

An Introduction to XML

Part 1

Selim Mimaroglu

This presentation is built on XML Bible 1.1 by Elliotte Rusty
Harold

Road Map

- XML
- Well Formedness
- Validity
- DTD
- Namespace
- XML Schema

Part 1

- XML is a meta-markup language
- XML stands for Extensible Markup Language
- XML is not a programming language:
 You can't add two numbers in XML
- XML is self describing language
- XML is text (I will repeat this)

Important

- XML is *de facto* standard for data exchange

I think

- XML will be *de facto* standard for data storage in a few years. It's more powerful than: Relational Databases, Object Oriented Databases, Network Databases, Hierarchical Databases

Hello XML!

```
<?xml version="1.1"  
  encoding="UTF-8"  
  standalone="yes" ?>
```

```
<FOO>Hello XML!</FOO>
```

Meaning

- Markup can indicate three kind of meaning: structural, semantic or stylistic.
- *Structure* specifies the relations between the different elements in the document
- *Semantics* relates the individual elements to the real world outside of the document itself
- *Style* indicates how an element is displayed

- *Structure* comes with the data. By looking at the data you can find the structural relationships.
- *Semantic* meaning exists **outside** of the document, in the mind of author or reader. An English-speaking human would be more likely to understand `<GREETING>` and `</GREETING>` or `<DOCUMENT>` and `</DOCUMENT>` than `<FOO>` and `</FOO>`.
- *Style* can be attached to an XML document through XSL (XML Style Sheet Language) or CSS (cascading style sheets).

XMLizing the Data

- You can turn almost any kind of data in XML. It's obvious how to convert a book into an XML: Chapters, Sections, Sub sections
- It may be harder to convert a GIF image into XML:

```
<PIXEL x="12" y="-256"  
color="HBE542"/>
```

- It's believed that 80% of today's data resides in relational databases. How would you XMLize this data?
- Use SQLXML. It's part of SQL standard from ISO. This implies that 80% of today's data can be converted to XML in no time, because Oracle, IBM, and Microsoft support and implement SQLXML standards.

Advantages of XML Format

- The data is self describing
- The data can be manipulated with standard tools
- The data can be viewed with standard tools
- Different views of the same data are easy to create with style sheets

<CAST>

<ACTOR GIVEN_NAME="Don"
SURNAME="Rickles" />

<ACTOR GIVEN_NAME="Jerry"
SURNAME="Springer" />

<ACTOR GIVEN_NAME="Richard"
SURNAME="Simmons" />

<ACTOR GIVEN_NAME="Vicki"
SURNAME="Lawrence" />

<ACTOR GIVEN_NAME="John"
SURNAME="Salley" />

<ACTOR MIDDLE_NAME="Kennedy" />

</CAST

Well formedness

Well-formedness is the minimum criterion necessary for XML processors and browsers to read files.

1. An XML document is made of up text
2. If an XML document does include an XML declaration, this declaration must be the first thing in the file (not even white space):

```
<?xml version="1.0" encoding="ASCII"  
standalone="no" ?>
```

3. Element Names (has to agree some rules, case sensitive)
4. Every start-tag must have a corresponding end-tag:
`<department>Computer Science</department>`
5. Or you may use Empty Element Tags
`<department name="Computer Science" />`
6. Elements may nest but not overlap

7. Attributes:

- Attributes are name-value pair separated by equals sign (=)
- Must obey name rules
- Attribute values are strings (May be confusing if you know something about XML Schema).

8. XML Comments are:

`<!-- Comment goes here -->`

Comments don't go inside tags.

9. Processing Instructions begin with <? and end with ?>

<?xml-stylesheet type="text/xml" href="5-2.xsl"?>

XML 1.1

Latest version of XML is 1.1. XML 1.0 was based on Unicode 2.0 XML 1.1. is independent of any particular Unicode version. You can't write XML 1.0 names in Amharic, Burmese or Cambodian because those scripts weren't added to Unicode until version 3.0

DTD (Document Type Definition)

- A *document type definition* lists the elements, attributes, entities and notations that can be used in a document, as well as their possible relationships to one another. A DTD specifies a set of rules.
- XML Schemas are more powerful than DTDs, however there are things that can be done with DTDs and not with XML Schemas. XML Schemas are more popular than DTDs. DTDs are still the choice for representing Document centric XML Data

A DTD can be included in the XML document it describes, or that document can link to it at an external URL

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

The GREETING element contains PCDATA
(Parsed Character Data)

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

```
<GREETING>Hello XML!</GREETING>
```

Or even this is valid according to the DTD we have:

```
<GREETING></GREETING>
```

Document Type Declaration

- Do NOT confuse it with the DTD (Document Type Definition).
- A document type declaration is placed in an XML document's prolog to say what DTD that document adheres to.
- It also specifies which is the **root element**.
- The document type declaration can either specify the DTD **directly**, by **including it inside**
- Or **indirectly** by giving the **URL**
- Or **BOTH**

Example that includes a DTD

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

```
<!DOCTYPE GREETING [
```

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

```
]>
```

```
<GREETING>Hello XML!</GREETING>
```

Example that includes a DTD and a URL:

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

```
<!DOCTYPE DOCUMENT SYSTEM
```

```
  "greeting.dtd" [
```

```
    <!ELEMENT DOCUMENT (GREETING,  
    DATE)>
```

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

```
]>
```

```
<DOCUMENT>
```

```
  <GREETING>Hello</GREETING>
```

```
  <DATE>January 10, 2004</DATE>
```

```
</DOCUMENT>
```

greeting.dtd (has the rest of DTD)

<!ELEMENT GREETING (#PCDATA)>

ANY

`<!ELEMENT SCHEDULE ANY>`

This says that all possible elements as well as plain text can be children of the SCHEDULE element

PCDATA

We have seen this before:

`<!ELEMENT GREETING (#PCDATA)>`

Stands for Parsed Character Data

There are two basic kinds of elements in XML. **Simple** elements can only contain plain text. They can't have any child elements. **Complex** elements can contain other elements or both plain text and other elements.

Child Elements

You can add additional children in their proper order, separated from each other by commas. Here is a complete declaration for the SCHEDULE element:

<!ELEMENT SCHEDULE (DATE,
STATION, STATION, STATION)>

This form of content model is called a sequence. This says that each SCHEDULE element should contain exactly one DATE child element, followed by exactly three STATION elements.

Similar Syntax with Regular Expressions

+ One or More Children

? Zero or One Child

* Zero or More Children

| Choice

Parentheses for grouping things

Empty Elements

It's occasionally useful to define an element that has no content.

```
<!ELEMENT IMG EMPTY>
```

For example:

```
<!ELEMENT STATION (((NETWORK,  
  CALL_LETTERS?) | CALL_LETTERS),  
  CHANNEL, SHOW+)>
```

```
<!ELEMENT ACTOR ((GIVEN_NAME,  
  (MIDDLE_NAME | MIDDLE_INITIAL)*,  
  SURNAME?) | ((MIDDLE_NAME |  
  MIDDLE_INITIAL)+, SURNAME) |  
  SURNAME)>
```

Attribute Declarations (in DTD)

Here is the syntax for declaring attributes using DTD

```
<!ATTLIST Element_name Attribute_name Type  
         Default_value>
```

Attribute Types:

CDATA: Character Data

Enumerated: A list of possible values from which exactly one will be chosen

ID: A unique name not shared by any other attribute

IDREF: The value of an ID type attribute of an element

IDREFS: Multiple IDs of elements separated by white space

ENTITY: The name of an unparsed entity declared in the DTD

ENTITIES: Multiple names of ENTITY

NMTOKEN: An XML name token (legal XML names)

NMTOKENS: Multiple XML name tokens

NOTATION One or more names of notations declared in the DTD

<!ELEMENT GREETING (#PCDATA)>

<!ATTLIST GREETING LANGAUGE CDATA
“English”>

<GREETING LANGAUGE=”French”>

Salut!

</GREETING>

<GREETING>

Hello!

</GREETING>

At the second example even though there is no visible LANGUAGE attribute, there is the default LANGUAGE attribute “English”. **So DTD is part of the data!!!**

Alternatives to Default Attribute Values
#REQUIRED, #IMPLIED, #FIXED

Document authors are not required to actually include the fixed attribute in their tags. If they don't include the fixed attribute, a parser that reads the DTD will report the fixed value. If the fixed attribute is included in the instance document, however, it must have the value indicated in the DTD, it's an error otherwise.

Entities

- Logically speaking, an XML document is composed of a prolog followed by a root element that strictly contains all other elements; but physically the content of an XML document can be spread across multiple files.
- The storage units that contain particular parts of an XML document are *entities*. An entity can be a file, a database record, or any other item that contains data.

There are two types of entities:

- Internal Entities `<!ENTITY name "replacement text">`
- External Entities

Internal Entity:

You can think of an internal general entity reference as an abbreviation for commonly used text or text that's hard to type.

For example, instead of typing the same footer at the bottom of every page, you can simply define that text as an internal entity and use it at the bottom of each page.

<?xml version="1.0"?>

<!DOCTYPE DOCUMENT [

<!ENTITY ERH "Elliote Rusty Harold">

<!ELEMENT DOCUMENT (TITLE, SIGNATURE)>

<!ELEMENT TITLE (#PCDATA)>

<!ELEMENT COPYRIGHT (#PCDATA)>

<!ELEMENT EMAIL (#PCDATA)>

<!ELEMENT LAST_MODIFIED (#PCDATA)>

<!ELEMENT SIGNATURE (COPYRIGHT, EMAIL, LAST_MODIFIED)>

]>

<DOCUMENT>

<TITLE>&ERH;</TITLE>

<SIGNATURE>

<COPYRIGHT>2004 &ERH;</COPYRIGHT>

<EMAIL>elharo@metalab.unc.edu</EMAIL>

<LAST_MODIFIED>July 30, 2004</LAST_MODIFIED>

</SIGNATURE>

</DOCUMENT>

You can define any entity you want; there are only five predefined entities in XML

& **&**

< **<**

> **>**

" **“**

' **‘**

External General Entities

As mentioned earlier an XML file may be composed of several entities. These entities may exist in different locations.

```
<!ENTITY name SYSTEM “URI”>
```

signature.xml

```
<COPYRIGHT>2004 Elliotte Rusty Harold</COPYRIGHT>
```

```
<EMAIL>elharo@metalab.unc.edu</EMAIL>
```

```
<LAST_MODIFIED>July 30, 2004</LAST_MODIFIED>
```

```
<HR/>
```

```
<!DOCTYPE DOCUMENT [  
  <!ELEMENT DOCUMENT  
    (TITLE, COPYRIGHT, EMAIL, LAST_MODIFIED,  
    HR?)>  
  <!ELEMENT TITLE (#PCDATA)>  
  <!ELEMENT COPYRIGHT (#PCDATA)>  
  <!ELEMENT EMAIL (#PCDATA)>  
  <!ELEMENT HR EMPTY>  
  <!ELEMENT LAST_MODIFIED (#PCDATA)>  
  <!ENTITY SIGNATURE SYSTEM  
    "http://www.cs.umb.edu/~smimarog/signature.xml"> ]>  
<DOCUMENT>  
  <TITLE>Entity references</TITLE>  
  &SIGNATURE;  
</DOCUMENT>
```

If you open it in IE that's what you see

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE DOCUMENT (View Source for full
doctype...)>
<DOCUMENT>
  <TITLE>Entity references</TITLE>
  <COPYRIGHT>
    2004 Elliotte Rusty Harold
  </COPYRIGHT>
  <EMAIL>elharo@metalab.unc.edu</EMAIL>
  <LAST_MODIFIED>
    July 30, 2004
  </LAST_MODIFIED>
  <HR />
</DOCUMENT>
```

Personal Note: Even though I am going to point research issues later. Research on this particular topic has been very short. People **assume** that an XML is standalone. Guess what, it is not!!

Namespaces

When **mixing and matching** elements from different XML applications, you are likely to find the same name used for two different things.

Namespaces are solution. They allow each element and attribute in a document to be placed in a different namespace mapped to a particular URI.

Namespace Example

```
<P:PERSON xmlns:P="http://ns.cafeconleche.org/people/">  
  <P:TITLE>Mr.</P:TITLE>  
  <P:FIRST>Mikhail</P:FIRST>  
  <P:LAST>Ovitsky</P:LAST>  
  <P:COMPANY>Duplicative Artists  
  Mismanagement</P:COMPANY>  
  <P:ADDRESS>  
    135 Agents Row, Hollywood, CA 90123  
  </P:ADDRESS>  
</P:PERSON>
```

Another Namespace Example

```
<P:PERSON xmlns:P="http://ns.cafeconleche.org/people/"  
          xmlns:CS="http://cs.umb.edu/~smimarog" >  
  <P:TITLE>Mr.</P:TITLE>  
  <P:FIRST>Mikhail</P:FIRST>  
  <P:LAST>Ovitsky</P:LAST>  
  <CS:COMPANY>Duplicative Artists  
  Mismanagement</CS:COMPANY>  
  <P:ADDRESS>  
    135 Agents Row, Hollywood, CA 90123  
  </P:ADDRESS>  
</P:PERSON>
```

Default Namespace Example

```
<PERSON xmlns="http://ns.cafeconleche.org/people/">  
  <TITLE>Mr.</TITLE>  
  <FIRST>Mikhail</FIRST>  
  <LAST>Ovitsky</LAST>  
  <COMPANY>Duplicative Artists  
    Mismanagement</COMPANY>  
  <ADDRESS>  
    135 Agents Row, Hollywood, CA 90123  
  </ADDRESS>  
</PERSON>
```