Introduction to XML Technologies

Angel Gruev
Dreamix ltd.

Introduction to XML

Markup Language

- A markup language annotates (or marks up) a text.
- A markup language must specify
  - What markup is allowed
  - What markup is required
  - How markup is to be distinguished from text
  - What the markup means

**XML only specify the first three, the fourth is specified by DTD or Schema*

What is XML?

- XML is a markup meta-language:
  - It enables you to define your own language
  - It establishes rules about how you can mark up a document
  - You can use it to extend your language if you feel the need

  Note: HTML is a markup language that does not obey the rules of XML
- Practical definition: XML consists of a set of standards to exchange and publish information in a structured manner

Dealing with XML

- In what way do you get involved with XML?
  - Many configuration files are stored in XML format.
  - XML text is routinely sent and received across the Internet.
  - You can generate XML with server-side programs.
  - You can parse XML to access the information.
  - You can query XML.
  - You can transform XML.

XML Application – Exchange data

- XML is used to Exchange Data
  - Text format
  - Software-independent, hardware-independent
  - Exchange data between incompatible systems, given that they agree on the same tag definition.
  - Can be read by many different types of applications
- Benefits:
  - Reduce the complexity of interpreting data
  - Easier to expand and upgrade a system

XML Application 2 – Store Data

- XML can be used to Store Data
  - Plain text file
  - Store data in files or databases
  - Application can be written to store and retrieve information from the store
  - Other clients and applications can access your XML files as data sources
- Benefits:
  - Accessible to more applications

XML Application 3 – Create new language

- XML can be used to Create new Languages
  - WML (Wireless Markup Language) used to markup Internet applications for handheld devices like mobile phones (WAP)
  - MusicXML - publish musical scores
  - ThML - Theological Markup Language
  - CML - Chemical Markup Language
  - MathML - Mathematical Markup Language

General Construction Plan for an XML Document

Prolog

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml:stylesheet type="text/css" href="s.css" ?>
<!DOCTYPE  test SYSTEM "test.dtd">
stylesheet and document type declaration
```

Instance

```
Elements (element hierarchy)
<author>
        <first>Ganyo</first>
        <last>Balkanski</last>
</author>
```

Additions

- **Comments** `<!-- comment -->`
- **Processing Instructions**
  `<?PITarget Status="draft" ?>`

Escaping characters

- Predefined entities for special characters

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

CDATA

- CDATA refers to character data.
- CDATA (Character DATA) comes from SGML, too
  - Starts with <![CDATA[
  - Ends with]]>
- CDATA contents are ignored by the parser and are given *as-is* to the application

Elements or Attributes?

- How should data be encapsulated?
  -
```
<book>
  <title>The Forty-nine Steps</title>
  …
</book>
```
  -
```
<book title="The Forty-nine Steps">
  …
</book>
```
- Depends upon what document type is designed for

Elements or Attributes ?

- General rules:

If all markup is stripped away, the document should still be readable and useable

If in doubt, use an element

Elements or Attributes?

- Attributes versus elements:
  - Elements can be nested; attributes cannot be.
  - Metadata is usually stored as attributes (developers may disagree on what data is to considered metadata).
  - When in doubt, use elements; when XML documents are transformed, some transformers may ignore attributes.

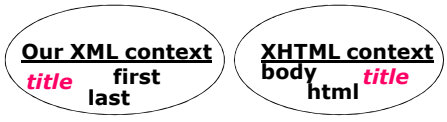
Processing Instructions

- Information required by an external application
- Processing Instructions
  - Format
    - `<? … ?>`
  - XML PI
    - `<?xml version='1.0' ?>`
    - Confusingly, this is called the XML declaration, but is a processing instruction

XML Namespaces

Avoiding duplication of names

- People create their XML dictionaries
- Many name duplications, with different semantics
- Example:



Example: duplication of <title>

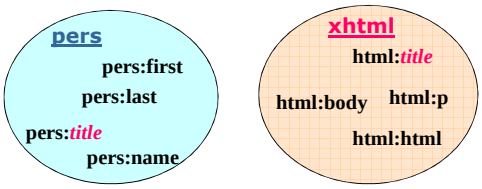
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person>
  <name id="1">
    <title>Sir</title>
    <first>John</first>
    <middle>Fitzgerald Johansen</middle>
    <last>Doe</last>
  </name>
  <resume>
    <html>
      <head><title>Resume of John Doe</title></head>
      <body>
        <h1>John Doe</h1>
        <p style="FONT-FAMILY: Arial">
          John's a great guy, you know?
        </p>
      </body>
    </html>
  </resume>
</person>
```

Example: duplication of <title>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pers:person xmlns:pers="http://sernaferna.com/pers"
             xmlns:html="http://www.w3.org/1999/xhtml">
  <pers:name id="1">
    <pers:title>Sir</pers:title>
    <pers:first>John</pers:first>
    <pers:middle>Fitzgerald Johansen</pers:middle>
    <pers:last>Doe</pers:last>
  </pers:name>
  <pers:resume>
    <html:html>
      <html:head>
        <html:title>...</html:title>
      </html:head>
      <html:body>
          ...
      </html:body>
    </html:html>
  </html:resume>
</html:person>
```

The Idea of Namespaces

- Namespaces
  - Abstract notion (category) for groups of names
  - A name belong to one group only



Presentation

- Elements have to have Qualified Names (called QNames)
- QName = namespace_prefix : local_name
- See http://www.w3.org/TR/1999/REC-xml-names
- *But how to assure unique names for a given namespace?? (two companies may use different 'pers' namespaces)*

Unique Namespace Identification by URL

xmlns:namespace-prefix="namespaceURI"

- Most standards have an official URL
  - A URL is unique
- Use URL for namespace identification
  - Application doesn't have to be Internet-aware to use Namespaces
    - URL becomes a well-known text string
- Distinguish elements and attributes by adding a prefix to the name
  - `prefix:name` (assign short unique name to URL)

Namespaces in XML

- Specification of W3C
  - A single schema (DTD) is considered to own its own namespace in which all element names and all attribute names are unique
  - Mechanism for identifying namespaces used in document and identifies to which namespace an element or attribute belongs
    - Any reference to an element name is unambiguous
    - A single document may contain information defined in a number of namespaces

Default Namespace – xmlns without any Prefix

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person xmlns="http://sernaferna.com/pers"
        xmlns:html="http://www.w3.org/1999/xhtml">
  <name>
    <title>Sir</title>
    <first>John</first>
    <middle>Fitzgerald Johansen</middle>
    <last>Doe</last>
  </name>
  <position>Vice President of Marketing</position>
  <résumé>
    <html:html>
      <html:head><html:title>Resume of John Doe</html:title></html:head>
      <html:body>
      <html:h1>John Doe</html:h1>
      <html:p>John's a great guy, you know?</html:p>
      </html:body>
    </html:html>
  </résumé>
</person>
```

Default NS

Namespaces for Attributes, too

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person xmlns="http://sernaferna.com/pers">
  <name id="1">
    <title>Sir</title>
    <first>John</first>
    <middle>Fitzgerald Johansen</middle>
    <last>Doe</last>
  </name>
  <position>Vice President of Marketing</position>
  <résumé>
    <html:html xmlns:html="http://www.w3.org/1999/xhtml">
    <html:head><html:title>Resume of John Doe</html:title></html:head>
    <html:body>
    <html:h1>John Doe</html:h1>
    <html:p html:style="FONT-FAMILY: Arial">
      John's a great guy, you know?
    </html:p>
    </html:body>
    </html:html>
  </résumé>
</person>
```

Uniqueness of Attributes

- In XML documents conforming to this specification, no tag may contain two attributes which:
  - Have identical names
  - Have qualified names with the same local part and with prefixes which have been bound to namespace names that are identical.

Uniqueness of Attributes Examples

```
<!-- http://www.w3.org is bound to n1 and n2 -->
<x xmlns:n1="http://www.w3.org"
   xmlns:n2="http://www.w3.org" >
  <bad a="1"     a="2" />
  <bad n1:a="1"  n2:a="2" />
</x>
```

```
<!-- http://www.w3.org is bound to n1
     and is the default -->
<x xmlns:n1="http://www.w3.org"
   xmlns="http://www.w3.org" >
  <good a="1"     b="2" />
  <good a="1"     n1:a="2" />
</x>
```

XML Validation – DTD, XSD

www.dreamix.eu

What is a DTD?

- A template for document markup
  - A file which contains a formal definition of a particular type of document
- A DTD describes:
  - What names can be used for element types
  - Where element types can occur
  - How element types fit together
  - Specifies document hierarchy and granularity
  - Specifies names and types of element attributes

Why have a DTD?

- Validating XML parser
  - Check the structure of the XML file against a DTD
  - Check that it is valid and well-formed
- DTD
  - Can be a mechanism for standardization
  - And hence document/data manipulation and exchange

General DTD Rules

- Each tag used in a valid XML document must be declared with an element declaration.
- An element declaration specifies the name and the contents of the element (if any).
- Element specifications are made using a simple grammar.
- The order of element declarations is not relevant.

General DTD Rules (2)

- DTD is case sensitive, but spacing and indentation in DTDs are not significant.
- Comments in DTDs are just like comments in XML (they may not appear inside an element declaration).
- Everything not explicitly permitted is forbidden.

DTD Declaration

- DTD syntax is stored either in an external file, in the XML file itself, or both
  - *Internal DTD overrides or adds to the external in cases of ENTITY and ATTLIST repetition*
- DTD parts
  - ELEMENT - Tag definition
  - ATTLIST - Attribute definitions
  - ENTITY - Define common text
  - PCDATA
  - CDATA

Internal DTD

- DTD may be specified internally as a document type declaration following the XML declaration:
  - `foo` refers to the root element.
  - `foo` contains parsed character data (`#PCDATA`) and may not contain other elements.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE foo [
  <!ELEMENT foo (#PCDATA)>
]>
<foo>
  Hello XML!
</foo>
```

External DTD

- External DTDs
  - May be on a file system or a Web site
  - Are often shared
  - Are specified at the beginning of an XML document

```
<!doctype html PUBLIC "-//w3c//dtd html 4.0
  transitional//en">
```

What is Schema?

- 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

Can DTD Do the Job?

- DTD is written with yet another syntax. XSD schemas are well-formed XML documents.
- DTD offers limited data types.
- DTD has a complex and fragile extension mechanism.
- XML schema overcomes the above limitations and offers a richer, expressive constraint-specifying mechanism.
- XML schema files are stored with the .xsd extension.

Simple XML

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

Simple XML with DTD

XML Doc

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

DTD

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

Simple XML with XSD

```
<?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>
```

Referencing a Schema in an XML Document

```
<?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">

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

Referencing (2)

- Default namespace declaration

```
xmlns="http://www.w3schools.com"
```

- **XML Schema Instance namespace**

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

- **schemaLocation attribute**

```
xsi:schemaLocation="http://www.w3schools.com/note.xsd"
```

XSD Simple Elements

- What is simple element
  - Contain only text (cannot contain elements or attributes)
  - The text can be of different types
    - Common types: boolean, string, date, integer, time, decimal
    - Custom types
  - Restriction (facets) can be added

Define a Simple Element

- Syntax

```
<xs:element name="xxx" type="yyy"/>
```

- Example:

Schema

```
<xs:element name="lastname"
            type="xs:string"/>
<xs:element name="age"
            type="xs:integer"/>
<xs:element name="dateborn"
            type="xs:date"/>
```

XML

```
<lastname>
   Refsnes
</lastname>
<age>34</age>
<dateborn>
   1968-03-27
</dateborn>
```

XSD Attributes

- What is an Attribute?
  - declared as simple types
  - only complex elements can have attributes
- Define an Attribute

```
<xs:attribute name="xxx" type="yyy"/>
```

  - default and fixed attributes
  - use attribute : optional and required

```
<xs:attribute name="lang"  type="xs:string"
              default="EN“ use=“required”/>
```

Restrictions (facets)

- Data types restrictions
- User defined restrictions (facets)
  - Restrictions on Values
  - Restrictions on a Set of Values
  - Restrictions on a Series of Values
  - Restrictions on White Space Characters
  - Restrictions on Length

Facets – Example

Example 1

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Example 2

Fasets – Example [2]

Example 3

```
<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

XSD Complex Elements

- What is a Complex Element?
- There are four kinds of complex elements:
  - empty elements
  - elements that contain only other elements
  - elements that contain only text
  - elements that contain both other elements and text

Example

```
<product pid="1345"/>
```

```
<employee>
<lastname>Smith</lastname>
</employee>
```

How to Define a Complex Element?

```
<employee>
  <firstname>
    John
  </firstname>
  <lastname>
    Smith
  </lastname>
</employee>
```

```
<xs:element name="employee">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
 <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Complex Types

- Complex type for empty elements

```
<product prodid="1345"/>
```

```
<xs:complexType>
  <xs:attribute name="prodid"
                type="xs:integer"/>
</xs:complexType>
```

- Complex type for elements only
- Complex type for text only
  - *<xs:simpleContent>*
- Complex type for mixed contend
  - *<xs:complexType mixed="true">*

Ref Attribute

- To use an existing element or attribute rather than declaring a new element or attribute
- Existing element must be global element - an element that is declared under root element

Ref Example

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element ref="age"/>
    </xs:sequence>
  </xs:complexType>
</xs:element />
```


XSD Complex Types Indicators

- Controls HOW elements are to be used in documents.
- We have seven types of indicators:
  - Order indicators:
    - All, Choice, Sequence
  - Occurrence indicators:
    - maxOccurs, minOccurs
  - Group indicators:
    - Group name, attributeGroup name

Order Indicators (1)

- <xs:all>
  - child elements can appear in any order
  - each child element must occur once and only once

```
<xs:complexType>
  <xs:all>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:all>
</xs:complexType>
```

Order Indicators (2)

- Choice Indicator
  - Specifies that either one child element or another can occur

```
<xs:choice>
  <xs:element name="employee" type="employee"/>
  <xs:element name="member" type="member"/>
</xs:choice>
```

- Sequence Indicator
  - Specifies that the child elements must appear in a specific order

Occurrence Indicators

- maxOccurr
  - Maximum number of times an element can occur
- minOccurr
  - Minimum number of times an element can occur

```
<xs:element name="child_name" type="xs:string"
        minOccurs="0" maxOccurs="5"/>
```

Extensible Documents

- How to Allow documents to contain additional elements that are not declared in the main XML schema ?
- The <*any*> Element
  - enables us to extend the XML document with elements not specified by the schema
- The <*anyAttribute*> Element
  - enables us to extend the XML document with attributes not specified by the schema

Xpath Language

www.dreamix.eu

XPath

- A scheme for locating in documents and identifying sub-structures within them
- Used by other XML specifications
  - XPointer, XQL, XSLT

Element Context

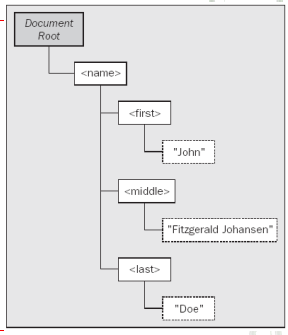
- Meaning of element can depend upon its context

```
<book><title>…</title></book>
<person><title>…</title></person>
```

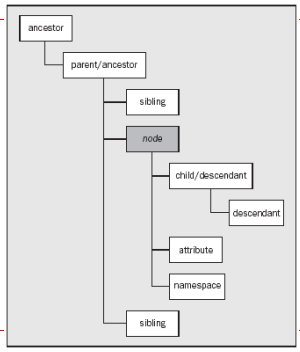
- Want to search for, e.g. title of book, not title of person
  - XPath exploits sequential and hierarchical context of XML to specify elements by their context (i.e. location in hierarchy)
    - `title` `book/title` `person/title`
- Context node (noted as '.') – the XML section where we begin a XPath expression

What is a XPath Node?

- XPath Node – any part of a document; can be:
  - Element
  - Attribute
  - Processing-instruction
  - Text
  - Namespace
  - Comment
  - Document (root)



Relationships - Treeview



XPath Expressions

- A text string to select an element, attribute, processing instructions, or text
- Expression may appear in URL or attribute
  - http://abc.com/getQuery?/book/title
- 'Node' used to describe anything significant in XML that can be selected

```
<xsl:pattern match="chapter/title">…</xsl:pattern>
```

Location Path

- Expressions identify items by their location in the XML hierarchy
- Location path may
  - dig down structure
  - skip over siblings
  - go up the structure
- Location path may be:
  - Relative – *starts from the context node*, or
  - Absolute - *starts from the root node*

Location Path Expression

- An absolute location path:
  - */step/step/...*
- A relative location path:
  - *step/step/...*

Note: Every step is separated by (/)

Selecting Nodes

Expression	Description
nodename	Selects all child nodes of the node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Path Expressions - Example

```
<?xml version="1.0" encoding="utf-8" ?>
<books>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
</books>
```

```
/books
```

Path Expressions - Example

```
<?xml version="1.0" encoding="utf-8" ?>
<books>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
</books>
```

```
/books/book  or  //book
```

Path Expressions - Example

```
<?xml version="1.0" encoding="utf-8" ?>
<books>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
</books>
```

```
//@lang
```

Predicates

- Predicates are used to find a specific node or a node that contains a specific value
- Predicates are always embedded in square brackets - []

Path Expressions - Example

```
<?xml version="1.0" encoding="utf-8" ?>
<books>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
</books>
```

```
/books/book[1]
```

Path Expressions - Example

```
<?xml version="1.0" encoding="utf-8" ?>
<books>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
</books>
```

```
/books/book[last()]
```

Path Expressions - Example

```
<?xml version="1.0" encoding="utf-8" ?>
<books>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
</books>
```

```
//title[@lang='eng']
```

Path Expressions - Example

```
<?xml version="1.0" encoding="utf-8" ?>
<books>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
  <book>
    <title lang="en">…</title>
    <author>…</author>
    <pages>…</pages>
  </book>
</books>
```

```
//book/title | //book/author
```

Axis

- An axis defines a *node-set* relative to the current node.
  - */child::A/child::B*
  - */A/B* (short version)

```
<A>
  <B></B>
  <C></C>
  <B></B>
<A>
```

- There are 13 axis
- The general syntax for a step is

axisname::nodetest[predicate]

Wildcards

- Sometimes we don't or can't know names
  - Can use wildcard '*' for any single element
    - `book/intro/title` and `book/chapter/title` are matched by `book/*/title` (but so is `book/appendix/title`)
  - @* - matches any attribute
  - node() - Matches any node of any kind
  - Multiple asterisks can match several levels
    - But attention for inappropriate matches!

Attribute Tests

- Attributes can be selected
  - feature/@type
- Elements can be selected dependant upon attribute value
  - feature[@type="exon"]

String Tests

- Strings can be tested for characters and substrings
  - `<note>hello there</note>`
    - `note[contains(text(), "hello")]`
  - `<note><b>hello</b> there</note>`
    - `note[contains(., "hello")]`
  - The '.' is current node, and will go through all children

XSL Transformation Language (XSLT)

DEMO