



Introduction to XML

2004 CERN School of Computing,
Vico Equense
Andreas Pfeiffer
CERN, PH/SFT



XML

eXtensible Markup Language

Overview

- XML – what it (not) is
- XML syntax
- DTD – describing XML documents
- Related technologies

XML – what it is (not)

- XML is a markup language and *only* a markup language
 - Not a programming language !
 - No “compiler” for XML
 - Not a network protocol !
 - Not a database either !

XML – what it is

- ☛ (Meta) Data description language
 - E.g., config files
 - Extensible – you can add your own “words”
- ☛ Simple, well-documented data format
 - Truly cross-platform
 - Ideal for long-term storage
 - Text file
 - Unicode (ASCII or other) encoding

Meta Markup Language

- No fixed set of tags and elements
 - Just a couple of “special characters” which are to avoid (using escape sequences)
- Users can define their own specific “language”
 - Document Type Definition (DTD)
 - Huge flexibility, very powerful
 - Namespaces to avoid conflicts

XML syntax: Tags

- Start tags begin with "<"
- End tags begin with "</"
- Tags are closed by ">"
 - Empty elements are closed by ">"
- Case sensitive
- Names of tags reflect the type of content, not formatting information
- Comments like in HTML
 - <!-- this is a comment -->**
 - The double hyphen -- should not appear anywhere else in the comment (---> is forbidden)

XML syntax

☛ Tags, *elements*, attributes, values

```
<?xml version="1.0"?>  
<book ISBN="0596000588">  
  <title> XML in a nutshell </title>  
  <author> E.R.Harold </author>  
  <author> W.S.Means </author>  
  <publisher> O'Reilly </publisher>  
</book>
```


XML syntax: Elements

- ☞ Elements consist of
`<startTag> content <endTag>`
- ☞ Typically elements contain other elements
 - XML documents are trees
 - Parent-child relation for elements in a tree
 - Each child has exactly one parent (with the exception of the first element in the doc ('root'))

XML syntax: Attributes

- ☞ XML elements can have attributes
 - In the start tag
- ☞ Attributes are name/value pairs
 - Name = "value" (or name = 'value')
 - Not sensitive to whitespace
- ☞ Usually used for meta-data
 - E.g., ID in a database

Attributes vs. elements

- Here are some of the problems using attributes:
 - attributes cannot contain multiple values (child elements can)
 - attributes cannot describe structures (child elements can)
 - attributes are not easily expandable (for future changes)
 - attributes are more difficult to manipulate by program code
 - attribute values are not easy to test against a DTD

Attributes vs. elements (II)

- If you use attributes as containers for data, you end up with documents that are difficult to read and maintain. Try to use **elements** to describe data. Use attributes only to provide information that is not relevant to the data.
- Exception: metadata (data about data, e.g. an ID) should be stored as attributes.

XML Names

- Names in XML may contain alphanumeric (also non-english) characters plus :
 - _ Underscore,
 - Hyphen, and
 - . Period
- All other punctuation chars are not allowed
 - No whitespace allowed in names
 - Colon is reserved for namespaces
- Names may start with letters, ideograms, or _

XML special characters

- There are 5 predefined "entity references" in XML:

<	<	Less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation

Discussed later

- Entity references always start with the "&" character and end with the ";" character.
 - Note:** Only the characters "<" and "&" are strictly illegal in XML. Apostrophes, quotation marks and greater than signs are legal, but it is a good habit to replace them.

Escape Characters

- Illegal XML characters have to be replaced by entity references.
 - If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
 - You cannot write something like this:
`<message>if salary < 1000 then</message>`
 - To avoid this, you have to **replace** the "<" character with an **entity reference**, like this:
`<message>if salary < 1000 then</message>`

Escaping characters using CDATA sections

- Tells parser to interpret following data literally (“raw” character data, not interpreting ‘<’ and ‘&’ as special characters)
 - E.g., embedded HTML or other XML sources
- Section starts with “<![CDATA[”
- Section ends with “]]>”
 - Which is of course forbidden in the data

Document Type Definitions (DTDs)

- Formal syntax to describe exactly what is allowed in an XML document
 - HTML: “ul” may only contain “li”s ...
- Used for validation of XML documents
 - If required by the user
- Declares elements `<!ELEMENT ... >`
- ... and attributes `<!ATTLIST ... >`
- A DTD is not an XML document !
 - ➔ XML Schema (much more verbose, but XML)

DTD for persons

Example of a DTD

● `<!ELEMENT element_name (content_model)>`

```
<!ELEMENT person (name, profession*)>
```

```
<!ELEMENT name (firstName, lastName)>
```

```
<!ELEMENT firstName (#PCDATA)>
```

```
<!ELEMENT lastName (#PCDATA)>
```

```
<!ELEMENT profession (#PCDATA)>
```

Ordering !

*Parsed
Character
DATA*

```
<!ELEMENT response (data | fault)*>
```

*? Zero or one
+ One or more
* Zero or more*

Usage of persons DTD

- Assume the example from last slide is in a file called "persons.dtd"

```
<?xml version="1.0" standalone="no" ?>  
<!DOCTYPE person SYSTEM "person.dtd">  
<person>  
  <name>  
    <firstName>  Andreas    </firstName>  
    <lastName>   Pfeiffer   </lastName>  
    <profession> physicist </profession>  
  </name>  
</person>
```

} *Prologue*

DTD inside the document

```
<?xml version="1.0"?>
```

Standalone version !

```
<!DOCTYPE person [
```

```
  <!ELEMENT person (name, profession*)>
```

```
  <!ELEMENT name (firstName, lastName)>
```

```
  <!ELEMENT firstName (#PCDATA)>
```

```
  <!ELEMENT lastName (#PCDATA)>
```

```
  <!ELEMENT profession (#PCDATA)>
```

```
]>
```

```
<person>
```

```
  <name>
```

```
    <firstName> Andreas </firstName>
```

```
    <lastName> Pfeiffer </lastName>
```

```
    <profession> physicist </profession>
```

```
  </name>
```

```
</person>
```

Attribute Declarations

`<!ATTLIST elementName attName attType
attDefault>`

```
<!ATTLIST image source CDATA #REQUIRED  
width CDATA #REQUIRED  
height CDATA #REQUIRED  
alt CDATA #IMPLIED  
>
```

optional

Attribute types

- ☞ CDATA – text string
 - Most generic
- ☞ NMTOKEN
 - Same rules as for any XML name
- ☞ NMTOKENS
 - One or more NMTOKEN separated by whitespace
- ☞ Enumeration
 - List of possible choices
- ☞ ID
 - Must contain a name *unique* within the document
- ☞ IDREF
 - Reference to an ID type attribute of an element in the doc.
- ☞ IDREFS
 - Separated by whitespace
- ☞ ENTITY
 - Name of an unparsed entity
- ☞ ENTITIES
 - Separated by whitespace
- ☞ NOTATION
 - Rarely used

Attribute defaults

☞ #IMPLIED

- Optional, no default provided

☞ #REQUIRED

- Each instance of the element must provide a value for this attribute, no default.

☞ #FIXED

- Attribute value is constant and immutable, cannot be overwritten

☞ Default values can be provided

- `<!ATTLIST webPage protocol NMTOKEN "http">`

Entity references

- “Shorthand” notation for DTDs
 - Five pre-defined
 - < > & " '
 - You can define your own
 - Eases readability and re-use

```
<!ENTITY coord "( (x,y) | (y,x) | (th,r) | (r,th) )" >  
<!-- a polygon has at least three points -->  
<!ELEMENT polygon (&coord;, &coord;, &coord;+)>
```


Parameter entities

➤ Different from general entities

- Can be used only in DTDs (not in the docs)
- `<!ENTITY % residentialContent "address, size, rooms, baths">`
`<!ENTITY % rentalContent "rent">`
`<!ELEMENT apartment (%residentialContent;, %rentalContent;)>`

➤ Adds higher level of flexibility

- Parameter is *replaced* by text it refers to
- Can be overwritten before being used !
- Useful also for conditional inclusions

Namespaces

- Distinguish between elements and attributes from different domains
 - “title” in book-list and web-page about books
 - “set” in MathML and SVG
- Group all related elements and attributes from a given domain (or XML application)
 - Using a prefix to refer (bind) to the URI
 - `<rdf:RDF`
`xmlns:rdf=http://www.w3.org/TR/REC-rdf-syntax#>`
 - Bindings have a scope with the element (and it's children)

XML documents

☞ “Well formed”

- Correct XML syntax (tags closed, ...)

☞ “Valid” documents

- Validation against an existing DTD/Schema
- No unknown elements/attributes
- No invalid values for elements/attributes

Programming models for XML

Parsers

“Event driven” (SAX) vs. “object based” (DOM)

SAX (simple API for XML)	DOM (Document Object Model)
Doesn't store data while parsing	Constructs an in memory copy of the document
No support for writing/modifying	In-memory tree can be modified
Document data becomes available as it's parsed	Entire document must be parsed before tree is available

XML related technologies (I)

✦ XLinks

- Attribute-based syntax for hyperlinks between XML and non-XML docs
 - One-directional, multi-directional, non-intrusive

✦ XPointers

- Syntax to identify parts of an XML doc
- Used often in conjunction with an XLink

XML related technologies (II)

➤ XSLT

- XML application describing transformations from one XML document to another
- XSLT and XSL-FO (XSL Formatting Objects)

➤ XPath

- Non-XML syntax used by XPointers and XSLT to identify particular pieces of an XML doc
- From the common parts of XSLT and XPointer

XML related technologies (III)

Namespaces

- Avoid (destructive) interferences when DTDs use the same names
- Similar use than in C++ or Python modules

DOM

- Document Object Model, tree oriented API

SAX

- Simple API for XML

Other applications of XML

- More specific to concrete domains
 - W3C endorsed standards
 - Scalable Vector Graphics – SVG
 - Mathematical ML – MathML
 - Resource Description Framework – RDF
 - Other
 - Chemical ML – CML
 - One of the first XML applications
 - Channel Definition Framework – CDF
 - MS specific (publish web sites to IE)

Items for further study

- Formatting Objects (XSL-FO)
- XPath
- XLink
- XPointer
- Internationalisation
- Cascading Style Sheets
- ...

XML in HEP

- Configuration files
- Detector geometry description
 - “Standard” is evolving
- Schema for introspection and persistency
 - LCG Dictionary through gcc-xml
- Data interchange
 - AIDA XML standards for Data Analysis related items
 - Histograms (binned and unbinned),
 - Vectors of data,
 - Ntuples,
 - Functions and Fits

Links

☞ WWW consortium

- <http://www.w3.org/>
- with **lots** of further links !

☞ XML – development

- <http://www.xml.org/>



Optional slides

eXtensible Stylesheet Language (XSL)

✦ XSL Transformations (XSLT)

- XML application specifying the rules for transforming a XML document into another
- XSLT document (XSLT style sheet) contains templates used by an XSLT processor

✦ XSL Formatting Objects (XSL-FO)

- Alternative to Cascading Style Sheets (CSS)
- XML application describing the layout of text on a page
- XSL-FO usually created through a XSLT stylesheet transforming the document from native to XSL-FO

XSL Transformations

Processors for XSLT

- Internet Explorer, Cocoon (Apache)
- Stand-alone: Saxon, Xalan

Minimal XSLT stylesheet

- Removes all tags from document, only content (text) is left

```
<?xml version="1.0">  
<xsl:stylesheet version="1.0"  
    xmlns:xsl=http://www.w3.org/1999/XSL/Transform>  
</xsl:stylesheet>
```

XSLT Example: Input

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
  <catalog>
    <cd>
      <title>Empire Burlesque</title>
      <artist>Bob Dylan</artist>
      <country>USA</country>
      <company>Columbia</company>
      <price>10.90</price>
      <year>1985</year>
    </cd>
    <!-- more items here -->
  </catalog>
```

XSLT Example: StyleSheet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
    <body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th align="left">Title</th>
          <th align="left">Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
            <td> <xsl:value-of select="title"/> </td>
            <td> <xsl:value-of select="artist"/> </td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

**O
U
T
P
U
T**

Match whole document

*Select <cd> in <catalog>
(potentially sort)*

Use <title> and <artist>

End of the loop

XSLT flow control

☛ Sorting inside a loop

- `<xsl:sort select="artist"/>`

☛ Conditional execution

- `<xsl:if test="price > 10">`
`</xsl:if>`
- `<xsl:choose>`
 `<xsl:when test="price > 10">`
 `<td bgcolor="#ff00ff">`
 `<xsl:value-of select="artist"/>`
 `</td>`
 `</xsl:when>`
 `<xsl:otherwise>`
 `<td>`
 `<xsl:value-of select="artist"/>`
 `</td>`
 `</xsl:otherwise>`
`</xsl:choose>`

Applying templates (I)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
      <html>
        <body>
          <h2>My CD Collection</h2>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>

    <xsl:template match="cd">
      <p>
        <xsl:apply-templates select="title"/>
        <xsl:apply-templates select="artist"/>
      </p>
    </xsl:template>
```

Applying templates (II)

```
<xsl:template match="title">  
  Title: <span style="color:#ff0000">  
    <xsl:value-of select="."/></span>  
  <br />  
</xsl:template>
```

```
<xsl:template match="artist">  
  Artist: <span style="color:#00ff00">  
    <xsl:value-of select="."/></span>  
  <br />  
</xsl:template>
```

```
</xsl:stylesheet>
```