|  |
| --- |
| | Praveen D |

|  |  |
| --- | --- |
| JSpider Basavangudi Bangalore | Web Services – Day 2 URI (URL & URN), XML, JSON, JAXB, Jackson |

Table of Contents

[URI, URL and URN 2](#_Toc499365051)

[Web URL (Web Uniform Resource Locator's) 3](#_Toc499365052)

[eXtensible Mark-up Language (XML) 4](#_Toc499365053)

[1. XML Structure 5](#_Toc499365054)

[2. Entity References 5](#_Toc499365055)

[3. PCDATA: Parsed Character Data 6](#_Toc499365056)

[4. CDATA: Character Data 6](#_Toc499365057)

[5. XML Elements 7](#_Toc499365058)

[6. XML Elements Naming Rules 7](#_Toc499365059)

[7. XML Attributes 8](#_Toc499365060)

[8. XML Elements v/s Attributes 8](#_Toc499365061)

[9. XML Schema's 8](#_Toc499365062)

[1) XML Document Type Definition (DTD) 9](#_Toc499365063)

[2) XML Schema Definition (XSD) 10](#_Toc499365064)

[10. Differences between DTD & XSD 11](#_Toc499365065)

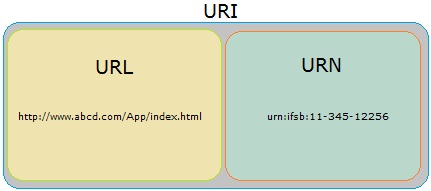
[11. Parsing XML Documents (XML Parsers) 11](#_Toc499365066)

# URI, URL and URN

* Uniform Resource Identifier (URI) is a string of characters used to
* identify a resource using name or
* Locate a resource in the network
* A URI identifies a resource either by location, or a name, or both. A URI has two specializations

1. URL (Uniform Resource Locator) and
2. URN (Uniform Resource Name)

* URN ONLY identifies the resource and does not let us know availability of the resource. A URN has to be of this form "urn:"
* URL that specifies where an identified resource is available and the mechanism for retrieving it. URL does not have to be HTTP URL (http://), a URL can also be (ftp://) or (smb://) or (jdbc:)
* For example,
* A URN is similar to a person's name, while
* A URL is like a street address.
* The URN defines something's identity, while the URL provides a location.
* Essentially, "what" vs. "where"
* To put it differently,
* A URL is a URI
* A URN is a URI
* but URNs and URLs are different, A URI is not necessarily a URL



* Few Examples:-

URL: <ftp://ftp.is.co.za/rfc/rfc1808.txt>

URL: <http://www.ietf.org/rfc/rfc2396.txt>

URL: <ldap://[2001:db8::7]/c=GB?objectClass?one>

URL: <mailto:John.Doe@example.com>

URL: <news:comp.infosystems.www.servers.unix>

URL: <telnet://192.0.2.16:80/>

URN (not URL): urn:oasis:names:specification:docbook:dtd:xml:4.1.2

URN (not URL): urn:isbn:0-486-27557-4

# Web URL (Web Uniform Resource Locator's)

* Web URL, uniquely identifies a particular web resource inside a web application
* In other words, every web resource should have its unique address in the form of Web URL
* Max. number of characters allowed in Web URL is around 2000 characters (exact number depends on Browser. For ex, IE supports 2048 characters)
* Web URL Structure:

<Protocol://Domain:Port/Path?QueryString#FragmentID>

* Protocol in case of Web URL is always http or https
* Domain Name uniquely identifies a computer in a network in which web application is present. It can be Computer Name/DNS Name (preferred) or IP address
* Port number in Web URL uniquely identifies web server application
* Default port number for HTTP is 80 & HTTPS is 443
* In Tomcat Webserver, default port number for HTTP is changed from 80 to 8080 and default port number for HTTPS is changed from 443 to 8443
* Path is the full path of the web resource at web application side.
* It consists of, Web Application Name / Configured URL of a Resource
* "Web Application Name" uniquely identifies One web application inside webserver
* "Configured URL" uniquely identifies web resource inside that web application
* Query String is a name & value string pair which passes information in the form of name=value pair to web resources. In URL, It’s an optional information and if present, it starts with question mark followed by one or more name=value pair which are separated by an ampersand(&)
* A Fragment ID or Fragment Identifier, as the name implies, it refers to a particular fragment / a section within a web page
* Matrix Parameters are a set of “name=value”. They can be present anywhere in URL (generally used with **path**) & URL can consist of N number of Matrix parameters but they should be separate by a semi colon “;“
* The important difference between Query Parameters & Matrix Parameters is that,
* Matrix Parameters apply to a particular path element while
* Query Parameters apply to the request as a whole
* This comes into play when making a complex REST-style query to multiple levels of resources and sub-resources
* **Note:-**
* Apart from Domain Name rest of the components of Web URL are Optional
* Few Examples:-

<http://www.google.com/search?q=Praveen>

<https://www.google.co.in/search?q=ABC&sitesearch=www.youtube.com>

<http://www.example.com/res/categories;name=foo/objects;name=green/?page=1>

# eXtensible Mark-up Language (XML)

* XML is "Programing Language & Platform Independent Language" which helps to store and transport data
* Different Applications which are developed using different technologies can Transfer the Data among themselves with the help of XML
* As the name implies it's an extension of HTML & hence XML looks similar to HTML but it’s not a HTML
* XML has User-defind Tags. XML tags are also called as "elements"
* XML Elements are "Case Sensitive"
* XML is "Strictly Typed" Language hence,
* For every element data, “data-type” should be defined,
* every opening element should have corresponding closing element and
* also XML elements must be properly nested/closed

**Ex:**

<employee>

<name>Praveen</name>

</employee>

**Note:-**

In the above example first you should closed </name> & then </employee> but in HTML it’s not mandatory. For example, <B><U><I>My Text</U></I></B> works perfectly fine

* Below line is called as "XML prolog", which is optional. If it exists, it must be the First Line of XML

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

* The syntax of XML comment is similar to that of HTML

<!-- This is a comment -->

* File extension of XML is ".xml"
* MIME type (Content Type) of XML is "application/xml"

## XML Structure

* Like HTML, XML follows a Tree Structure
* An XML tree starts at a "root element" and branches from "root element" will have "child elements"
* XML Consists of "Only One" root element which is parent of all other elements
* "child elements" can have "sub elements / child elements"
* Structure

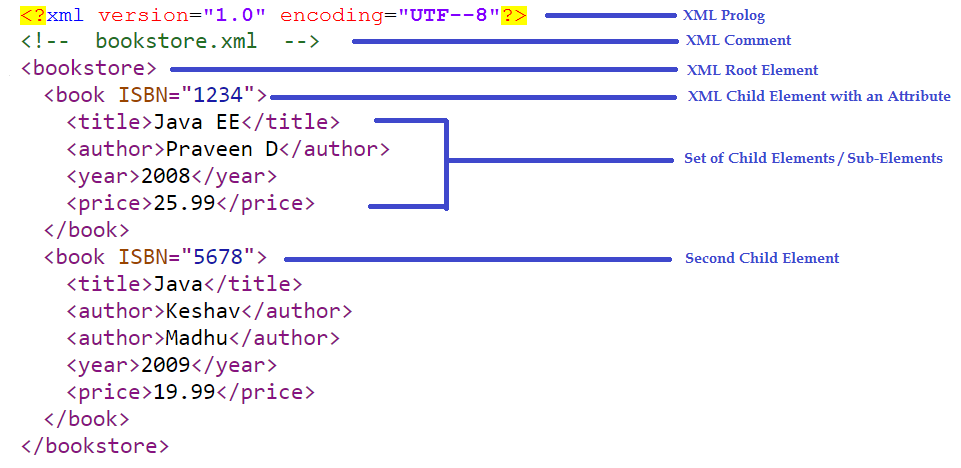
<root>

<child>

<subchild>.....</subchild>

</child>

</root>



## Entity References

* Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because it represents the start of a new element

**Ex:** <message>salary<1000</message>

* To avoid this error, we can replace the "<" character with an "entity reference" as shown below

<message>salary **&lt;** 1000</message>

* There are 5 pre-defined entity references in XML:

&lt; < less than

&gt; > greater than

&amp; & ampersand

&apos; ' apostrophe

&quot; " quotation mark

## PCDATA: Parsed Character Data

* Text between start-element and end-element is called as PCDATA which will be examined by the parser

**Example:-**

<employee>Praveen</employee>

The string "Praveen" is considered as PCDATA

## CDATA: Character Data

* W.K.T special characters (such as "<", "&") must be referenced through pre-defined entities
* If XML data contain many special characters, it is cumbersome to replace all of them. Instead we can use "CDATA (character data) section"

* A CDATA section starts with the following sequence:

<![CDATA[

and ends with the next occurrence of the sequence:

]]>

All characters enclosed between these two sequences are interpreted as characters

* The XML parsers ignores all the mark-up within the CDATA section.

**Example: -**

<employee>Praveen</employee>

the start and end "employee" elements are interpreted as mark-up. However, if written like this:

<![CDATA[ <employee>Praveen</employee> ]]>

then the parsers interprets the same as if it had been written like this:

**&lt;**employee**&gt;**Praveen**&lt;**/employee**&gt;**

## XML Elements

* XML element is everything from (including) the element's start tag to (including) the element's end tag

* An element can contain:

1. data

2. Attributes

3. other elements OR

4. All of the above

* In the above example
* <title>, <author>, <year>, and <price> have text content
* <bookstore> and <book> have element contents
* <book> has an attribute (ISBN="------")
* An element with no content is said to be "empty". In XML, we can indicate an empty element like this

<element></element>

OR

<element />

* Empty elements can have attributes <book ISBN="5678" />
* If data present between elements consist of white spaces then they are considered in XML. However HTML truncates multiple white-spaces to one single white-space

## XML Elements Naming Rules

* they are case-sensitive
* they cannot contain spaces
* they must start with a letter or underscore
* they are cannot start with the letters like xml or XML or Xml etc.,
* they can contain letters, digits, hyphens, underscores, and periods
* Any name can be used, no words are reserved (except xml)

**Best Naming Practices**

* Avoid "." and ":"
* Create descriptive names, like

<person>, <firstname>, <lastname>

* Create short and simple names, like

<book\_title> not like this: <the\_title\_of\_the\_book>

* Non-English letters are perfectly legal in XML but avoid them

## XML Attributes

* Like HTML, XML elements can also have attributes
* Attributes are designed to contain data related to a specific element
* XML Attributes Must be Quoted either single or double quotes can be used

**Ex:**

<person gender="female">

OR

<person gender='female'>

* If the attribute value itself contains double quotes then we can use single quotes

**Ex:**

<person name='Praveen "Bangalore" D'>

OR

<person name='Praveen **&quot;**Bangalore**&quot;** D'>

## XML Elements v/s Attributes

**Example 1:-**

<person gender="male">

<name>Praveen</name>

</person>

**Example 2:-**

<person>

<gender>male</gender>

<name>Praveen</name>

</person>

**Note:**

* In Example 1 gender is an attribute &
* In Example 2 gender is an element
* Both examples provide the same information
* There are no rules about when to use attributes or when to use elements in XML

**When to avoid XML Attributes?**

* Attributes cannot contain multiple values but Elements can
* Attributes cannot contain tree structures but Elements can
* Attributes are not easily expandable for future changes but Elements can

## XML Schema's

* W.K.T XML helps us to store & transfer the data
* When sending data from one application to an another, it is essential that both applications have the same "expectations / agreement" about the content/data
* for example, A date like "03-11-2004"
* in some countries, be interpreted as 3rd November and
* in other countries as 11th March

* With XML Schemas, the sender application can describe the data in a way that the receiver application will understand
* Schema is nothing but a "Structure". It is a formal description of structure of an XML.
* i.e., which elements are allowed,
* which elements must be present,
* which elements are optional,
* the sequence and relationship of the elements, etc.,
* For example,
* abc@gmail.com is a Valid Email ID. However
* abc#gmail is Invalid because there is "NO @ and ."
* hence email schema looks something like some-name@domain-name.com
* Schema "does not validate the data" instead "it validates the structure"
* There are two ways to define a Schema for XML

1. Document Type Definition (DTD)
2. XML Schema Definition (XSD)

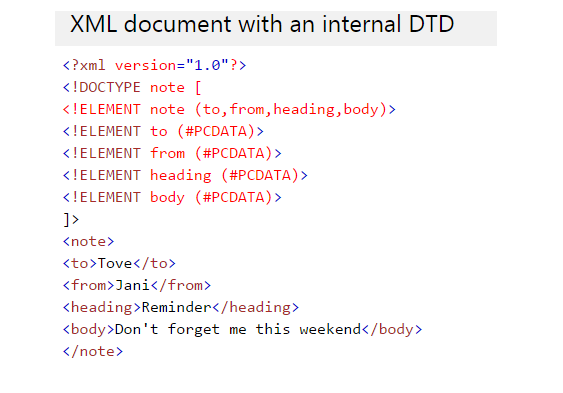
### XML Document Type Definition (DTD)

* + A DTD defines the structure and the legal elements and attributes of an XML document
  + An application can use a DTD to verify that XML data is valid
  + There are 2 ways to declare the DTD
  1. An Internal DTD Declaration
  2. An External DTD Declaration
* An Internal DTD Declaration has the following syntax:

<!DOCTYPE root-element [

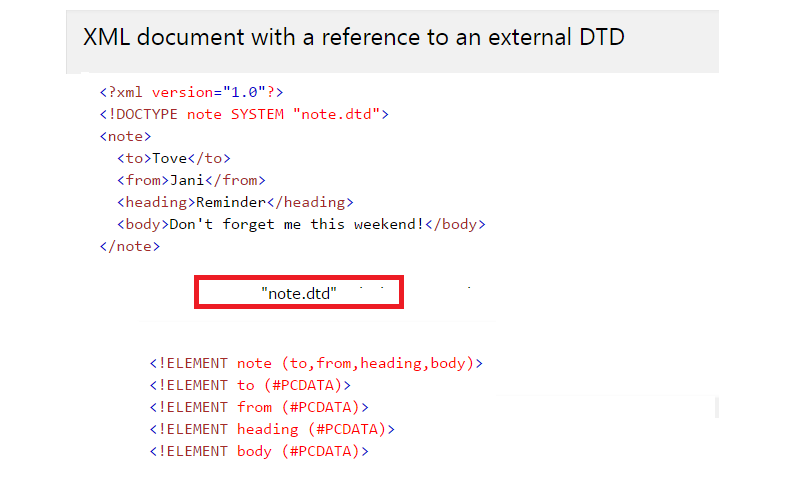
declarations

]>



* A DTD can also be stored in an external file. An XML can reference an external DTD via the following syntax:

<!DOCTYPE root-element SYSTEM "DTD-filename">



### XML Schema Definition (XSD)

* + XSD also describes the structure, legal elements and attributes for an XML
  + It defines,
* the elements and attributes that can appear in XML
* the number of and also the order of child elements
* data types for elements and attributes
* default and fixed values for elements & attributes
* One of the greatest strength of XML Schemas is the support for data types
* For Example, the following is an example of a date declaration in XSD:

<xs:element name="start-date" type="xs:date"/>

it defines the structure/format of the Date as "YYYY-MM-DD"

An element in XML might look like <start-date>2002-09-24</start-date>

* Another great strength about XML Schemas is that they are written in XML

* Hence XSD's are extensible so, we can
* Reuse Schema in other Schemas
* Create your own data types derived from the standard types
* Reference multiple schemas in the same document

**NOTE:**

* + Functionality wise both XSD & DTD similar in nature but XSD's are more sophisticated compared to DTD
  + In other words, DTD provides less control on XML structure whereas XSD provides more control
  + Hence XSD's preferred over DTD's
  + Without an XSD/DTD, an XML need only follow the rules for being well-formed
  + With an XSD/DTD, an XML must adhere to additional constraints placed upon the names and values of its elements and attributes in order to be considered valid

## Differences between DTD & XSD

|  |  |
| --- | --- |
| **DTD** | **XSD** |
| DTD’s are written in Mark-up Language | XSD’s are written in XML |
| DTD is not extensible i.e. We cannot inherit one DTD into an another | XSD is extensible. We can inherit one XSD into an another |
| DTD doesn't support data types (limited to string) | XSD supports data types for elements and attributes |
| DTD doesn't define order for child elements | XSD defines order for child elements |
| DTD's occurrence indicator is limited to 0, 1 and many; cannot support a specific number such as 8 | XSD can support a specific number |
| DTD doesn't support namespace | XSD supports namespace |
| We cannot inherit one DTD into an another | We can inherit one XSD into an another |
| DTD provides less control on XML structure | XSD provides more control on XML structure |

## Parsing XML Documents (XML Parsers)

* To process the data contained in XML documents, we need to write a application program (in any programming language such as Java/C/C++, etc)
  + - The program makes use of an XML parser to tokenize and retrieve the data from the XML documents
    - An XML parser is the software that sits between the application and the XML documents to shield the application developer from the details of the XML syntax.
    - The parser reads a raw XML document, ensures that is well-formed, and may validate the document against a DTD or XSD
    - There are two standard APIs for parsing XML documents:
  1. SAX (Simple API for XML)
  2. DOM (Document Object Model)
     + The JAXB provides a common interface for creating, parsing and manipulating XML documents using the standard SAX, DOM and XSLTs