# XML Concepts

## What is XML?

* XML stands for **extensible Markup Language**. It’s a markup language like HTML – it has open and closing tags with specific meanings.
* XML lets us define our own markup elements, it was conceived because of hardware and software in the past being unable to exchange data.

**Example**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <note>   <to>Tove</to>   <from>Jani</from>   <heading>Reminder</heading>   <body>Don't forget me this weekend!</body> </note> |

**Note:** The xml declaration at the top is optional

## XML Validator

XML files can be validated using an XML schema from on XSD file (another type of XML file)

An XML file is **well-formed** provided that:

* There is a root document. I.e., there is an enclosing tag around all elements
* Each open tag is closed and are case sensitive.
* Attribute values are quoted (single or double)
* Elements are properly nested (it may be illegal to have a specific nesting pattern)

## Where is XML used?

* Used to represent configuration files (e.g., pom.xml)
* Storing and manipulating data

## XML Processor

**What is an XML Processor?**

This is software which can read XML. It can store the XML’s data as in-memory structures which then can be accessed by a program. It checks for **validation and well-formedness**

**What is an XML Parser?**

This is also an XML processor which represents XML in a format which can be used by a programming language. E.g., Java has several XML parsers such as SAX, STAX etc

# XML Instances

## What is an XML instance?

* This is an **XML file with extension .xml** which contains data.
* In version 1.1 of XML recommendation, an XML declaration is required at the top of the XML file

## XML Declaration

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* standalone=*"no"*?> |

* The **encoding** **attribute** is a facet which determines the valid character set for the XML file.
  + It’s default value of UTF-8 but can also be set to UTF-16 which has a wider valid space of characters
* The **standalone attribute** is a facet which indicates if an external XML file is needed to process the XML. For example, we may have an XML schema for validation
  + Its default value is “no”

## Elements

* Elements of an XML file are components which are wrapped with an opening and closing tag
  + E.g., <Message>Hello World</Message>
* The name of elements is an XML non-colonised name (cannot contain a colon)
* Can start with either a letter or underscore (\_)
* Can contain letters, digits, hyphens (-), periods (.) and underscores (\_)
* Cannot contain special characters like whitespace, ?, \*, /, … etc

## Root Element

* All valid XML files have a root element
* This is the enclosing element tag of all child elements

## Child Elements

* We may have elements within other elements.
* E.g.

|  |
| --- |
| <ParentElement>  <Child1></Child1>  <Child2></Child2>  <Child3></Child3>  </ParentElement> |

## Attributes

* Attributes are also knowns as facets
* They provide additional data to an element
* We could use an attribute to provide a link to an entity

# XML Schemas

## What is an XML Schema?

* Schemas define the blueprint of an XML file, its an XML file itself but with extension .xsd
* If we wanted to restrict how an XML file is composed, we validate it against a schema.

## Purpose of Schemas

* Verification and validation of XML documents
* Control properties such as:
  + Attributes
  + Order of elements
  + Uniqueness of values
  + Restrict data values to enumeration, ranges, and patterns
* Schemas act as a contract for valid XML instances
* Aids XML processors to effectively process XML data

## Example

**Product.xsd:**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <xs:schema xmlns:xs=*"http://www.w3.org/2001/XMLSchema"*>  <xs:element name=*"product"* type=*"ProductType"* />  <xs:complexType name=*"ProductType"*>  <xs:sequence>  <xs:element name=*"number"* type=*"xs:integer"* />  <xs:element name=*"size"* type=*"SizeType"* />  </xs:sequence>  <xs:attribute name=*"effDate"* type=*"xs:date"* />  </xs:complexType>  <xs:simpleType name=*"SizeType"*>  <xs:restriction base=*"xs:integer"*>  <xs:minInclusive value=*"2"* />  <xs:maxInclusive value=*"18"* />  </xs:restriction>  </xs:simpleType>  </xs:schema> |

**Product.xml:**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <product effDate=*"2001-04-12"*>  <number>557</number>  <size>10</size>  </product> |

# XML Schema Definitions: Simple Types

## Simple and Complex types

* To get a grasp of how to define XML schemas we need to understand what exactly simple types and complex types are
* Elements of an XML instance can either be of simple or complex type

## Simple Type

* Have a no child elements and do not have attributes
* We can **distinguish simple types** further to

1. **Atomic Types**: values which are indivisible. E.g., <size>10</size>
2. **List Types**: whitespace separated atomic values. E.g., <size> 10 Large M </size>
3. **Union Types**: This is a type which has a valid space of two or more simple types. E.g., if we had a simple type with a range of 2-10 and another with S|R|L, then a simple type of these two combined would yield a union type

**Examples:**

|  |
| --- |
| <size>10</size>  <comment>Runs large.</comment>  <availableSizes>10 large 2</availableSizes> |

|  |
| --- |
| <size system=*"US-DRESS"*>10</size>  <comment>Runs <b>large</b>.</comment>  <availableSizes><size>10</size><size>2</size></availableSizes> |

## Complex Type

* A complex type may have child attributes and/or attributes
* We can distinguish Complex Types into four categories:

1. **Simple Content**: there are no child elements but the type allows for attributes

Example:

|  |
| --- |
| <size system=*"US-DRESS"*>10</size> |

1. **Empty Content**: No child elements and no text. Only has attribute

Example:

|  |
| --- |
| <picture src=*"servername/filename"*></picture> |

1. **Element-Only Content**: Child elements defined but no character data within parent element itself (only within child elements)

Example:

|  |
| --- |
| <product>  <number>557</number>  <name>Short-Sleeved Linen Blouse</name>  <size system=*"US-DRESS"*>10</size>  <color value=*"blue"* />  </product> |

1. **Mixed Content:** Child elements with intermingled characters in parent element

Example:

|  |
| --- |
| <desc>This is our <i>best-selling</i> shirt.  <b>Note: </b> runs <u>large</u>.</desc> |

* We have the ability to constrain our XML files to fit into these content models

## Built in Simple Types

* There are **49** built-in simple types in the XML schema recommendation.
* Things like strings, dates, numbers are represented by the simple types

**Summary**

|  |  |
| --- | --- |
| **Category** | **Built-in-types** |
| Strings and names | string, normalizedString, token, Name, NCName, QName, language |
| Numeric | float, double, decimal, integer, long, int, short, byte, positiveInteger, nonPositiveInteger, negativeInteger, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte |
| Date and Time | duration, dateTime, date, time, gYear, gYearMonth, gMonth, gMonthDay, gDay, 1.1 dayTimeDuration, 1.1 yearMonthDuration, 1.1 dateTimeStamp |
| XML DTD types | ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN,  NMTOKENS, NOTATION |
| Other | boolean, hexBinary, base64Binary, anyURI |

# Creating Simple Types

## Named and Anonymous Types

* A **named type** is a type which has a name and is defined globally:

|  |
| --- |
| <xs:element name=*"Dress"* type=*"tns:SizeType"* />  <xs:simpleType name=*"SizeType"*>  <xs:restriction base=*"xs:integer"*>  <xs:minInclusive value=*"2"* />  <xs:maxInclusive value=*"18"* />  </xs:restriction>  </xs:simpleType> |

**Named type instance:**

|  |
| --- |
| <tns:product xmlns:tns=*"shivkumar.org/named"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"shivkumar.org/named NamedSimpleType.xsd "*>  <Dress>2</Dress>  </tns:product> |

* An **anonymous simple type** is defined locally of the element using it

|  |
| --- |
| <element name=*"Dress"*>  <!-- Anonymous simple type: -->  <simpleType>  <restriction base=*"integer"*>  <minInclusive value=*"2"* />  <maxInclusive value=*"20"* />  </restriction>  </simpleType>  </element> |

**Anonymous type instance:**

|  |
| --- |
| <element name=*"dress"*>  <!-- Anonymous simple type: -->  <simpleType>  <restriction base=*"integer"*>  <minInclusive value=*"2"* />  <maxInclusive value=*"20"* />  </restriction>  </simpleType>  </element> |

## Simple Type Restrictions

* Every simple type is a restriction of another simple type – we are also able to extend simple types to other simple types
* You will typically see restrictions having a base type of an inbuilt type (e.g., strings, number…) but you can also have a user defined base type
* If we are using a user-defined simple type, we must be aware that the restriction MUST restrict the valid space and NOT extend it

**Example: Restriction with in-built base type**

|  |
| --- |
| <!-- Creating a named simple type for demonstration purposes: -->  <simpleType name=*"IntegerRestriction100"*>  <!-- Restriction on numerical types -->  <restriction base=*"integer"*>  <minInclusive value=*"0"* />  <maxInclusive value=*"100"* />  </restriction>  </simpleType>  <element name=*"numberRange"* type=*"tns:IntegerRestriction100"* /> |

* In this example we have a base which is of the inbuilt type “integer”
* So, our instance could be:

|  |
| --- |
| <tns:numberRange>20</tns:numberRange> |

**Example: Restriction with user-defined base type**

|  |
| --- |
| <element name=*"numberRangeRestriction"*>  <simpleType>  <!-- We can also use user-defined types for the base value -->  <restriction base=*"tns:IntegerRestriction100"*>  <!-- The restrictions must restrict the valid space of the base type -->  <maxExclusive value=*"20"*></maxExclusive>  <minExclusive value=*"1"*></minExclusive>  </restriction>  </simpleType>  </element> |

* If we set the value of maxExclusive to “120” for example, it would generate an error since 119 is outside the valid space of “tns:IntegerRestriction100”

**Example: Restrictions using Regex**

* We are also able to apply regular expression patterns as a restriction
* E.g., we can create a pattern so that only a 3 digit number followed by a hyphen is allowed

|  |
| --- |
| <element name=*"StringPattern"*>  <simpleType>  <restriction base=*"string"*>  <pattern value=*"\d{3}-[A-Z]{2}"* />  </restriction>  </simpleType>  </element> |

## Simple Type Summary

**Restriction Facets Summary**

|  |  |
| --- | --- |
| **Facet** | **Meaning** |
| **minExclusive** | Value must be greater than x |
| **minInclusive** | Value must be greater than or equal to x |
| **maxInclusive** | Value must be less than or equal to x |
| **minInclusive** | Value must greater than or equal to x |
| **length** | The length of the value must be equal to x |
| **minLength** | The length must be greater than or equal to x |
| **maxLength** | The length must be less than or equal to x |
| **totalDigits** | The number of significant digits must be less than or equal to x |
| **fractionDigits** | The number of fractional digits must be less than or equal to x |
| **whiteSpace** | The schema processor should either preserve, replace or collapse the whitespace |
| **enumeration** | x is one of the valid values |
| **pattern** | x is one of the regular expressions that the value may match |
| **explicitTimezone** | The time zone of date/time value is required, optional or prohibited |
| **assertion** | The value must conform to a constraint in the XPath expression |

## Example Use Case: Patient Information

* We need to create a schema file to hold patient information
* The current fields we need to collect are the patients name, age, gender, email, and phone.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <schema xmlns=*"http://www.w3.org/2001/XMLSchema"*  targetNamespace=*"http://www.shivkumar.org/Patient"*  xmlns:tns=*"http://www.shivkumar.org/Patient"*  elementFormDefault=*"qualified"*>  <!-- \* We shall create a schema to create xml files for patients  \* The patients have the following data fields: name, age, email,  gender, phone -->  <complexType name=*"Patient"*>  <sequence>  <element name=*"name"*>  <!-- Using an anonymous simple type: -->  <simpleType>  <restriction base=*"string"*>  <maxLength value=*"15"*></maxLength>  </restriction>  </simpleType>  </element>  <element name=*"age"* type=*"integer"* />  <element name=*"email"* type=*"string"* />  <element name=*"gender"* type=*"string"*/>  <element name=*"phone"* type=*"string"* />  </sequence>  </complexType>  </schema> |

# Creating Complex Types

## Complex Types Definition

* We’ve defined a complex as a type which has an attribute and/or child elements.
* Complex Types have 4 different content models: simple content (text and attributes), empty (no text but has attributes), element only (no intermingled text), mixed (intermingled text)
* We can deterministically define what model our complex types will fit into
* Using model groups like sequence, all, choice lets us define the ordering and occurrence of elements

## Choice Model Group

* The choice model group allows only one element to be defined in the instance.
* Having 0 or >1 will yield an error

### Example

|  |
| --- |
| <element name=*"Patient"* type=*"tns:PatientHealth"*></element>  <complexType name=*"PatientHealth"*>  <choice>  <element name=*"weightLbs"* type=*"decimal"*/>  <element name=*"WeightKg"* type=*"integer"*/>  </choice>  </complexType> |

|  |
| --- |
| <tns:Patient  xmlns:tns=*"http://www.example.org/ChoiceGroupExample"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.example.org/ChoiceGroupExample ChoiceGroup.xsd "*>  <tns:WeightKg>12</tns:WeightKg>  </tns:Patient> |

## Sequence Model Group

* We have already used this model group in our Patient use case example
* It simply indicates that the elements must appear in the same order as the elements in the sequence group
* We can use facets like minOccurs and MaxOccurs to make element optional or enable them to appear multiple times

### Example

|  |
| --- |
| <element name=*"Patient"* type=*"tns:sequenceGroup"*/>  <complexType name=*"sequenceGroup"*>  <sequence>  <element name=*"weight"* type=*"integer"*/>  <element name=*"height"* type=*"integer"*/>  <element name=*"age"* type=*"integer"*/>  <element name=*"disease"* type=*"string"* minOccurs=*"0"*  maxOccurs=*"unbounded"*/>  </sequence>  </complexType> |

|  |
| --- |
| <!-- The following element is optional  We can also include multiple diseases but they must appear together  I.e. we can not have a disease element at the top -->  <tns:disease>Cancer</tns:disease>  <tns:disease>Diabetes</tns:disease>  <tns:disease>Heart Disease</tns:disease> |