XML Namespaces

- XML Namespaces provide a method to avoid element name conflicts.

Example:

1)
```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

2)
```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

- XML namespaces uses keyword 'xmlns' to create namespace prefixes and assign corresponding URI, that uniquely identifies the namespace.

- Name conflicts in XML can easily be avoided using a name prefix.

Example:

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

XML DTD

- XML **D**ocument **T**ype **D**eclaration commonly known as DTD is a way to describe precisely the XML language. DTDs check the validity of, structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

Types

DTD can be classified on its declaration basis in the XML document, such as −

- Internal DTD
- External DTD

When a DTD is declared within the file it is called **Internal DTD** and if it is declared in a separate file it is called **External DTD**.

Features

Following are some important points that a DTD describes −

- the elements that can appear in an XML document.
- the order in which they can appear.
- optional and mandatory elements.
- element attributes and whether they are optional or mandatory.
- whether attributes can have default values.

Simple example for internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
```

```
<address>
  <name>lalitha</name>
  <company>GNITS</company>
  <phone>(040)2222222</phone>
</address>
```

Example for external DTD:

address.dtd

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Xmlfile.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>lalitha</name>
  <company>GNITS</company>
  <phone>(040)22222222</phone>
</address>
```

DTD - Components

This chapter will discuss about XML Components from DTD perspective. A DTD will basically contain declarations of the following XML components −

- Element
- Attributes
- Entities

DTD-ELEMENT

XML elements can be defined as building blocks of an XML document. Elements can behave as a container to hold text, elements, attributes, media objects or mix of all.

A DTD element is declared with an ELEMENT declaration. When an XML file is validated by DTD, parser initially checks for the root element and then the child elements are validated.

Syntax: `<!ELEMENT elementname (content)>`

Element Content Types

Content of elements declaration in a DTD can be categorized as below −

- Empty content
- Element content
- Mixed content
- Any content

➢ For empty elements, we declare in DTD as

<!ELEMENT ele-name EMPTY>

```
<?xml version = "1.0"?>

<!DOCTYPE hr[
   <!ELEMENT address EMPTY>
]>
<address />
```

- For elements which contain any combination of parsable data

<!ELEMENT ele-name ANY>

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

<!DOCTYPE address [
   <!ELEMENT address ANY>
]>

<address>
   Here's a bit of sample text
</address>
```

- For element content

<!ELEMENT elementname (child1, child2...)>

```
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
```

List of Operators

Operator	Syntax	Description	Example
+	<!ELEMENT element-name (child1+)>	It indicates that child element can occur one or more times inside parent element.	<!ELEMENT address (name+)>
*	<!ELEMENT element-name	It indicates that child element can occur zero or more times	<!ELEMENT address (name*)>

	(child1*)>	inside parent element.	
?	<!ELEMENT element-name (child1?)>	It indicates that child element can occur zero or one time inside parent element.	<!ELEMENT address (name?)>
,	<!ELEMENT element-name (child1, child2)>	It gives sequence of child elements separated by comma which must be included in the the element-name.	<!ELEMENT address (name, company)>
|	<!ELEMENT element-name (child1 | child2)>	It allows making choices in the child element.	<!ELEMENT address (name | company)>

NOTE: no operator refers that child element can occur exactly once.

DTD-ATTRIBUTES

- Attribute gives more information about an element or more precisely it defines a property of an element. An XML attribute is always in the form of a name-value pair. An element can have any number of unique attributes.
- Attribute declaration is very much similar to element declarations in many ways except one; instead of declaring allowable content for elements, you declare a list of allowable attributes for each element. These lists are called ATTLIST declaration.

Syntax: `<!ATTLIST element-name attribute-name attribute-type attribute-value>`

Example:

```
?xml version = "1.0"?>

<!DOCTYPE address [
   <!ELEMENT address ( name )>
   <!ELEMENT name ( #PCDATA )>
   <!ATTLIST name id CDATA #REQUIRED>
]>

<address>
```

```
   <name id = "123">Tanmay Patil</name>
</address>
```

Rules of Attribute Declaration

- All attributes used in an XML document must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration
- Attributes may only appear in start or empty tags.
- The keyword ATTLIST must be in upper case
- No duplicate attribute names will be allowed within the attribute list for a given element.

Attribute types:

- ID: value is unique id.
- IDREF : value is the ID of another element.
- IDREFS: value is list of other IDs.
- NMTOKEN: value is a valid xml name.
- NMTOKENS: value is the list of valid xml names.
- ENTITY: value is an entity.

Attribute Value Declaration

Within each attribute declaration, you must specify how the value will appear in the document. You can specify if an attribute −

- can have a default value
- can have a fixed value
- is required
- is implied

GNITS

Default Values

Syntax: `<!ATTLIST element-name attribute-name attribute-type "default-value">`

FIXED Values

Syntax: `<!ATTLIST element-name attribute-name attribute-type #FIXED "value" >`

REQUIRED values

Syntax: `<!ATTLIST element-name attribute-name attribute-type #REQUIRED>`

IMPLIED Values

If the attribute you are declaring has no default value, has no fixed value, and is not required, then you must declare that the attribute as *implied*.

Syntax: `<!ATTLIST element-name attribute-name attribute-type #IMPLIED>`

DTD-ENTITIES

Entities are used to define shortcuts to special characters within the XML documents.

Syntax: `<!ENTITY entity_name "entity_value">`

Example:

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

<!DOCTYPE address [
   <!ELEMENT address (#PCDATA)>
   <!ENTITY name "lalitha">
   <!ENTITY company "GNITS">
```

```
   <!ENTITY phone_no "(040)22222222">
]>

<address>
   &name;
   &company;
   &phone_no;
</address>
```

CDATA:

It is an attribute type contains Character data that are not parsed by XML parser, but it pass the data to the application without modification.

The predefined entities such as <, >, and & require typing and are generally difficult to read in the markup.

In such cases, CDATA can be used. By using CDATA , you are commanding the parser that the particular section of the document contains no markup and should be treated as regular text.

Let's take an example for CDATA:

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
<![CDATA[
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@xyz.com</email>
]]>
</employee>
```

Output:

<firstname>vimal</firstname><lastname>jaiswal</lastname><email>vimal@xyz.com</email>

PCDATA:

PCDATA stands for Parsed Character data. PCDATA is the text that will be parsed by a parser. Tags inside the PCDATA will be treated as markup and entities will be expanded.

In other words you can say that a parsed character data means the XML parser examine the data and ensure that it doesn't content entity if it contains that will be replaced.

Let's take an example:

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@xyz.com</email>
</employee>
```

Output: vimal jaiswal vimal@xyz.com

XML Schema

- XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

XML Schemas are More Powerful than DTD

- XML Schemas are written in XML
- XML Schemas are extensible to additions
- XML Schemas support data types
- XML Schemas support namespaces

XML Schema Struture

An XSD is an XML document and must follow all the syntax rules like any other XML document. It must be well-formed. XSDs also have to be valid against the rules defined in the "schema of schemas", which defines, among other things, the structure, elements and attributes of XSD.

Since every XSD is itself an XML document, it starts with an XML declaration. In addition, an XML schema definition has the following components:

- The schema element
- Element definitions
- Attributes definitions
- Annotations
- Type definitions

Syntax

You need to declare a schema in your XML document as follows:

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

Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
   <xs:complexType>
      <xs:sequence>
```

```
        <xs:element name="name" type="xs:string" />
        <xs:element name="company" type="xs:string" />
        <xs:element name="phone" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

As we saw in the XML - Elements chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

Definition Types

You can define XML schema elements in following ways:

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:element name="phone_number" type="xs:int" />
```

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
          <xs:element name="company" type="xs:string" />
      <xs:element name="phone" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
          <xs:element name="company" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Now let us use this type in our example as below:

```
<xs:element name="Address1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="address" type="AddressType" />
          <xs:element name="phone1" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Address2">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="address" type="AddressType" />
          <xs:element name="phone2" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for*Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have*name* and *type* property as shown below:

```
<xs:attribute name="x" type="y"/>
```

NOTE:

- Defining default values (either in element or attribute)

  <xs:element name=”ele-name” type=”data-type” default=”default-value”/>

  <xs:attribute name=”attr-name” type=”data-type” default=”default-value”/>

- Defining fixed value:

<xs:element name=”ele-name” type=”data-type” fixed=”fixed-value”/>

<xs:attribute name=”attr-name” type=”data-type” fixed=”fixed -value”/>

Restriction on values:

Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on xml elements are called “FACETS”. It limits the possible values that an element can hold.

Example:

```
<xs:element name=”age”>
    <xs: simpletype>
        <xs:restriction base=”xs:integer”>
        <xs:minInclusive value=”0”/>
        <xs:maxInclusive value=”25”/>
        </xs:restriction>
    </xs:simpletype>
</xs:element>
```

Example:

```
<xs:element name=”age”>
    <xs: simpletype>
        <xs:restriction base=”xs:integer”>
```

<xs:enumeration value=”0”/>

<xs:enumeration value=”25”/>

</xs:restriction>

</xs:simpletype>

</xs:element>

Same as maxLength, minLength, totalDigits and so on.

A Reference to an XML Schema

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>

<note
xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/xml note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- **strength** about XML Schemas is that they are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schemas with the XML DOM
- You can transform your Schemas with XSLT