XML training doc

Introduction

What is XML ?

- XML is a markup language for documents containing structured information.
- XML stands for eXtensible Markup Language. It is designed to transport and store data.
- A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

What's a Document ?

The number of applications currently being developed that are based on, or make use of, XML documents is truly amazing! For our purposes, the word "document" refers not only to traditional documents, like this one, but also to the myriad of other XML "data formats". These include vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs, and a thousand other kinds of structured information.

Is it just like HTML?

No.

In HTML, both the tag semantics and the tag set are fixed. An <h1> is always a first level heading and the tag <ati.product.code> is meaningless. The W3C, in conjunction with browser vendors and the WWW community, is constantly working to extend the definition of HTML to allow new tags to keep pace with changing technology and to bring variations in presentation (stylesheets) to the Web. However, these changes are always rigidly confined by what the browser vendors have implemented and by the fact that backward compatibility is paramount.

And for people who want to disseminate information widely, features supported by only the latest releases of Netscape and Internet Explorer are not useful. XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them.

Since there's no predefined tag set, there can't be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by stylesheets.

The Difference Between XML and HTML

1. XML is not a replacement for HTML.
2. XML and HTML were designed with different goals:
3. XML was designed to transport and store data, with focus on what data is.
4. HTML was designed to display data, with focus on how data looks.
5. HTML is about displaying information, while XML is about carrying information.

Why XML?

In order to appreciate XML, it is important to understand why it was created. XML was created so that richly
structured documents could be used over the web. The only viable alternatives, HTML and SGML, are not practical for this purpose.

HTML, as we've already discussed, comes bound with a set of semantics and does not provide arbitrary
structure.

SGML provides arbitrary structure, but is too difficult to implement just for a web browser. Full SGML
systems solve large, complex problems that justify their expense. Viewing structured documents sent over the web rarely carries such justification.

This is not to say that XML can be expected to completely replace SGML. While XML is being designed to
deliver structured content over the web, some of the very features it lacks to make this practical, make SGML a more satisfactory solution for the creation and long-time storage of complex documents. In many organizations, filtering SGML to XML will be the standard procedure for web delivery.

Origin and goals :

XML was developed by an XML Working Group (originally known as the SGML Editorial Review
Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. The membership of the XML Working Group is given in an appendix.
Dan Connolly served as the Working Group's contact with the W3C.

- The design goals for XML are:

  - XML shall be straightforwardly usable over the Internet.
  - XML shall support a wide variety of applications.
  - XML shall be compatible with SGML.
  - It shall be easy to write programs which process XML documents.
  - The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
  - XML documents should be human-legible and reasonably clear.
  - The XML design should be prepared quickly.
  - The design of XML shall be formal and concise.
  - XML documents shall be easy to create.
  - Terseness in XML markup is of minimal importance.

Key Terminology

The material in this section is based on the XML Specification. This is not an exhaustive list of all the constructs that appear in XML; it provides an introduction to the key constructs most often encountered in day-to-day use.

- **(Unicode) character**
  By definition, an XML document is a string of characters. Almost every legal Unicode character may appear in an XML document.
- **Processor and application**
  The processor analyzes the markup and passes structured information to an application. The specification places requirements on what an XML processor must do and not do, but the application is outside its scope. The processor (as the specification calls it) is often referred to colloquially as an XML parser.
- **Markup and content**
  The characters making up an XML document are divided into markup and content, which may be distinguished by the application of simple syntactic rules. Generally, strings that constitute markup either begin with the character < and end with a >, or they begin with the character & and end with a ;. Strings of characters that are not markup are content. However, in a CDATA section, the delimiters <![CDATA[and]]> are classified as markup, while the text between them is classified as content. In addition, whitespace before and after the outermost element is classified as markup.

- **Tag**
  A markup construct that begins with < and ends with >. Tags come in three flavors:
  start-tags; for example: <section>
  end-tags; for example: </section>
  empty-element tags; for example: <line-break />
- **Element**
  A logical document component which either begins with a start-tag and ends with a matching
  end-tag or consists only of an empty-element tag. The characters between the start- and end-tags, if any, are the
  element's content, and may contain markup, including other elements, which are called child elements. An example of an element is <Greeting>Hello, world.</Greeting> (see hello world). Another is <line-break />.
- **Attribute**
  A markup construct consisting of a name/value pair that exists within a start-tag or empty-
  element tag. In the example (below) the element img has two attributes, src and alt:
  <img src="madonna.jpg" alt='Foligno Madonna, by Raphael'/>
  Another example would be
  <step number="3">Connect A to B.</step>
  where the name of the attribute is "number" and the value is "3".
- **XML declaration**
  XML documents may begin by declaring some information about themselves, as in the
  following example:
  <?xml version="1.0" encoding="UTF-8" ?>

XML Documents

- Escaping
  XML provides escape facilities for including characters which are problematic to include directly.
  For example:
  The characters "<" and "&" are key syntax markers and may never appear in content outside a CDATA section.
  Some character encodings support only a subset of Unicode. For example, it is legal to encode an XML document in ASCII, but ASCII lacks code points for Unicode characters such as " � ". It might not be possible to type the character on the author's machine. Some characters have glyphs that cannot be visually distinguished from other
  characters:
  There are five predefined entities:

    - &lt; represents "<"
    - &gt; represents ">"
    - &amp; represents "&"
    - &apos; represents '
    - &quot; represents "
- **Comments**
  Comments may appear anywhere in a document outside other markup. Comments cannot appear before the XML declaration. Comments start with "<!--" and end with "-->". The string "--" (double-hyphen) is not allowed inside comments; this means comments cannot be nested. The ampersand has no special significance within comments, so entity and character references are not recognized as such, and there is no way to represent characters outside the character set of the document encoding.
  An example of a valid comment: "<!-- no need to escape <code> & such in comments -->"
- **Validity**
  1. Well-formed Documents
     A document can only be well-formed if it obeys the syntax of XML. A document that includes sequences of markup characters that cannot be parsed or are invalid cannot be well-formed. In addition, the document must meet all of the following conditions (understanding some of these conditions may require experience with SGML).
     - No attribute may appear more than once on the same start-tag.
     - String attribute values cannot contain references to external entities.
     - Non-empty tags must be properly nested.
     - Parameter entities must be declared before they are used.
     - All entities except the following: amp, lt, gt, apos, and quot must be declared.
     - A binary entity cannot be referenced in the flow of content, it can only be used in an attribute declared
       as ENTITY or ENTITIES.
     - Neither text nor parameter entities are allowed to be recursive, directly or indirectly.
     - By definition, if a document is not well-formed, it is not XML. This means that there is no such thing
       as an XML document which is not well-formed, and XML processors are not required to do anything with such documents.
  2. Valid Documents
     A well-formed document is valid only if it contains a proper document type declaration and if the
     document obeys the constraints of that declaration (element sequence and nesting is valid, required attributes are provided, attribute values are of the correct type, etc.). The XML specification identifies all of the criteria in detail.

- **XML Syntax Rules**
  The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.
  1. *All XML Elements Must Have a Closing Tag*
     In HTML, some elements do not have to have a closing tag:
     <p>This is a paragraph.
     <br>
     In XML, it is illegal to omit the closing tag. All elements must have a closing tag:
     <p>This is a paragraph.</p>
     <br />
     Note: You might have noticed from the previous example that the XML declaration did not
     have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.
  2. *XML Tags are Case Sensitive.*
     XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.
     Opening and closing tags must be written with the same case:
     <Message>This is incorrect</message>
     <message>This is correct</message>
     Note: "Opening and closing tags" are often referred to as "Start and end tags". Use whatever
     you prefer. It is exactly the same thing.
  3. *XML Elements Must be Properly Nested*
     In HTML, you might see improperly nested elements:
     <b><i>This text is bold and italic</b></i>
     In XML, all elements must be properly nested within each other:
     <b><i>This text is bold and italic</i></b>
     In the example above, "Properly nested" simply means that since the <i> element is opened
     inside the <b> element, it must be closed inside the <b> element.
  4. *XML Documents Must Have a Root Element*
     XML documents must contain one element that is the parent of all other elements. This
     element is called the root element.
     ```
     <root>
       <child>
         <subchild>.....</subchild>
       </child>
     </root>
     ```
  5. *XML Attribute Values Must be Quoted*
     XML elements can have attributes in name/value pairs just like in HTML.
     In XML, the attribute values must always be quoted.

Study the two XML documents below. The first one is incorrect, the second is correct:

```
<note date=12/11/2007>
   <to>Tove</to>
    <from>Jani</from>
</note>
<note date="12/11/2007">
   <to>Tove</to>
   <from>Jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

- **CDATA section**

  In a document, a CDATA section instructs the parser to ignore most markup characters.

  Consider a source code listing in an XML document. It might contain characters that the XML parser would ordinarily recognize as markup (< and &, for example). In order to prevent this, a CDATA section can be used.

  ```
  <![CDATA[
              *p = &q;
               b = (i <= 3);
  ]]>
  ```

  Between the start of the section, <![CDATA[and the end of the section,]]>, all character data is passed directly to the application, without interpretation. Elements, entity references, comments, and processing instructions are all unrecognized and the characters that comprise them are passed literally to the application.

  The only string that cannot occur in a CDATA section is]]>.

DTD

The oldest schema language for XML is the Document Type Definition (DTD), inherited from SGML.

DTDs have the following benefits:

- DTD support is ubiquitous due to its inclusion in the XML 1.0 standard.
- DTDs are terse compared to element-based schema languages and consequently present more information in a single screen.
- DTDs allow the declaration of standard public entity sets for publishing characters.

- DTDs define a document type rather than the types used by a namespace, thus grouping all

constraints for a document in a single collection.

DTDs have the following limitations:

- They have no explicit support for newer features of XML, most importantly namespaces.
- They lack expressiveness. XML DTDs are simpler than SGML DTDs and there are certain structures that cannot be expressed with regular grammars. DTDs only support rudimentary datatypes.
- They lack readability. DTD designers typically make heavy use of parameter entities (which behave essentially as textual macros), which make it easier to define complex grammars, but at the expense of clarity.
- They use a syntax based on regular expression syntax, inherited from SGML, to describe the schema. Typical XML APIs such as SAX do not attempt to offer applications a structured representation of the syntax, so it is less accessible to programmers than an element-based syntax may be.

Two peculiar features that distinguish DTDs from other schema types are the syntactic support for embedding a DTD within XML documents and for defining entities, which are arbitrary fragments of text and/or markup that the XML processor inserts in the DTD itself and in the XML document wherever they are referenced, like character escapes.
DTD technology is still used in many applications because of its ubiquity.
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.

- Internal DTD Declaration
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>
```

Open the XML file above in your browser (select "view source" or "view page source" to
view the DTD)

- 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

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">
```

This is the same XML document as above, but with an external DTD (Open it, and select view source):

```
<?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>
```

And this is the file "note.dtd" which contains the DTD:

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

- 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.

XML schema

XML Schema is an XML-based alternative to DTD.
An XML schema describes the structure of an XML document.
The XML Schema language is also referred to as XML Schema Definition (XSD).
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 attributesdefines default and fixed values for elements and attributes.

XML Schemas are the Successors of DTDs.
We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:

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

Why Use XML Schemas?

XML Schemas are much more powerful than DTDs.
XML Schemas Support Data Types
One of the greatest strength of XML Schemas is the support for data types.

With support for 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 data facets (restrictions on data)
It is easier to define data patterns (data formats)
It is easier to convert data between different data types

XML Schemas use XML Syntax

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

Some benefits of that XML Schemas are written in XML:

You don't have to learn a new language

You can use your XML editor to edit your Schema files

You can use your XML parser to parse your Schema files

You can manipulate your Schema with the XML DOM

You can transform your Schema with XSLT

XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

<date type="date">2004-03-11</date>

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

XML Schemas are Extensible

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

Reuse your Schema in other Schemas

Create your own data types derived from the standard types

Reference multiple schemas in the same document

Load an XML file

Using JavaScript.

```
<!DOCTYPE html>
<html>
<head>
<style>
table, th, td {
    border: 1px solid black;
    border-collapse:collapse;
}
th, td {
    padding: 5px;
}
</style>
```

```
</head>
<body>

<script>
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.open("GET","cd_catalog.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.write("<table><tr><th>Artist</th><th>Title</th></tr>");
var x=xmlDoc.getElementsByTagName("CD");
for (i=0;i<x.length;i++)
  {
  document.write("<tr><td>");
  document.write(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].noc
  document.write("</td><td>");
  document.write(x[i].getElementsByTagName("TITLE")[0].childNodes[0].node
  document.write("</td></tr>");
  }
document.write("</table>");
</script>
</body>
</html>
```

** **cd_catalog.xml** **

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
    <CD>
        <TITLE>Empire Burlesque</TITLE>
        <ARTIST>Bob Dylan</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Columbia</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1985</YEAR>
    </CD>
    <CD>
        <TITLE>Hide your heart</TITLE>
        <ARTIST>Bonnie Tyler</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>CBS Records</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1988</YEAR>
    </CD>
    <CD>
        <TITLE>Greatest Hits</TITLE>
        <ARTIST>Dolly Parton</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>RCA</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1982</YEAR>
    </CD>
    <CD>
        <TITLE>Still got the blues</TITLE>
        <ARTIST>Gary Moore</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Virgin records</COMPANY>
        <PRICE>10.20</PRICE>
        <YEAR>1990</YEAR>
    </CD>
    <CD>
        <TITLE>Eros</TITLE>
        <ARTIST>Eros Ramazzotti</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>BMG</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1997</YEAR>
    </CD>
    <CD>
        <TITLE>One night only</TITLE>
        <ARTIST>Bee Gees</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Polydor</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1998</YEAR>
    </CD>
    <CD>
        <TITLE>Sylvias Mother</TITLE>
        <ARTIST>Dr.Hook</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>CBS</COMPANY>
        <PRICE>8.10</PRICE>
        <YEAR>1973</YEAR>
    </CD>
```

```
<CD>
    <TITLE>Maggie May</TITLE>
    <ARTIST>Rod Stewart</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Pickwick</COMPANY>
    <PRICE>8.50</PRICE>
    <YEAR>1990</YEAR>
</CD>
<CD>
    <TITLE>Romanza</TITLE>
    <ARTIST>Andrea Bocelli</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.80</PRICE>
    <YEAR>1996</YEAR>
</CD>
<CD>
    <TITLE>When a man loves a woman</TITLE>
    <ARTIST>Percy Sledge</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Atlantic</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1987</YEAR>
</CD>
<CD>
    <TITLE>Black angel</TITLE>
    <ARTIST>Savage Rose</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Mega</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1995</YEAR>
</CD>
<CD>
    <TITLE>1999 Grammy Nominees</TITLE>
    <ARTIST>Many</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Grammy</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1999</YEAR>
</CD>
<CD>
    <TITLE>For the good times</TITLE>
    <ARTIST>Kenny Rogers</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Mucik Master</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1995</YEAR>
</CD>
<CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
</CD>
<CD>
    <TITLE>Tupelo Honey</TITLE>
    <ARTIST>Van Morrison</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1971</YEAR>
</CD>
<CD>
    <TITLE>Soulsville</TITLE>
    <ARTIST>Jorn Hoel</ARTIST>
    <COUNTRY>Norway</COUNTRY>
    <COMPANY>WEA</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1996</YEAR>
</CD>
<CD>
    <TITLE>The very best of</TITLE>
    <ARTIST>Cat Stevens</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Island</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1990</YEAR>
</CD>
<CD>
    <TITLE>Stop</TITLE>
    <ARTIST>Sam Brown</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>A and M</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1988</YEAR>
</CD>
<CD>
    <TITLE>Bridge of Spies</TITLE>
```

```
        <ARTIST>T'Pau</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Siren</COMPANY>
        <PRICE>7.90</PRICE>
        <YEAR>1987</YEAR>
    </CD>
    <CD>
        <TITLE>Private Dancer</TITLE>
        <ARTIST>Tina Turner</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Capitol</COMPANY>
        <PRICE>8.90</PRICE>
        <YEAR>1983</YEAR>
    </CD>
    <CD>
        <TITLE>Midt om natten</TITLE>
        <ARTIST>Kim Larsen</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>Medley</COMPANY>
        <PRICE>7.80</PRICE>
        <YEAR>1983</YEAR>
    </CD>
    <CD>
        <TITLE>Pavarotti Gala Concert</TITLE>
        <ARTIST>Luciano Pavarotti</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>DECCA</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1991</YEAR>
    </CD>
    <CD>
        <TITLE>The dock of the bay</TITLE>
        <ARTIST>Otis Redding</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Atlantic</COMPANY>
        <PRICE>7.90</PRICE>
        <YEAR>1987</YEAR>
    </CD>
    <CD>
        <TITLE>Picture book</TITLE>
        <ARTIST>Simply Red</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>Elektra</COMPANY>
        <PRICE>7.20</PRICE>
        <YEAR>1985</YEAR>
    </CD>
    <CD>
        <TITLE>Red</TITLE>
        <ARTIST>The Communards</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>London</COMPANY>
        <PRICE>7.80</PRICE>
        <YEAR>1987</YEAR>
    </CD>
    <CD>
        <TITLE>Unchain my heart</TITLE>
        <ARTIST>Joe Cocker</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>EMI</COMPANY>
        <PRICE>8.20</PRICE>
        <YEAR>1987</YEAR>
    </CD>
</CATALOG>
```

XPath

What is XPath

The XPath is an official recommendation of the World Wide Web Consortium (W3C). It defines a language to find information in an XML file. It is used to traverse elements and attributes of an XML document. XPath provides various types of expressions which can be used to enquire relevant information from the XML document.

- **Structure Definations** - XPath defines the parts of an XML document like element, attribute, text, namespace, processing-instruction, comment, and document nodes

- **Path Expressions** XPath provides powerful path expressions select nodes or list of nodes in XML documents.

- **Standard Functions**XPath provides a rich library of standard functions for manipulation of string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values etc.

- **Major part of XSLT**XPath is one of the major elements in XSLT standard and is must have knowledge in order to work with XSLT documents.

- **W3C recommendation**XPath is official recommendation of World Wide Web Consortium (W3C).

Features

- XPath is core component of XSLT standard. XSLT can not work without XPath.
- XPath is basis of XQuery and XPointer.

Syntax and semantics

This section describes XPath 1.0.
The most important kind of expression in XPath is a location path. A location path consists of
a sequence of location steps.

Each location step has three components:

1. **An axis**

   `child::`

   Selects children of the context node. Attribute nodes are not included; use the **`attribute::`** axis to get attribute nodes.

   `parent::`

   Selects only the parent node, if there is one. Can be abbreviated as "`../`"

   `self::`

   Selects only the context node. This can be abbreviated as "`./`".

   `attribute::`

   Selects only the attributes of the context node. Can be abbreviated as "`@`".

   `ancestor::`

   Refers to all ancestors of the context node: its parent, its parent's parent, and so on up to and including the document node.

   `ancestor-or-self::`

   Refers to the context node and its ancestors.

`descendant::`

Refers to the descendants of the context node: its children, their children, and so on up to and including the leaves of the tree.

`descendant-or-self::`

Refers to the context node and its descendants.

`preceding-sibling::`

Refers to all children of the context node's parent that occur before the context node.

`following-sibling::`

Refers to all children of the context node's parent that occur after the context node.

`preceding::`

All nodes that precede the context node in the whole document. This set does not include the context node's descendants or attributes.

`following::`

All nodes that follow the context node in the whole document. This set does not include the context node's descendants or attributes.
axes.jpg

2. A node test
   `text()`
   This function selects all the text children of the context node.

   `comment()`
   Selects all comments that are children of the context node.

   `processing-instruction()`
   Selects all children of the context node that are processing instructions.

3. Zero or more predicates.

XQuery

- Introduction

The W3C is finalizing the XQuery specification, aiming for a final release in late 2002. XQuery is a
powerful and convenient language designed for processing XML data. That means not only files in XML format, but also other data including databases whose structure -- nested, named trees with attributes -- is similar to XML. XQuery provides the means to extract and manipulate data from XML documents or any data source
that can be viewed as XML, such as relational databases or office documents.

- An Expression Language

The first thing to note is that in XQuery everything is an expression which evaluates to a
value. An XQuery program or script is a just an expression, together with some optional function and other definitions.
So 3+4 is a complete, valid XQuery program which evaluates to the integer 7.
There are no side-effects or updates in the XQuery standard, though they will probably be added at a future date. The standard specifies the result value of an expression or program, but it does not specify how it
is to be evaluated. An implementation has considerable freedom in how it evaluates an XQuery program, and what optimizations it does.

- Primitive Data Types

The primitives data types in XQuery are the same as for XML Schema.
Numbers, including integers and floating-point numbers.
The boolean values true and false.
Strings of characters, for example: "Hello world!". These are immutable - i.e. you cannot modify a character in a string.
Various types to represent dates, times, and durations.

- Path expression and relation to XPath

XQuery borrows path expressions from XPath. XQuery can be viewed as a generalization of XPath.
Except for some obscure forms (mostly unusual "axis specifiers"), all XPath expressions are also XQuery expressions.
For this reason the XPath specification is also being revised by the XQuery committee, with the plan that XQuery 1.0 and XPath 2.0 will be released

about the same time.
The following simple example assumes an XML file "mybook.xml" whose root element is a <book>,
containing some <chapter> children:
let $book := document("mybook.xml")/book
return $book/chapter
The document function returns the root node of a document. The /book expression selects the child
elements of the root that are named book, so $book gets set to the single root element.
The $book/chapter selects the child elements of the top-level book elements, which results in a
sequence of the second-level chapter nodes in document order.
The next example includes a predicate:
$book//para[@class="warning"]
The double slash is a convenience syntax to select all descendants (rather than just children) of $book,
selecting only <para> element nodes that have an attribute node named class whose value is "warning"

XSLT

Overview

1. XSL stands for EXtensible Stylesheet Language, and is a style sheet language for XML documents.
2. XSLT stands for XSL Transformations.
3. XSLT to transform XML documents into other formats, like XHTML.

Elements

Element	Description
xsl:apply-imports	Invokes an overridden template rule.
xsl:apply-templates	Directs the XSLT processor to find the appropriate template to apply, based on the type and context of each selected node.
xsl:attribute	Creates an attribute node and attaches it to an output element.

xsl:attribute-set	Defines a named set of attributes.
xsl:call-template	Invokes a template by name.
xsl:choose	Provides multiple conditional testing in conjunction with the `<xsl:otherwise>` element and `<xsl:when>` element.
xsl:comment	Generates a comment in the output.
xsl:copy	Copies the current node from the source to the output.
xsl:copy-of	Inserts subtrees and result tree fragments into the result tree.
xsl:decimal-format	Declares a decimal-format, which controls the interpretation of a format pattern used by the `format-number` function.
xsl:element	Creates an element with the specified name in the output.
xsl:fallback	Calls template content that can provide a reasonable substitute to the behavior of the new element when encountered.
xsl:for-each	Applies a template repeatedly, applying it in turn to each node in a set.
xsl:if	Allows simple conditional template fragments.
xsl:import	Imports another XSLT file.
xsl:include	Includes another XSLT file.

xsl:key	Declares a named key for use with the`key()` function in XML Path Language (XPath) expressions.
xsl:message	Sends a text message to either the message buffer or a message dialog box.
xsl:namespace-alias	Replaces the prefix associated with a given namespace with another prefix.
xsl:number	Inserts a formatted number into the result tree.
xsl:otherwise	Provides multiple conditional testing in conjunction with the `<xsl:choose>` element and `<xsl:when>` element.
xsl:output	Specifies options for use in serializing the result tree.
xsl:param	Declares a named parameter for use within an `<xsl:stylesheet>` element or an `<xsl:template>` element. Allows you to specify a default value.
xsl:preserve-space	Preserves white space in a document.
xsl:processing-instruction	Generates a processing instruction in the output.
msxsl:script*	Defines global variables and functions for script extensions.
xsl:sort	Specifies sort criteria for node lists selected by `<xsl:for-each>` or `<xsl:apply-templates>`.

xsl:strip-space	Strips white space from a document.
xsl:stylesheet	Specifies the document element of an XSLT file. The document element contains all other XSLT elements.
xsl:template	Defines a reusable template for generating the desired output for nodes of a particular type and context.
xsl:text	Generates text in the output.
xsl:transform	Performs the same function as `<xsl:stylesheet>`.
xsl:value-of	Inserts the value of the selected node as text.
xsl:variable	Specifies a value bound in an expression.
xsl:when	Provides multiple conditional testing in conjunction with the `<xsl:choose>` element and `<xsl:otherwise>` element.
xsl:with-param	Passes a parameter to a template.