H2K
Live - Green - Everywhere
H2K Infosys, LLC
Dream, Strive & Achieve Victory
H2K Infosys, LLC
Dream, Strive & Achieve Victory
H2K Infosys, LLC
Dream, Strive & Achieve Victory

www.H2KINFOSYS.com
USA: 770-777-1269
Training@H2KInfosys.com
UK : (020) 3371 7615

Parsing XML

Understanding XML:

W3 School defines XML as below:

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store, and transport information.

- XML Separates Data from HTML
- XML Simplifies Data Sharing
- XML Simplifies Data Transport
- XML Simplifies Platform Changes
- XML Makes Your Data More Available

Tree Structure:

XML documents must contain a **root** element. This element is "the parent" of all other elements. Elements can have attributes.

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

For Example:

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>

  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Attribute Values Must be quoted

Entity References

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Avoid XML Attributes?

Some of the problems with using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Attributes are difficult to read and maintain. Use elements for data. Use attributes for information that is not relevant to the data.

What is DTD?

W3 School defines DTD as “A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes”

A DTD can be declared inline inside an XML document, or as an external reference.

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

Example XML document with an internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
```

```
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body></note>
```

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element contains four elements: "to,from,heading,body"
- **!ELEMENT to** defines the 'to' element to be of type "#PCDATA" and so on.

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

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

Why Use a DTD?

With a DTD, each of your XML files can carry a description of its own format.

With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.

Your application can use a standard DTD to verify that the data you receive from the outside world is valid.

You can also use a DTD to verify your own data.

PCDATA: PCDATA means parsed character data. Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that **WILL be parsed by a parser**. The text will be examined by the parser for entities and markup.

CDATA: CDATA means character data.

CDATA is text that **WILL NOT be parsed by a parser**.

Declaring Elements

Elements	Declaration
Elements	<!ELEMENT element-name (element-content)>
Empty Elements	<!ELEMENT element-name EMPTY>
With PCDATA	<!ELEMENT from (#PCDATA)>
With any Contents	<!ELEMENT note ANY>
With Children (sequences)	<!ELEMENT note (to,from,heading,body)>
Only One Occurrence	<!ELEMENT note (message)>
Minimum One Occurrence	<!ELEMENT note (message+)>
Zero or More Occurrences	<!ELEMENT note (message*)>

Zero or One Occurrences	<!ELEMENT note (message?)>
Either/or Content	<!ELEMENT note (to,from,header,(message|body))>
Mixed Content	<!ELEMENT note (#PCDATA|to|from|header|message)*>

Declaring Attributes

<!ATTLIST element-name attribute-name attribute-type default-value>

E.g.: <!**ATTLIST** payment type CDATA "check">

The **attribute-value** can be one of the following:

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

The **default-value** can be one of the following:

Value	**Explanation**
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is not required
#FIXED value	The attribute value is fixed

Declaring Entity

Entities are variables used to define shortcuts to standard text or special characters.

- Entity references are references to entities
- Entities can be declared internal or external

Syntax: <!ENTITY entity-name "entity-value">
Example:

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright H2KInfosys">

<author>&writer;&copyright;</author>
```

What is XSD?

W3 School mentions the purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

XML Schemas are the Successors of DTDs

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

For discussed XML above:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XSD will look like:

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

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

</xs:schema>
```

The <schema> element is the root element of every XML Schema

xmlns:xs – Defines a namespace required for XSD
targetNamespace – mentions that the nodes defined in this XSD are for yoursite.com
xmlns – mentions default namespace
elementFormDefault - any elements used by the XML instance document which were declared in this schema must be namespace qualified.

Referencing a Schema

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

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Simple Element

The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

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

Attribute:

```
<xs:attribute name="xxx" type="yyy"/>
```

Default and Fixed Values

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

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

<xs:attribute name="lang" type="xs:string" default="EN"/>

<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Required Attributes

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Restrictions for Datatypes

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Examples:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="replace"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Parsing XML With Xstream:

XStream is a simple library to serialize objects to XML and back again. You can download Xstream libraries from here: http://xstream.codehaus.org/download.html

Features

- **Ease of use.** A high level facade is supplied that simplifies common use cases.
- **No mappings required.** Most objects can be serialized without need for specifying mappings.
- **Performance.** Speed and low memory footprint are a crucial part of the design, making it suitable for large object graphs or systems with high message throughput.
- **Clean XML.** No information is duplicated that can be obtained via reflection. This results in XML that is easier to read for humans and more compact than native Java serialization.
- **Requires no modifications to objects.** Serializes internal fields, including private and final. Supports non-public and inner classes. Classes are not required to have default constructor.
- **Full object graph support.** Duplicate references encountered in the object-model will be maintained. Supports circular references.
- **Integrates with other XML APIs.** By implementing an interface, XStream can serialize directly to/from any tree structure (not just XML).
- **Customizable conversion strategies.** Strategies can be registered allowing customization of how particular types are represented as XML.
- **Error messages.** When an exception occurs due to malformed XML, detailed diagnostics are provided to help isolate and fix the problem.
- **Alternative output format.** The modular design allows other output formats. XStream ships currently with JSON support and morphing.

Let's Understand Through Example:

1. Create a Transformer class, which handles two way transformations.
2. Create a DTO which needs to be transformed to XML
3. Call the transforming method.

```java
import com.thoughtworks.xstream.XStream;

public class XMLTransformer {

        private static XStream xstream = new XStream();

        public static String toXML(Object obj) {
                String xml = null;
                try {
                  xml = xstream.toXML(obj);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return xml;
        }
        public static Object fromXML(String xml) {
                Object obj = null;
                try {
                  obj = xstream.fromXML(xml);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return obj;
        }
}

public class ProjectDetailsTO {
        private int projectId ;
        private String  projectName;

        public int getProjectId() {
                return projectId;
        }
        public void setProjectId(int projectId) {
                this.projectId = projectId;
        }
        public String getProjectName() {
                return projectName;
        }
        public void setProjectName(String projectName) {
                this.projectName = projectName;
        }
}

public class XstreamTest {
         public static void main(String[] args) {
                 ProjectDetailsTO to = new ProjectDetailsTO();
                 to.setProjectId(11022);
                 to.setProjectName("Store Management");
                 String xml = XMLTransformer.toXML(to);
                 System.out.println(xml);
         }
}
```