sequence>
</complexType>

<element name="student" type="persontype"/>
<element name="professor" type="persontype"/>
```

Order Indicators (1)

The **sequence** indicator specifies that the child elements must appear in a specific order.

Sequence definition:

```
<element name="person">
  <complexType>
    <sequence>
      <element name="firstname" />
      <element name="lastname" />
    </sequence>
  </complexType>
</element>
```

Sequence usage:

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

Order Indicators (2)

The **all** indicator specifies that the child elements can appear in any order.

All definition:

```
<element name="person">
  <complexType>
    <all>
      <element name="firstname" />
      <element name="lastname" />
    </sequence>
  </complexType>
</element>
```

All usage:

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

or

```
<person>
  <lastname>Smith</lastname>
  <firstname>John</firstname>
</person>
```

Order Indicators (3)

The **choice** indicator specifies that either one child element or another can occur

Definition:

```
<element name="name">
    <complexType>
        <choice>
            <element name="personal" />
            <element name="company" />
        </choice>
    </complexType>
</element>
```

Usage:

```
<name>
    <personal>John Smith</personal>
</name>
```

or

```
<name>
    <company>Corp Ltd.</company>
</name>
```

Occurrence Indicators (1)

- By default sub-elements must be present, and cannot be repeated
- The **minOccurs** and **maxOccurs** attributes modify this behaviour
  - By default minOccurs and maxOccurs have value 1
- Example: first name is optional, middle name is both optional and repeatable, and last name is required but not repeatable

```
<sequence>
  <element name="first" minOccurs="0" maxOccurs="1" />
  <element name="middle" minOccurs="0" maxOccurs="unbounded" />
  <element name="last"  />
</sequence>
```

Occurrence Indicators (2)

- The possible values for MinOccurs and MaxOccurs attributes are

Common occurrence requirement	minOccurrs	maxOccurs
required	1	1
optional	0	1
required and repeatable	1	unbounded
optional and repeatable	0	unbounded

- Any value between 0 and unbounded is valid

```
<sequence>
  <element name="person" minOccurs="5" maxOccurs="100" />   <!-- at least 5 repetitions, 100 at most -->
  <element name="company" minOccurs="0" maxOccurs="5" />    <!-- optional value , 5 at most -->
</sequence>
```

Occurrence Indicators (3)

- Use occurrence indicators in a choice to select some of values or repeat some

Definition:

```
<element name="chapter">
  <complexType>
    <choice maxOccurs="unbounded">
      <element name="para"/>
         <element name="list"/>
         <element name="table"/>
    </choice>
  </complexType>
</element>
```

Use occurrence indicators in a sequence to repeat it

Definition:

```
  <element name="personalNames">
    <complexType>
      <sequence maxOccurs="unbounded">
           <element name="first"/>
           <element name="middle"/>
           <element name="last"/>
      </sequence>
    </complexType>
  </element>
```

Group Indicators

- Group indicators are used to define related sets of elements or attributes and reference them in another definition

Element Groups

```
<group name="persongroup">
  <sequence>
    <element name="firstname" type="string"/>
    <element name="lastname" type="string"/>
    <element name="birthday" type="date"/>
  </sequence>
</group>

<complexType name="personinfo">
  <sequence>
    <group ref="persongroup"/>
    <element name="country" type="string"/>
  </sequence>
</complexType>

<element name="person" type="personinfo"/>
```

Group Indicators

- Group indicators are used to define related sets of elements or attributes and reference them in another definition

Attribute Groups

```
<attributeGroup name="personattrgroup">
  <attribute name="firstname" type="string"/>
  <attribute name="lastname" type="string"/>
  <attribute name="birthday" type="date"/>
</attributeGroup>

<element name="person">
  <complexType>
    <attributeGroup ref="personattrgroup"/>
  </complexType>
</element>
```

Global Elements

- Global element declarations appear at the top level of the schema document, meaning that their parent must be schema
- Global element declarations can then be used in multiple complex types

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://example.org/prod"
           targetNamespace="http://example.org/prod">

  <xs:element name="name" type="xsd:string"/>
  <xs:element name="size" type="xsd:integer"/>

  <xs:complexType name="ProductType">
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="size" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

Local Elements

- Local element declarations appear entirely within a complex type definition.
- Local element declarations can only be used in that type definition, never referenced by other complex types or used in a substitution group

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://example.org/prod"
           targetNamespace="http://example.org/prod">

  <xs:complexType name="ProductType">
    <xs:sequence>
      <xs:element name="name" type="xsd:string"/>
      <xs:element name="size" type="xsd:integer" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Global vs Local Elements

Global elements:

- use the exact same element declaration in more than one complex type
- use the element declaration in a substitution group

Local elements:

- allow unqualified element-type names in the instance
- declare several element types with the same name but different data types or other properties

Comments

- Normal XML comments in an XML Schema document

```
<!-- normal XML comment in document -->
```

- The **annotation** element for comments

```
<annotation>
  <documentation source="...">
    This is documentation
  </documentation>
</annotation>
```

- Distinguish between comments aimed at humans and comments aimed at software processing (supplanting processing instructions).
- For human consumption, the Documentation element contains the actual comment text, and may also carry a source attribute, containing a URL reference
- A single `annotation` element may contain any number of `documentation`.

Including Models (1)

- Schemas can be composed of one or more schema documents
- Breaking up the schemas into multiple documents is useful for:
  - *Easier reuse*. Small and focused schema documents are more likely to be reused.
  - *Ease of maintenance*. Smaller schema documents are more readable and manageable.
  - *Reduced chance of name collisions*. If different namespaces are used for the different schema documents, name collisions are less likely.
  - *Access control*. Security can be managed per schema document, allowing more granular access control.
- Possible schema fragments
  - *Subject area*. A schema document per application or per database OR different schema document for each type of document.
  - *General/specific*. Base set of components that can be extended for a variety of different purposes. For example, a schema document that contains generic (possibly abstract) definitions and separate schema documents for specific extensions.
  - *Basic/advanced*. A core set of components that are used in all instances, plus a number of optional components defined in a separate schema document. This allows an instance to be validated against just the core set of components, or the enhanced set, depending on the application.

Including Models (2)

The Include element references a schema file, using a URL reference in the SchemaLocation attribute, the contents of which are to be included in the main file:

```
<include schemaLocation="tables.xsd" />
```

- This element must occur before any element, attribute or complex definition
- The include elements may only appear at the top level of a schema
- The include elements must appear at the beginning
- The schemaLocation attribute indicates where the included schema document is located. Attribute is required, although the location is not required to be resolvable. If resolvable, it must be a complete schema
- When using includes, one of the following must be true:
  - Both schema documents have the same target namespace.
  - Neither schema document has a target namespace.
  - The including schema document has a target namespace, and the included schema document does not have a target namespace.

Example Include

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com/schema">

<xs:include schemaLocation="https://www.w3schools.com/xml/customer.xsd"/>
<xs:include schemaLocation="https://www.w3schools.com/xml/company.xsd"/>

..

..

..

</xs:schema>
```

Redefine

- The **redefine** is similar to include, with the additional option of specifying new definitions of some or all of the components in the redefined schema document.
  Only complex types, simple types, named model groups, and attribute groups can be redefined

Myschema1.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:complexType name="pname">
  <xs:sequence>
    <xs:element name="firstname"/>
    <xs:element name="lastname"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="pname"/>

</xs:schema>
```

Redefine

Myschema2.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:redefine schemaLocation="Myschema1.xsd">
  <xs:complexType name="pname">
    <xs:complexContent>
      <xs:extension base="pname">
        <xs:sequence>
          <xs:element name="country"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>

<xs:element name="author" type="pname"/>
</xs:schema>
```

Import

- **import** is used to tell the processor that you will be referring to components from other namespaces

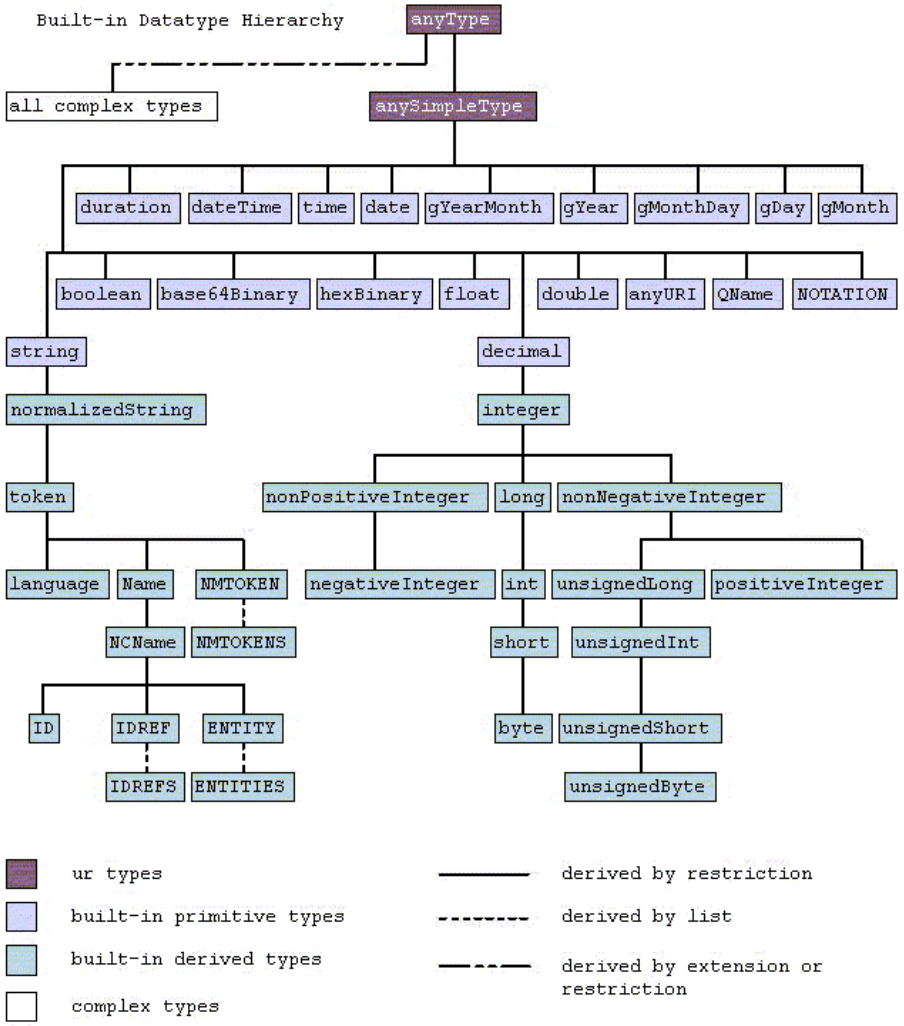
```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:import namespace="https://www.w3schools.com/schema"/>
    ...
</xs:schema>
```

- **includes** only takes place *within* a namespace, while **imports** take place *across* namespaces.
- The purpose of import is to record a dependency on another namespace, not necessarily another schema document. Import allows to specify the schema location document for that namespace, but it is just a hint, and the processor is not required to try to resolve it. (include is specifically to pull in other schema documents)

Built-in simple types

- There are 44 simple types built into XML Schema
- Most of the built-in types are atomic, although there are three list types (NMTOKENS, ENTITIES, and IDREFS).
- Built-in types have a hierarchy
- At the top of the hierarchy is anySimpleType, which is an unrestricted type that cannot actually be used in a schema.
- Types directly under anySimpleType, are known as primitive types, while the rest are derived built-in types.
- Primitive types represent basic data type concepts, and all other built-in types are either restrictions or lists of those types.
- New simple types must be derived from another type

Built-in simple types



String-based Types

- String-based types are used to represent a character string that may contain any Unicode character.
- Certain characters (e.g. “less than” symbol (<) and the ampersand (&)) must be escaped (using the entities < and &, respectively)
- Three string types, the only difference is in the way whitespace is handled by a schema-aware processor
  - **string**: preserve all whitespaces. Used for text where formatting (tabs, line breaks,..) is significant.
  - **normalizedString**: replaces each carriage return, line feed, and tab by a single space (equivalent to the processing of CDATA attribute values in XML 1.0). There is no collapsing of multiple consecutive spaces into a single space. Is used when formatting is not significant but consecutive whitespace characters are significant. This can be used when the information in the string is positional.
  - **token**: replaces each carriage return, line feed, and tab by a single space. Each group of consecutive spaces is collapsed into one space character, and all leading and trailing spaces are removed. (equivalent to the processing of non-CDATA attribute values in XML 1.0). Used for most short, atomic strings, especially ones that have an enumerated set of values. Basing your enumerated types on token means that `<gender>F </gender>` will be valid as well as `<gender>F</gender>`.

Numeric Types

- **float** represents an IEEE single-precision 32-bit floating-point number
- **double** represents an IEEE double-precision 64-bit floating-point number.
- Both float and double values is a mantissa followed, optionally, by the character “E” or “e” followed by an exponent. The exponent must be an integer. In addition, the following values are valid: INF (infinity), -INF (negative infinity), and NaN (Not a Number).
- **decimal** represents a decimal number of arbitrary precision. Leading and trailing zeros are permitted, but they are not considered significant.
- **integer** represents an arbitrarily large integer, from which twelve other built-in integer types are derived (directly or indirectly)

Date and Time Types

- XML Schema provides a number of built-in date and time data types, whose formats are based on ISO 8601.
  - **date** represents a Gregorian calendar date. The lexical representation of date is YYYY-MM-DD
  - **time** represents a time of day. The lexical representation of time is hh:mm:ss.sss
  - **dateTime** represents a specific date and time. The lexical representation of dateTime is YYYY-MM-DDThh:mm:ss.sss
  - **gYear** represents a specific Gregorian calendar year. The letter g at the beginning of most date and time types signifies “Gregorian.” The lexical representation of gYear is YYYY.
  - **gMonth** represents a specific month that recurs every year (such as the month of May). The lexical representation of gYearMonth is --MM
  - **gDay** represents a day that recurs every month (such as the third day of each month). The lexical representation of gDay is ---DD
  - **gYearMonth** represents a specific month of a specific year. The lexical representation of gYearMonth is YYYY-MM
  - **gMonthDay** represents a specific day that recurs every year (such as the 5th of July). The lexical representation of gMonthDay is --MM-DD
  - **duration** represents a duration of time expressed as a number of **Y**ears, **M**onths, **D**ays, **H**ours, **M**inutes, and **S**econds. The lexical representation of duration is **P**nYnMnD**T**nHnMnS, where P is a literal value that starts the expression, and, T is a literal value that separates the date and time (e.g. P1Y2M3DT5H2M3.123S, PT130S, PT2M10S, P2MT10S, -P1Y)

References and Exercises

- https://www.w3schools.com/xml/default.asp
- https://www.w3schools.com/xml/xml_dtd_intro.asp
- https://www.w3schools.com/xml/schema_intro.asp
- https://www.w3schools.com/xml/schema_complex.asp
- https://www.w3schools.com/xml/schema_dtypes_string.asp
- https://www.w3schools.com/xml/xpath_intro.asp
- https://docs.oracle.com/cd/B14099_19/integrate.1012/b14069/xsd.htm#sthref505
- Create a XML Schema based on http://www.w3.org/2001/XMLSchema
- Create a XML document using the schema
- Validate document with the schema (https://www.xmlvalidation.com/ or https://www.freeformatter.com/xml-validator-xsd.html)