XML Theory

What is XML ?

XML Extensible Markup Language is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services.

Several schema systems exist to aid in the definition of XML-based languages, while programmers have developed many application programming interfaces (APIs) to aid the processing of XML data.

The following is an example of an XML document:

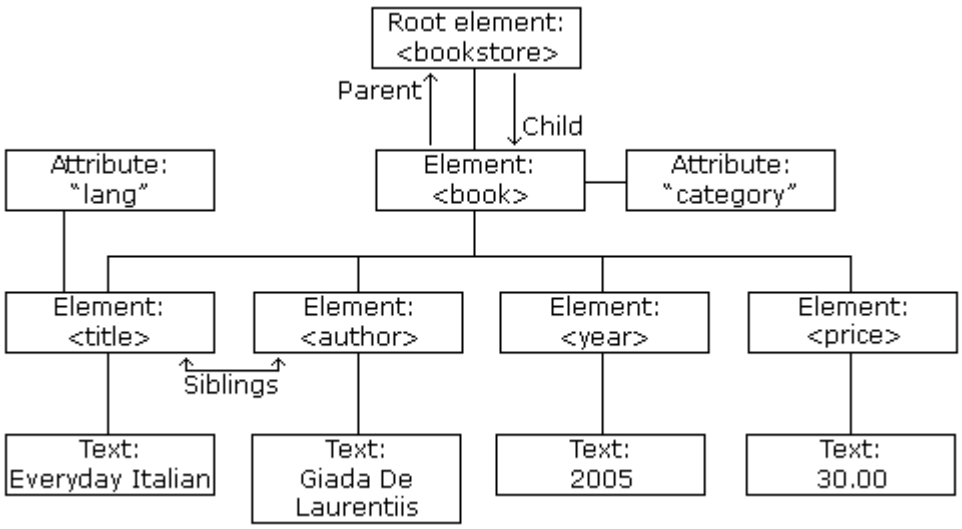
```
<?xml version= "1.0" encoding="ISO-8859-1"?>
<note>
<to>Dick</to>
<from>Jane</from>
<heading>Notice</heading>
<body>See Spot run!</body>
</note>
```

XML DOM

The W3C **DOM** Document Object Model is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.

The DOM defines a standard for accessing and manipulating documents:

- **The HTML DOM** defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.
- **The XML DOM** defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.



The XML DOM

All XML elements can be accessed through the XML DOM.

The XML DOM is:

1. A standard object model for XML;

2. A standard programming interface for XML;
3. Platform- and language-independent;
4. A W3C standard.

In other words: The XML DOM is a standard for how to get, change, add, or delete XML elements.

XML Characteristics

XML has several important characteristics:

XML is a structured format

which means that we can define exactly how the data is to be arranged, organized and expressed within the file. When we are given a file, we can validate that it conforms to a specific structure, prior to importing the data.

As we know the structure of the file in advance, we know what it contains and how to process each item. Prior to XML, the only structure in a text file was positional – we knew the bit of text after the fourth comma should be a date of birth – and we had no way to validate whether it was a date of birth, or even a date, or whether it was in day/month/year or month/day/year order.

XML is a described format

which means that within the text file, every item of data has a name that is both human- and machine-readable as well as being uniquely identifiable. We can open these files, read their contents and understand the data they contain, without having to refer back to another document to find out what the text after the fourth comma represents (and was that comma a separator, or part of the text of the second item?). Similarly, we can edit these documents with a fairly high level of confidence that we're making the correct changes.

XML can easily describe hierarchical data and the relationships between data

If we want to import and export a list of authors, with their names, addresses and the books they've written, deciding on a reasonable format for a .csv file is by no means straightforward. Using XML, we can define what an Author item is and that it has a name, address and multiple Book items. We can also define what a Book item it is and that it has a title, a publisher and an ISBN.

The hierarchy and relationships are a natural consequence of the definition.

XML can be validated

which means we can provide a second XML file – an XML Schema Definition file – that describes exactly how the XML data file should be structured. Before processing an XML file, we can compare it with the schema to ensure it conforms to the structure we expect to receive.

XML is a discoverable format

which means programs (including Excel 2003/2007/2010/2013) can parse an XML data file and infer the structure and relationships between the items. This means we can read an XML file, infer its structure and generate new XML data files that conform to the same structure, with a high degree of confidence the new XML data files will pass validation.

XML is a strongly-typed format

which means the schema definition file specifies the data type of each element. When importing the data, the application can check the schema definition to identify the data type to import it as. We no longer run the risk of the product code 01-03 being imported as a date.

XML is a global format

There is only one way to express a number in an XML file and only one way to express a date. We no longer have to check whether a .csv file was created with US or French settings and adjust our processing of it accordingly.

XML is a standard format

The way in which the content of an XML file is defined has been specified by the World Wide Web Consortium.

This allows applications (including Excel 2003/2007/2010/2013) to read, understand and validate the structure of an XML file and create files that conform to the specified structure. It also allows different applications to read, write, understand, and validate the same XML files, allowing us to share data between applications in an extremely robust manner.

XML Syntax

In this chapter, we will discuss the simple syntax rules to write an XML document. XML documents that bind to the syntax rules are said to be “well formed”.

XML syntax rules are very simple and very strict. For this reason, creating software that can read and manipulate XML is very easy to do.

All XML documents must have a root element

There must be one and only root element, on each XML document (XSD instances), into which other elements may be nested

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE example [<!ENTITY copy "&#xA9;">]>

 <rootElement attribute="xyz">
   <contentElement/>
 </rootElement>
```

The prolog line is optional and always at first on

```
<?xml version="1.0" encoding="UTF-8"?>
```

All XML elements must have a closing tag

All element tags must have a closing tag, except the prolog that is not part of the document

Incorrect:

```
<body>See Spot run.<body>See Spot catch the ball.
```

Correct:

```
<body>See Spot run.</body><body>See Spot catch the ball.</body>
```

XML tags are case sensitive

When you createXML documents the tag <Body> is different from the tag <body>

Incorrect:

```
<Body>See Spot run.</body>
```

Correct:

```
<body>See Spot run.</body>
```

All XML elements must be properly nested

XML elements must be properly nested, and the inner ones must be closed before the outer ones, in the correct sequence.

Incorrect:

```
<b><i>This text is bold and italic.</b></i>
```

Correct:

```
<b><i>This text is bold and italic.</i></b>
```

Attribute values must always be quoted

It is illegal to omit quotation marks around attribute values. XML elements can have attributes in name/value pairs; however the attribute value must always be quoted.

Incorrect:

```
<?xml version= "1.0" encoding="ISO-8859-1"?><note
date=05/05/05><to>Dick</to><from>Jane</from></note>
```

Correct:

```
<?xml version= "1.0" encoding="ISO-8859-1"?><note
date="05/05/05"><to>Dick</to><from>Jane</from></note>
```

Special symbols

Entity references are useful for text-characters that would interfere with the sintax, such as '<', '>', '&', apostrophe and quotes. Use entity references for them instead. The pre-defined ones are:

- ➔ < for <
- ➔ > for >
- ➔ & for &
- ➔ ' for '
- ➔ " for "

White-spaces are not truncated to one, differently from HTML.

New line is represented as line feed, with no carriage return.

Comments

Use <!-- and --> for comments

Note: "--" between a comment not allowed

XML documents are made of elements, which are the whole thing between the opening until the end tag, both inclusive:

```
<client id="50">John Smith</client>
```

An element may contain text, attributes, nested elements or a mix of these. It can also be empty, with or without attributes.

```
<client id="50">
<name>John Smith</name>
<dependent></dependent> or <dependent />
</client>
```

Element naming:

- Element names are case-sensitive;
- Element names must start with a letter or underscore;

- Element names can't start with the letters «xml» (or XML, or Xml, etc);
- Element names can contain letters, digits, hyphens, underscores, and periods;
- Element names can't contain spaces.
- Any name can be used, no words are reserved except "xml";
- Element naming style (free to choose):
    - lowercase <firstname>
    - uppercase <FIRSTNAME>
    - underscore <first_name>
    - pascal case <FirstName>
    - camel case <firstName>

Best practice: create simple, short, semantic and auto-descriptive names for your tags, avoid punctuation chars, and avoid non-english chars.

XML Attributes

The XML attribute is a part of an XML element. The addition of attributes in the XML element gives more precise properties of the element i.e, it enhances the properties of the XML element.

Syntax:

```
<element_name attribute1 attribute2 ... > Contents... </element_name>
```

Example:

```
<text category = "message">Hello Geeks</text>
```

Attributes store pieces of information about the element in which they are set and not about other elements

Attribute's values must be single or double quotes. If a value has a quote on the name itself, use the other quote type for the attribute delimitation or entity references:

```
<movie name='A "movie" name'></movie> or
<movie name="A &quot;movie&quot; name"></movie>
```

Disadvantages of attributes:

- Can't contain multiple values;
- Can't contain tree structures;
- Not easily expandable for future change.

When to use attributes:

- Use attributes for metadata only, that is, info that does not refer to the data semantics or business logic, for example, and id of a person, but not data about the person itself (name, birthDate, nationality, etc)
- When the metadata, represented by the id, is simple and will not need to have a complex structure or multiple values

If an attribute is not to be used, simply use an additional nested element with the data. Either at an attribute or at an inner element, the data will be easily generated and consumed by the application that is supposed to make use of it.

XML Namespaces

As defined by the W3C Namespaces in XML Recommendation, an XML namespace is a collection of XML elements and attributes identified by an Internationalized Resource Identifier (IRI).

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

Solving the Name Conflict Using a Prefix:

Name conflicts in XML can easily be avoided using a name prefix. This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

- XML namespaces are defined by the attribute xmlns:prefix followed by the URI between double quotes (e.g.: xmlns:danielpm1982=”danielpm1982.com/xml”).

```
<?xml version="1.0" encoding="UTF-8"?>

<danielpm1982:animalGroup
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.danielpm1982.com/AnimalGroup AnimalGroup.xsd"
      xmlns:danielpm1982="http://www.danielpm1982.com/AnimalGroup"
      danielpm1982:groupId="1">
      <danielpm1982:dog danielpm1982:animalId="1">
```

```
            <danielpm1982:name>DogA</danielpm1982:name>
            <danielpm1982:species>Canis familiaris</danielpm1982:species>
            <danielpm1982:color>Brown</danielpm1982:color>
            <danielpm1982:birthDate>2005-01-01</danielpm1982:birthDate>
            <danielpm1982:owner>OwnerDEF</danielpm1982:owner>
      </danielpm1982:dog>
      <danielpm1982:cat danielpm1982:animalId="2">
            <danielpm1982:name>CatA</danielpm1982:name>
            <danielpm1982:species>Felis catus</danielpm1982:species>
            <danielpm1982:color>white</danielpm1982:color>
            <danielpm1982:birthDate>2000-01-01</danielpm1982:birthDate>
            <danielpm1982:owner>OwnerABC</danielpm1982:owner>
      </danielpm1982:cat>
</danielpm1982:animalGroup>
```

- If you wanna use the elements of a certain namespace, you just proceed the element's declaration with the 'prefix:' (e.g.: <danielpm1982:name>)
- If the attribute xmlns has no prefix associated to it, then it's considered the default namespace for all elements inside that element, and no prefix is required when using its elements inside this scope (e.g.: <name>)
- The attribute can be declared and defined at one or more inner elements or at the root element of the XML document, for all elements of the XML instance.