XML

DTD(Document Type Definition)

- The purpose of DTD is to define the building blocks of an XML document.
- A DTD defines document structure with a list of legal attributes and elements.
- Example for DTD(newspaper)

```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

]>
```

- A DTD can be declared inline inside an XML document or as an external reference.
- Internal DTD Declaration: **<!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"
- !ELEMENT from defines the from element to be of type "#PCDATA"
- !ELEMENT heading defines the heading element to be of type "#PCDATA"
- !ELEMENT body defines the body element to be of type "#PCDATA"
- External DTD Declaration: **<!DOCTYPE root-element SYSTEM "filename">**
- XML Document is given below is saved separately let us say note.xml
- <?xml version="1.0"?>
  <!DOCTYPE note SYSTEM "**note.dtd**">
  <note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
  </note>
- External DTD is given as follows: (file name is **note.dtd**)

- <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
- **Use of 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.
- As XML documents are made up of building blocks, they are given as below:
  - Elements
  - Entities
  - Attributes
  - PCDATA and
  - CDATA

- **Elements** are the **main building blocks** of both XML and HTML documents.
- Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".
- Examples:
- <body>some text</body>
  <message>some text</message>
- **Attributes** provide **extra information about elements**.
- Attributes are always placed inside the opening tag of an element.
- Attributes always come in name/value pairs. The following "img" element has additional information about a source file:
-

- The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".
- **Entities**
- Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.
- Most of you know the HTML entity: " ". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.
- The following entities are predefined in XML:

  < <

  > >

  & &

  " "

  ' '

- **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 an XML parser. The text will be examined by the parser for entities and mark-up**.
- Tags inside the text will be treated as mark-up and entities will be expanded.
- However, parsed character data should not contain any &, <, or > characters; these need to be represented by the & < and > entities, respectively.
- **CDATA** means Character DATA.
- **CDATA is text that will NOT be parsed by a parser**. Tags inside the text will NOT be treated as mark-up and entities will not be expanded.
- In DTD elements declaration is given as follows:-
- **Declaring Elements:** <!ELEMENT element-name category> or <!ELEMENT element-name (element-content)>
- **Declaring empty elements:** Empty elements can be declared as <!ELEMENT element-name EMPTY> example: <!ELEMENT br EMPTY> and is used as
- **Elements with Parsed Character DATA:** <!ELEMENT element-name ANY> Example: <!ELEMENT note ANY>
- **Element with Children:** Elements with one or more children are declared as :

- <!ELEMENT element-name (child1)> or <!ELEMENT element-name (child1,child2,child3…)> Example: <!ELEMENT note (to, from, heading, body)>
- The full declaration of the "note" element is:

  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
- **Declaring Only One Occurrence of an Element**

  <!ELEMENT element-name (child-name)> **Example**: <!ELEMENT note (message)>
  - The example above declares that the child element "message" must occur once, and only once inside the "note" element.
- **Declaring Minimum One Occurrence of an Element**
- <!ELEMENT element-name (child-name+)> **Example**: <!ELEMENT note (message+)> The + sign in the example declares that the child element "message" must occur one or more times inside the "note" element.
- **Declaring Zero or More Occurrences of an Element**
- <!ELEMENT element-name (child-name*)> **Example**: <!ELEMENT note (message*)> The * sign in the example declares that the child element "message" can occur zero or more times inside the "note" element.

- **Declaring Zero or One Occurrences of an Element**
- <!ELEMENT element-name (child-name?)> **Example**: <!ELEMENT note (message?)> The ? sign in the example above declares that the child element "message" can occur zero or one time inside the "note" element.
- **Declaring either/or Content**
- **Example**: <!ELEMENT note (to,from,header,(message|body))> The example declares that the "note" element must contain a "to" element, a "from" element, a "header" element, and either a "message" or a "body" element.
- **Declaring Mixed Content**
- **Example**: <!ELEMENT note (#PCDATA|to|from|header|message)*> The example declares that the "note" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "message" elements.
- *XML parser is a defined as a program or a module that checks the well formed syntax and have the capability to manipulate the XML data elements.*

XML-DTD

- The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:
- <!DOCTYPE note
  [
  <!ELEMENT note (to ,from ,heading ,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  ]>

XML-Schema

- W3C supports an XML-based alternative to DTD, called XML Schema:
- <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>
```

XML SCHEMA

- XML schema is an XML based alternative to DTD. Which describes the structure of an XML document.
- The XML schema language is also referred as XML Schema Definition (XSD).
- The purpose of an XML schema is to define the legal building blocks of an XML document like a DTD. An XML schema defines :
  - elements and attributes that can appear in a document
  - which elements are child elements, the order of child elements
  - the number of child elements and whether an element is empty or can include text
  - data types for elements and attributes
  - default and fixed values for elements and attributes
- XML Schemas are the Successors of DTD's
- XML Schemas will be used in most Web applications as a replacement for DTDs because :
  - XML Schemas are richer and more powerful than DTDs
  - XML Schemas are extensible to future additions
  - XML Schemas are written in XML, also support data types and namespaces
- XML Schema is W3c recommendation.

- As XML Schemas Support Data Types
  - It is easier to describe allowable document content
  - It is easier to validate the correctness of data
  - It is easier to work with data from a database
  - It is easier to define restrictions on data
  - It is easier to define data patterns/formats
  - It is easier to convert data between different data types
- As XML Schemas use XML Syntax
  - No need to learn a new language
  - We can use an XML editor to edit Schema files
  - we can use XML parser to parse Schema files
  - we can manipulate Schema with the XML DOM
- Using XML schema we can provide the secured data communication between sender and receiver.
- As XML Schemas are extensible we can reuse the schema in other schema's and we can provide reference of multiple schemas in the same document.

Simple XML Document

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

Simple DTD

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

XML Schema

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

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM “http://www.w3schools.com/dtd/note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Reference to DTD

```
<?xml version="1.0"?>
<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Reference to XML Schema

What is DOM?

- The DOM is a W3C (World Wide Web Consortium) standard.
- The DOM defines a standard for accessing documents like XML and HTML:
- *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
- The DOM is separated into 3 different parts / levels:
  - Core DOM - standard model for any structured document
  - XML DOM - standard model for XML documents
  - HTML DOM - standard model for HTML documents
- The DOM defines the **objects and properties** of all document elements, and the **methods** (interface) to access them.
- The HTML DOM defines the **objects and properties** of all HTML elements, and the **methods** (interface) to access them.

What is XML DOM?

- The XML DOM is:
- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard
- The XML DOM defines the **objects and properties** of all XML elements, and the **methods** (interface) to access them.
- In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**
- In XML DOM everything in an xml document is treated as a node.

  DOM Nodes:
- The DOM says:
  - The entire document is a document node
  - Every XML element is an element node
  - The text in the XML elements are text nodes
  - Every attribute is an attribute node
  - Comments are comment nodes

DOM Example (books.xml)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
 <book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
```

```
 <year>2003</year>
   <price>49.99</price>
 </book>
 <book category="web" cover="paperback">
   <title lang="en">Learning XML</title>
   <author>Erik T. Ray</author>
   <year>2003</year>
   <price>39.95</price>
 </book>
</bookstore>
```

- The root node in the XML above is named <bookstore>. All other nodes in the document are contained within <bookstore>, which holds four <book> nodes.
- The first <book> node holds four nodes: <title>, <author>, <year>, and <price>, which contains one text node each, "Everyday Italian", "Giada De Laurentiis", "2005", and "30.00".
- **Text is Always Stored in Text Nodes**
- A common error in DOM processing is to expect an element node to contain text.
- However, the text of an element node is stored in a text node.
- In this example: **<year>2005</year>**, the element node <year>, holds a text node with the value "2005".
- "2005" is **not** the value of the <year> element!

- XML DOM views an XML document as a tree structure which is called as a **node-tree**. All the nodes in the tree have a relation ship with each other.
- All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created.
- The node tree shows the set of nodes, and the connections between them. The tree starts at the root node and branches out to the text nodes at the lowest level of the tree:

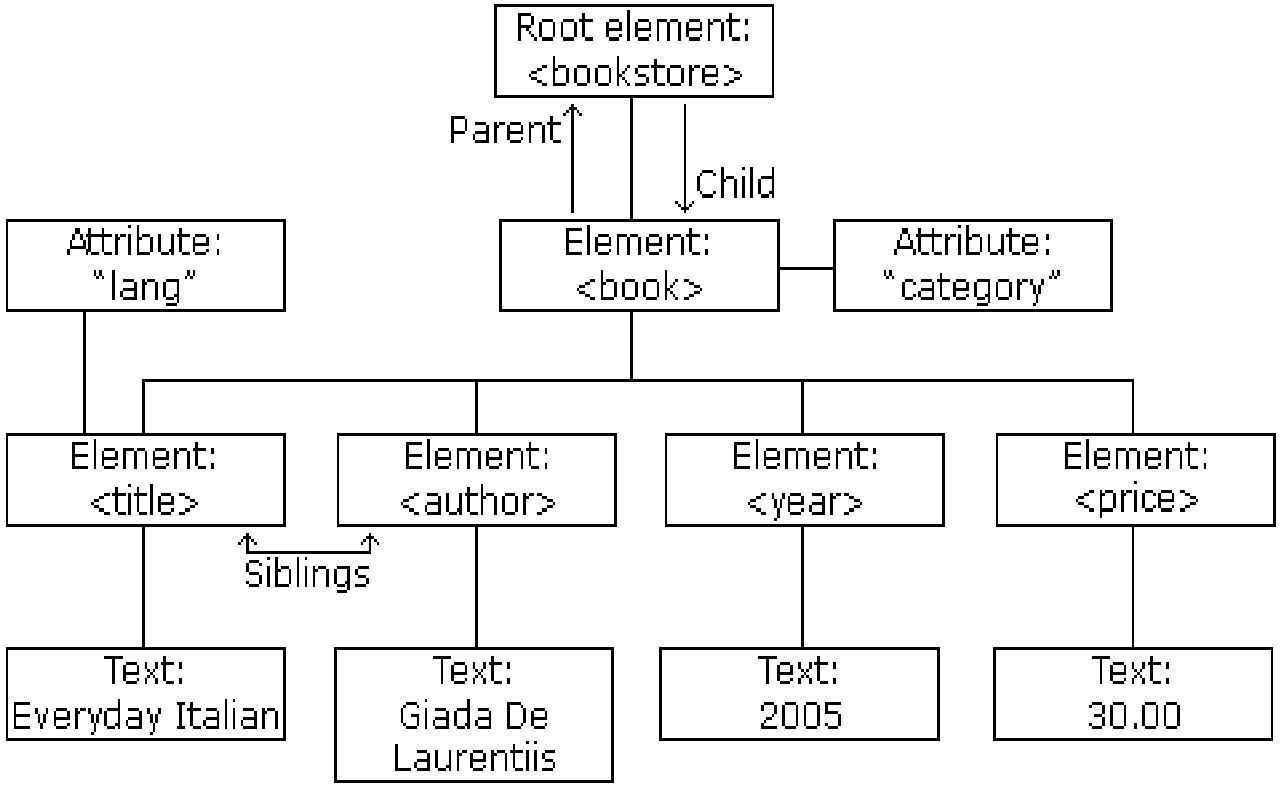

image above represents the XML file books.xml.

Parent, Child and Siblings:

- The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings who have the same parent.
  - In a node-tree, the top node is called the root
  - Every node, except the root, has exactly one parent node
  - A node can have any number of children
  - A leaf is a node with no children
  - Siblings are nodes with the same parent

```
<bookstore>                                  ➔ element <bookstore> is root
<book category="children">
  <title lang="en">Harry Potter</title>      ➔ element <title> is first child of <book>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>                       ➔ element <price> is  last child of <book>
 </book>
</bookstore>
```

- Element <book> is parent node to <title>, <author>, <year>, <price> and is child node to root node element named <bookstore>.

SAX PARSER

- Simple API for XML- SAX
  - Fast and Efficient
  - It will Fires Event handlers at Start Tag, Tag Body and End Tag e.g. <to> jk </to> respectively.
  - SAX is Read Only API therefore it cannot be used to Update XML documents.
  - It is good for large documents to process as it requires less memory.
  - It process the document sequentially.
  - SAX allows us to abort processing at any time.
  - It is useful to retrieve small amount of information.

```
<?xml version="1.0" encoding="UTF-8"?>
<parts>
<part> TurboWidget </part>
</parts>
```

```
StartDocument( )
StartElement( "parts" )
StartElement( "part" )
Characters( "TurboWidget" )
EndElement( "part" )
EndElement( "parts" )
EndDocument( )
```

Advantages of SAX:

- **SAX reduces memory and CPU usage** because it only processes one section of an XML document at a time.
- **SAX is customizable** because when an event is triggered, only the associated application receives data about the triggering event.
- **SAX is streamlined, fast, and supports pipelining** due to the reason that the parser can produce output while the document is being parsed.

Drawbacks of SAX:

- The SAX parser is unidirectional because It can only parse forwards in a document.
- As it can hold only a portion of the XML document in memory at any time, it is difficult to add or edit nodes using SAX. If this functionality is required, then DOM should be considered.
- SAX can only work with a fully formed XML document. It cannot be used to process partial XML documents.
- It won't support dynamic access of data in a document.
- It is difficult to implement complex searches.
- SAX support is not built-in in Microsoft® Internet Explorer.