XML Basics

XML stands for **E**xtensible **M**arkup **L**anguage and is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML is a software- and hardware-independent tool for storing and transporting data.

- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- **XML is extensible:** XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it:** XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

The Difference between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags
- XML documents form a tree structure that starts at "the root" and branches to "the leaves".

XML Syntax:

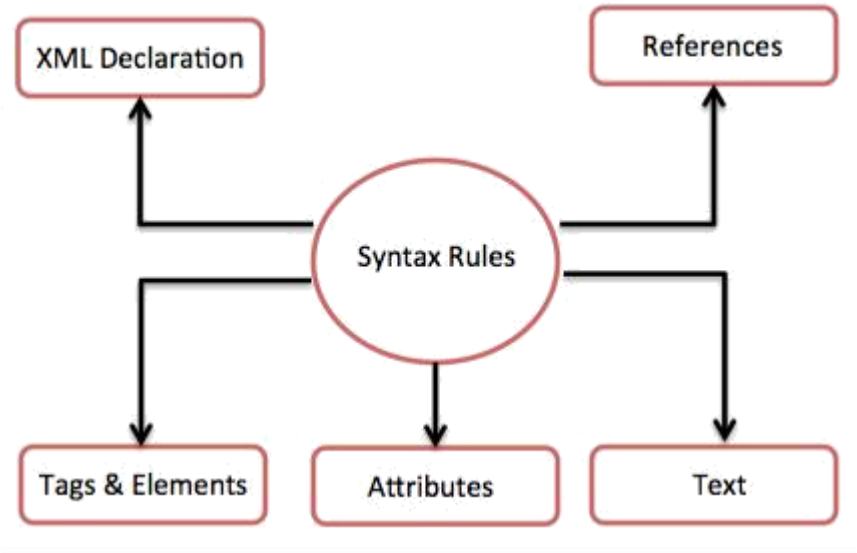
Following is a complete XML document:

```
<?xml version="1.0"?>
<contact_info>
     <name>Rajesh</name>
     <company>TCS</company>
     <phone>9333332354</phone>
</contact_info>
```

You can notice there are two kinds of information in the above example:

- markup, like <*contact-info*> and
- the text, like Rajesh etc.

The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



)

Let us see each component of the above diagram in detail:

XML Declaration

The XML document can optionally have an XML declaration. It is written as below:

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

Where *version* is the XML version and *encoding* specifies the character encoding used in the document.

Syntax Rules for XML declaration

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. XML-elements' names are enclosed by triangular brackets < > as shown below:

```
<element>
```

Syntax Rules for Tags and Elements

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element>....</element>
```

or in simple-cases, just this way:

```
<element/>
```

Nesting of elements:

An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags:

```
<?xml version="1.0"?>
<contact_info>
<company>TCS
<contact_info>
</company>
```

Following example shows correct nested tags:

```
<?xml version="1.0"?>
<contact_info>
<company>TCS</company>
<contact_info>
```

Root element:

An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

```
<x>...</x>
<y>...</y>
```

The following example shows a correctly formed XML document:

```
<root>
   <x>...</x>
   <y>...</y>
</root>
```

Case sensitivity:

The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example **<contact_info>** is different from **<Contact_Info>**.

Attributes

An **attribute** specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes. For example:

```
<a href="http://www.tutorialspoint.com/">Tutorialspoint!</a>
```

Here *href* is the attribute name and *http://www.tutorialspoint.com/* is attribute value.

Syntax Rules for XML Attributes

Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.

Same attribute cannot have two values in a syntax. The following example shows incorrect syntax because the attribute *b* is specified twice:

```
<a b="x" c="y" b="z">....</a>
```

Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates incorrect xml syntax:

```
<a b=x>....</a>
```

In the above syntax, the attribute value is not defined in quotation marks.

XML References

References usually allow you to add or include additional text or markup in an XML document. References always begin with the symbol **"&"** ,which is a reserved character and end with the symbol **";"**. XML has two types of references:

Entity References: An entity reference contains a name between the start and the end delimiters. For example **&** where *amp* is *name*. The *name* refers to a predefined string of text and/or markup.

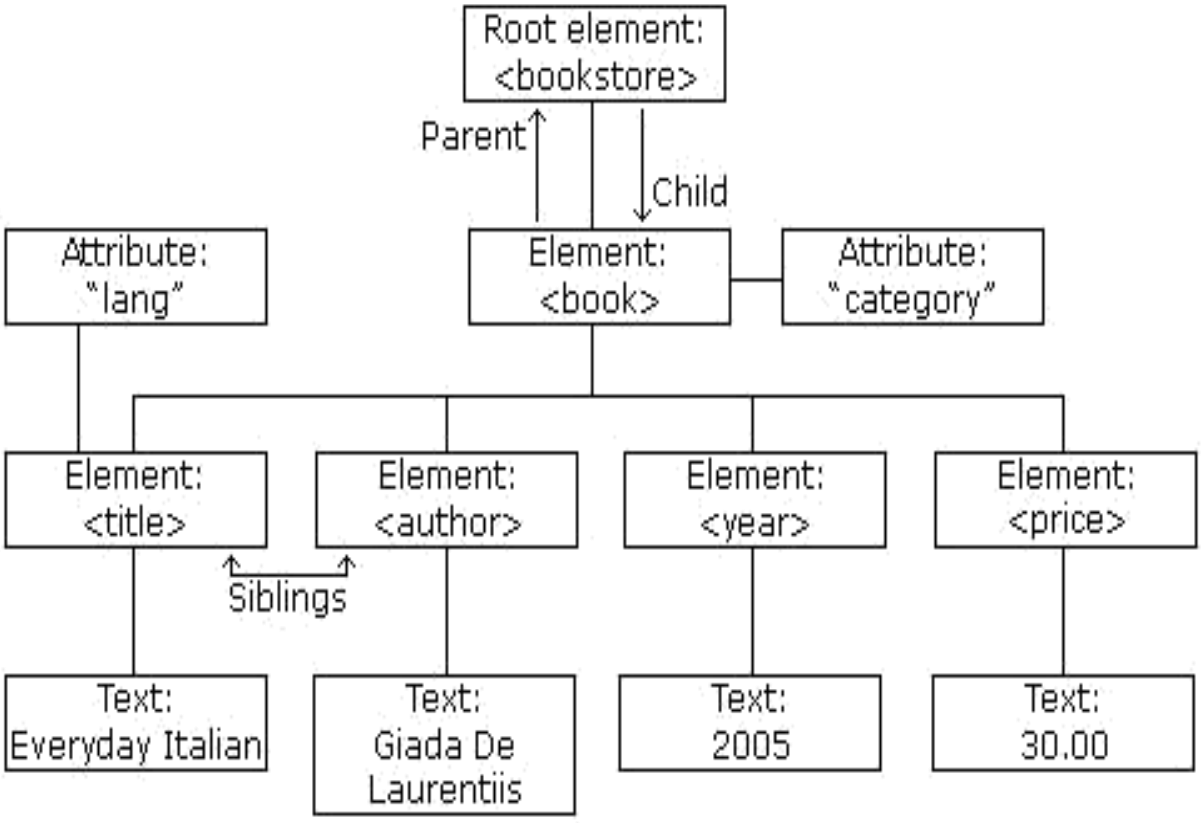
Character References: These contain references, such as **A**, contains a hash mark (“#”) followed by a number. The number always refers to the Unicode code of a character. In this case, 65 refers to alphabet "A".

XML Text

- The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case.
- To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.
- Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.
- Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly. To use them, some replacement-entities are used, which are listed below:

not allowed character	replacement-entity	character description
<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

XML Tree Structure:



An Example XML Document

The image above represents books in this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<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">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML documents are formed as **element trees**.

An XML tree starts at a **root element** and branches from the root to **child elements**.

All elements can have sub elements (child elements):

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

The terms parent, child, and sibling are used to describe the relationships between elements.

Parents have children. Children have parents. Siblings are children on the same level (brothers and sisters).

XML Namespaces

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

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

This XML carries information about a table (a piece of furniture):

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

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

A user or an XML application will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

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

This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a **namespace** for the prefix must be defined.

The namespace can be defined by an **xmlns** attribute in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

```
--------------------------------------------------

<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="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>

--------------------------------------------------
```

In the example above:

The xmlns attribute in the first <table> element gives the h: prefix a qualified namespace.

The xmlns attribute in the second <table> element gives the f: prefix a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

XML Validator

Use our XML validator to syntax-check your XML.

Well Formed XML Documents

An XML document with correct syntax is called "Well Formed".

The syntax rules were described in the previous chapters:

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

Example 1:

```
<?xml version=”1.0”?>

<book>
  <title>Java</Title>
  <author>James</book>
  <pirce>570
</author>
```

The above XML document is not a well formed document. Reasons given below...

- tags are not matching <title> … </Title>
- There is no proper nesting <author>….</book>
- Tag doesn’t closed <price>

Example 2:

```
<?xml version=”1.0”?>
<book>
  <title>Java</title>
  <author>James</author>
  <price>500</price>
</book>
```

The above XML document is a well formed document.

Valid XML Documents

A "well formed" XML document is not the same as a "valid" XML document.

A "valid" XML document must be well formed. In addition, it must conform to a document type definition.

There are two different document type definitions that can be used with XML:

- DTD - The original Document Type Definition
- XML Schema - An XML-based alternative to DTD

A document type definition defines the rules and the legal elements and attributes for an XML document.

XML DTD: (Document Type Definition)

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".
- A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD.
- DTD is the basic building block of XML.
- 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 book
[
<!ELEMENT book
(title,author,price)> <!ELEMENT
title (#PCDATA)> <!ELEMENT author
(#PCDATA)> <!ELEMENT price
(#PCDATA)> ]>
```

The DTD above is interpreted like this:

- !DOCTYPE book defines that the root element of the document is book
- !ELEMENT book defines that the book element must contain the elements: "title, author, price”
- !ELEMENT title defines the title element to be of type "#PCDATA"
- !ELEMENT author defines the author element to be of type "#PCDATA"
- !ELEMENT price defines the price element to be of type "#PCDATA"

Note: PCDATA: Parse able Character Data, CDATA: Character Data.

There are two types of DTDs:

1) Internal / Embedded DTD.

2) External DTD.

1) Internal / Embedded DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
<!ELEMENT student (id,name,age,addr,email,ph)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT addr (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT ph (#PCDATA)> ]>

<student>
        <id>543</id>
        <name>Ravi</name>
        <age>21</age>
        <addr>Guntur</addr>
        <email>nsr@gmail.com</email>
        <ph>9855555</ph>
        <gender>male</gender>
</student>
```

2) External DTD.

```
<!ELEMENT student (id,name,age,addr,email)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT addr (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Save the above code as “student.dtd” and prepare “student.xml” as follows...

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student SYSTEM "student.dtd">
<student>
<id>543</id>
<name>Ravi</name>
<age>21</age>
<addr>Guntur</addr>
<email>nsr@gmail.com</email>
</student>
```

In the above example we are using <!DOCTYPE student SYSTEM "student.dtd"> which is used to provide “student.dtd” code in our “student.xml” file.

If the above xml code follows the exact rules defined in DTD then we can conclude that our xml document is a valid document. Otherwise it is an invalid document.

When to Use a DTD/Schema?

- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- With a DTD, you can verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

XML Schema

An XML Schema describes the structure of an XML document, just like a DTD.

An XML document with correct syntax is called "Well Formed".

An XML document validated against an XML Schema is both "Well Formed" and "Valid".

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 Schema is an XML-based alternative to DTD:

Syntax

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

```
<xs:schema>

<xs:element name="book">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="author" type="xs:string"/>
    <xs:element name="price" type="xs:integer"/>
    <xs:element name="edition" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

The Schema above is interpreted like this:

- <xs:element name="book"> defines the element called "book" is a root.
- <xs:complexType> the "book" element is a complex type i.e. root
- <xs:sequence> the complex type is a sequence of elements i.e. childrens
- <xs:element name="title" type="xs:string"> the element "title" is of type string (text)
- <xs:element name="author" type="xs:string"> the element "author" is of type string
- <xs:element name="price" type="xs:integer"> the element "price" is of type integer.
- <xs:element name="edition" type="xs:string"> the element "edition" is of type string

Example:

```
---------------------------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com" elementFormDefault="qualified">
        <xs:element name="student">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="name" type="xs:string"/>
                        <xs:element name="age" type="xs:integer"/>
                        <xs:element name="addr">
                        <xs:complexType>
                        <xs:sequence>
                        <xs:element name="city" type="xs:string"/>
                        <xs:element name="pincode" type="xs:long"/>
                        </xs:sequence>
                        </xs:complexType>
                        </xs:element>
                        <xs:element name="ph" type="xs:integer"/>
                </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
-----------------------------------------------------------------------------------------------
```

Save the above code as "student.xsd"

Prepare "student.xml" as follows…

```
-----------------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com student.xsd">
        <name>rajesh</name>
        <age>25</age>
        <addr>
                <city>mylavaram</city>
                <pincode>53333</pincode>
        </add>
        <ph>9343434</ph>
</student>
-----------------------------------------------------------------------------------------------
```

If the above xml code follows the exact rules defined in "student.xsd" then we can conclude that our xml document is a valid document. Otherwise it is an invalid document.

XSLT Introduction

XSL (EXtensible Stylesheet Language) is a styling language for XML.

XSLT stands for XSL Transformations. **CSS = Style Sheets for HTML**

HTML uses predefined tags. The meaning of, and how to display each tag is well understood.

CSS is used to add styles to HTML elements.

XSL = Style Sheets for XML

XML does not use predefined tags, and therefore the meaning of each tag is not well understood.

A <table> element could indicate an HTML table, a piece of furniture, or something else - and browsers do not know how to display it!

So, XSL describes how the XML elements should be displayed.

What is XSLT?

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="book.xsl"?>

<book_store>
 <book>
  <title>JAVA</title>
  <author>James</author>
 </book>
 <book>
  <title>DBMS</title>
  <author>Raghu</author>

 </book>
</book_store>
```

Save the above code as "book.xml" and prepare the style sheet for this xml file.

```
<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet
version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
 <h2>Book Details</h2>
 <table border="1">
  <tr>
   <th>Title</th>
   <th>Author</th>
  </tr>
  <xsl:for-each select="book_store/book">
  <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="author"/></td>
  </tr>
  </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Save the above file as "book.xsl"

Now open "book.xml" file through any browser, observe the output as given below

Title	Author
Java	James
DBMS	Raghu

XML DOM Parser

The DOM defines a standard for accessing and manipulating documents:

The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.

The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.

The HTML DOM

All HTML elements can be accessed through the HTML DOM.

This example changes the value of an HTML element with id="demo":

Example:

```
<!DOCTYPE html>
<html>
<body>

<h1 id="demo">This is a Heading</h1>

<button type="button"
onclick="document.getElementById('demo').innerHTML = 'Hello World!'">Click Me!
</button>

</body>
</html>
```

The XML DOM

All XML elements can be accessed through the 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

In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

Programming Interface

The DOM models XML as a set of node objects. The nodes can be accessed with JavaScript or other programming languages.

The programming interface to the DOM is defined by a set standard properties and methods.

Properties are often referred to as something that is (i.e. nodename is "book").

Methods are often referred to as something that is done (i.e. delete "book").

XML DOM Properties

These are some typical DOM properties:

- x.nodeName - the name of x
- x.nodeValue - the value of x
- x.parentNode - the parent node of x
- x.childNodes - the child nodes of x
- x.attributes - the attributes nodes of x

Note: In the list above, x is a node object.

XML DOM Methods

- x.getElementsByTagName(*name*) - get all elements with a specified tag name
- x.appendChild(*node*) - insert a child node to x
- x.removeChild(*node*) - remove a child node from x

Note: In the list above, x is a node object.

DOM Example:

```
<html>
<body>

<p id="demo">this is paragraph text</p>
<button type="button" onclick="myfun()">click me</button>

<script>
function myfun()
{
	var text, parser, xmlDoc;

	text = "<bookstore><book>" +
	"<title>Java</title>" +
	"<author>James</author>" +
	"<year>1991</year>" +
	"<book>" +
	"<title>DBMS</title>" +
	"<author>Raghu</author>" +
	"<year>1970</year>" +
	"</book>"+
	"</book></bookstore>";

	parser = new DOMParser();
	xmlDoc = parser.parseFromString(text,"text/xml");

	document.getElementById("demo").innerHTML =
	xmlDoc.getElementsByTagName("year")[0].childNodes[0].nodeValue;
}
</script>
</body>
</html>
```

Output:

1991

Example Explained

- **xmlDoc** - the XML DOM object created by the parser.
- **getElementsByTagName("year")[0]** - get the first <year> element
- **childNodes[0]** - the first child of the <year> element (the text node)
- **nodeValue** - the value of the node (the text itself)