XML

WHAT IS XML?

- XML is a software- and hardware-independent tool for storing and transporting data.
- Stands for eXtensible Markup Language
- It is a markup language much like HTML
- Designed to store and transport data
- Designed to be self-descriptive
- XML is a W3C Recommendation

XML Example

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

- XML does not DO anything.
- This note is a note to Tove from Jani, stored as XML
- The XML above is quite self-descriptive:
  - ✓ It has sender information.
  - ✓ It has receiver information
  - ✓ It has a heading
  - ✓ It has a message body.
- But still, the XML above does not DO anything.
- XML is just information wrapped in tags.
- Someone must write a piece of software to send, receive, store, or display it

XML Example

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

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 are

The tags in the example above (like <to> and <from>) are not defined in any XML standard.

These tags are "invented" by the author of the XML document.

HTML works with predefined tags like <p>, <h1>, <table>, etc.

XML is Extensible

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

```xml
<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

- Most XML applications will work as expected even if new data is added (or removed).
- Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <body>).
- Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.
- The way XML is constructed, older version of the application can still work

XML SIMPLIFIES THINGS

Data Sharing

- Many computer systems contain data in incompatible formats.
- Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers.
- Large amounts of data must be converted, and incompatible data is often lost.

Data Transport

- XML stores data in plain text format.
- This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML SIMPLIFIES THINGS

Platform sharing

- XML makes it easier to expand or upgrade without losing data to
- new operating systems
- new applications
- new browsers

Data Availability

- data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

HOW TO USE XML?

XML Separates Data from Presentation

- XML does not carry any information about how to be displayed.
- The same XML data can be used in many different presentation scenarios.
- Because of this, with XML, there is a full separation between data and presentation.

XML is Often a Complement to HTML

- In many HTML applications, XML is used to store or transport data, while HTML is used to format and display the same data.

HOW TO USE XML?

XML Separates Data from HTML

- When displaying data in HTML, you should not have to edit the HTML file when the data changes.
- With XML, the data can be stored in separate XML files.
- With a few lines of JavaScript code, you can read an XML file and update the data content of any HTML page.

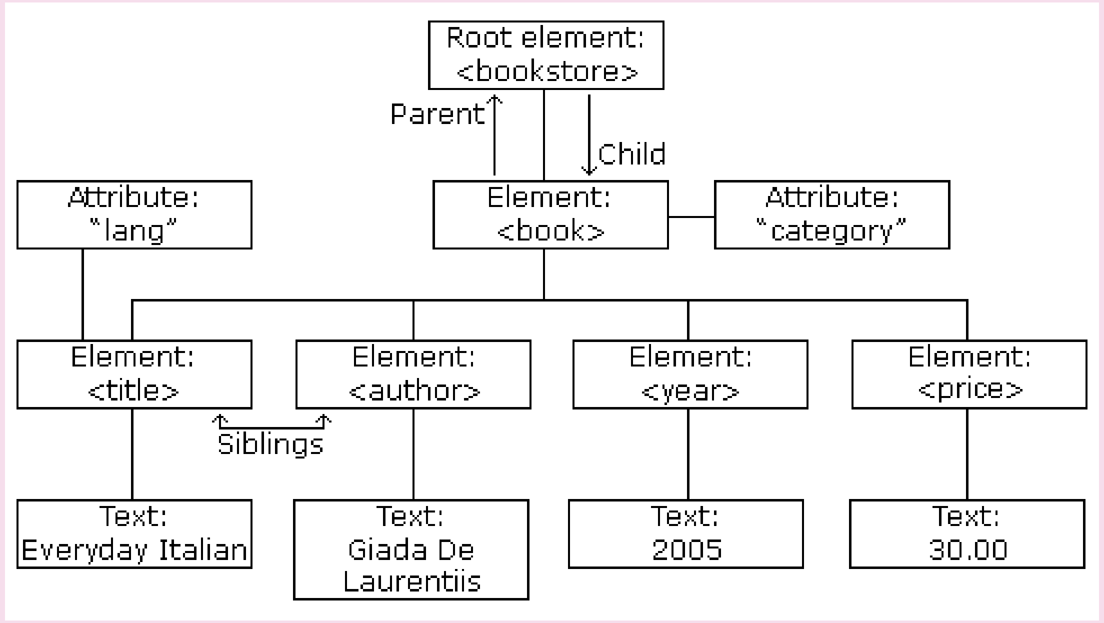
Books.xml

```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">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>
```

Title	Author
Everyday Italian	Giada De Laurentiis
Harry Potter	J K. Rowling
XQuery Kick Start	James McGovern
Learning XML	Erik T. Ray

XML Tree

```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 Tree

```
<?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).
- All elements can have text content (Harry Potter) and attributes (category="cooking").

XML Syntax Rule

```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 document must have ROOT element
- XML Prolog: Optional, if exist then first statement
- All XML elements must have closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML Attribute Values Must Always be Quoted
- White space is preserved in XML

Entity Reference

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

XML HTTPRequest

- All modern browsers have a built-in XMLHttpRequest object to request data from a server.
- The XMLHttpRequest object can be used to request data from a web server.

The XMLHttpRequest object is a developers dream, because you can:

1. Update a web page without reloading the page
2. Request data from a server - after the page has loaded
3. Receive data from a server - after the page has loaded
4. Send data to a server - in the background

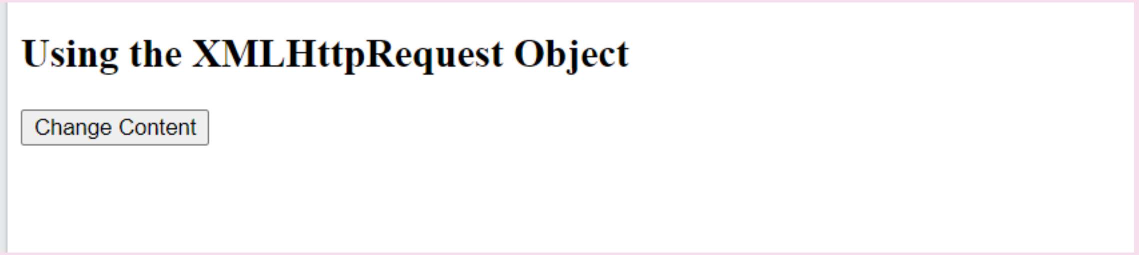
Sending an XMLHTTPRequest

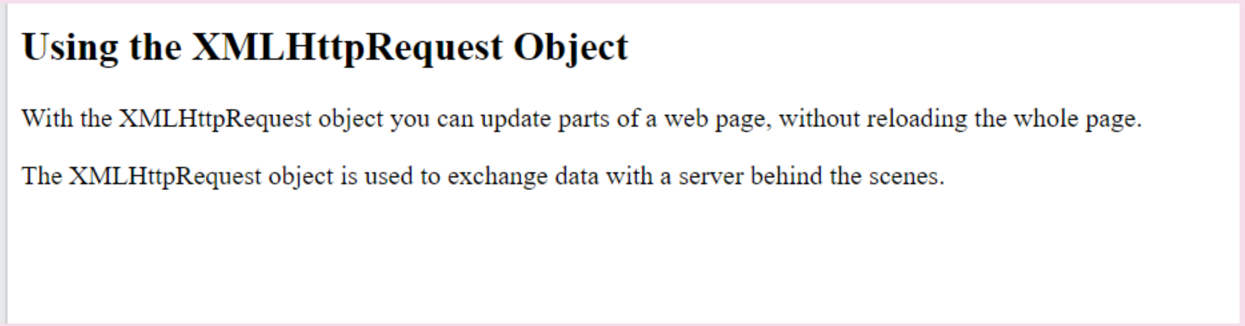
```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
       // Typical action to be performed when the document
is ready:
       document.getElementById("demo").innerHTML =
xhttp.responseText;
    }
};
xhttp.open("GET", "filename", true);
xhttp.send();
```

- The first line in the example above creates an **XMLHttpRequest** object
- The **onreadystatechange** property specifies a function to be executed every time the status of the XMLHttpRequest object changes
- When **readyState** property is 4 and the **status** property is 200, the response is ready
- The **responseText** property returns the server response as a text string.
- The text string can be used to update a web page

Sending an XMLHTTPRequest

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

<h2>Using the XMLHttpRequest Object</h2>

<div id="demo">
<button type="button" onclick="loadXMLDoc()">Change
Content</button>
</div>

<script>
function loadXMLDoc() {
  var xmlhttp;
  if (window.XMLHttpRequest) {
    xmlhttp = new XMLHttpRequest();
  } else {
    // code for older browsers
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xmlhttp.open("GET", "xmlhttp_info.txt", true);
  xmlhttp.send();
}
</script>

</body>
</html>
```


Using the XMLHttpRequest Object


Using the XMLHttpRequest Object

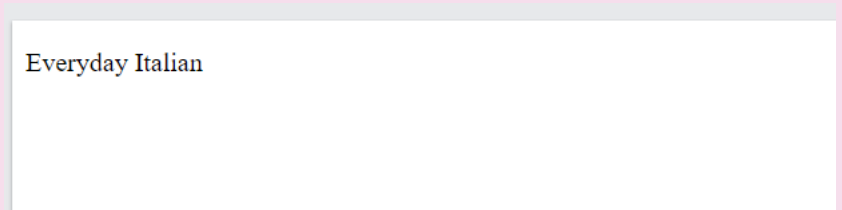
With the XMLHttpRequest object you can update parts of a web page, without reloading the whole page.

The XMLHttpRequest object is used to exchange data with a server behind the scenes.

XML Parser

- The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.
- However, before an XML document can be accessed, it must be loaded into an XML DOM object.
- All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

```
<html>
<body>

<p id="demo"></p>

<script>
var text, parser, xmlDoc;

// text string is defined
text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";


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

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```



```
Everyday Italian
```

XML Parser

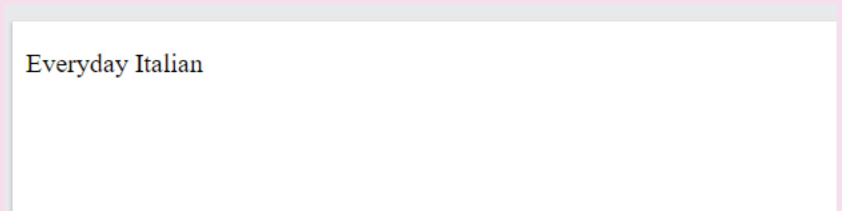
- All XML elements can be accessed through the XML DOM.
- This code retrieves the text value of the first <title> element in an XML document:

Example

txt = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;

- The XML DOM is a standard for how to get, change, add, and delete XML elements.
- This example loads a text string into an XML DOM object, and extracts the info from it with JavaScript:

```
<html>
<body>

<p id="demo"></p>

<script>
var text, parser, xmlDoc;

// text string is defined
text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";


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

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```



```
Everyday Italian
```

XPath

- is a major element in the XSLT standard.
- can be used to navigate through elements and attributes in an XML document.
- is a syntax for defining parts of an XML document
- uses path expressions to navigate in XML documents
- contains a library of standard functions
- is a major element in XSLT and in XQuery
- is a W3C recommendation

Xpath Path Expression

- XPath uses path expressions to select nodes or node-sets in an XML document.
- These path expressions look very much like the expressions you see when you work with a traditional computer file system.
- XPath expressions can be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

Xpath Path Expression

XPath Expression	**Result**
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00