Testing XML technologies

Software testing & Quality
Assurance

What is xml ?

- **Extensible Markup Language (XML)**
- you may have heard many reasons why your organization should use it. But **what is XML**, exactly?

Xml example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PersonList Type="Employee">
<Title Value="Employee List"></Title>
<Contents>
<Employee>
<Name>John Barrimore</Name>
<No>18316</No>
<Deptno>d1</Deptno>
<Address>
<City>Seattle</City>
<Street>Abbey Rd</Street>
</Address>
</Employee>
</Contents>
</PersonList>
```

Markup Languages

- Markup is the process of using codes called tags (or sometimes tokens) to define the structure, the visual appearance, and — in the case of XML — the meaning of any data

Extensible

- The ability to create **tags** that define almost any data structure is what makes XML "extensible."

Benefits of XML

XML	HTML
the tags define the structure and meaning of your data	the tags define the look and feel of your data, the headlines go here, the paragraph starts there
XML allows you to create any tag that you need to describe your data	HTML is limited to a predefined set of tags that all users share.
designed to transport and store data, with focus on what data is	HTML was designed to display data, with focus on how data looks

Xml example

```xml
<?xml version="1.0"?>
<person>
    <name>
        <firstname>Alaa</firstname>
        <lastname>Mohamed</lastname>
    </name>
    <job>Singer</job>
    <gender>female</gender>
</person>
```

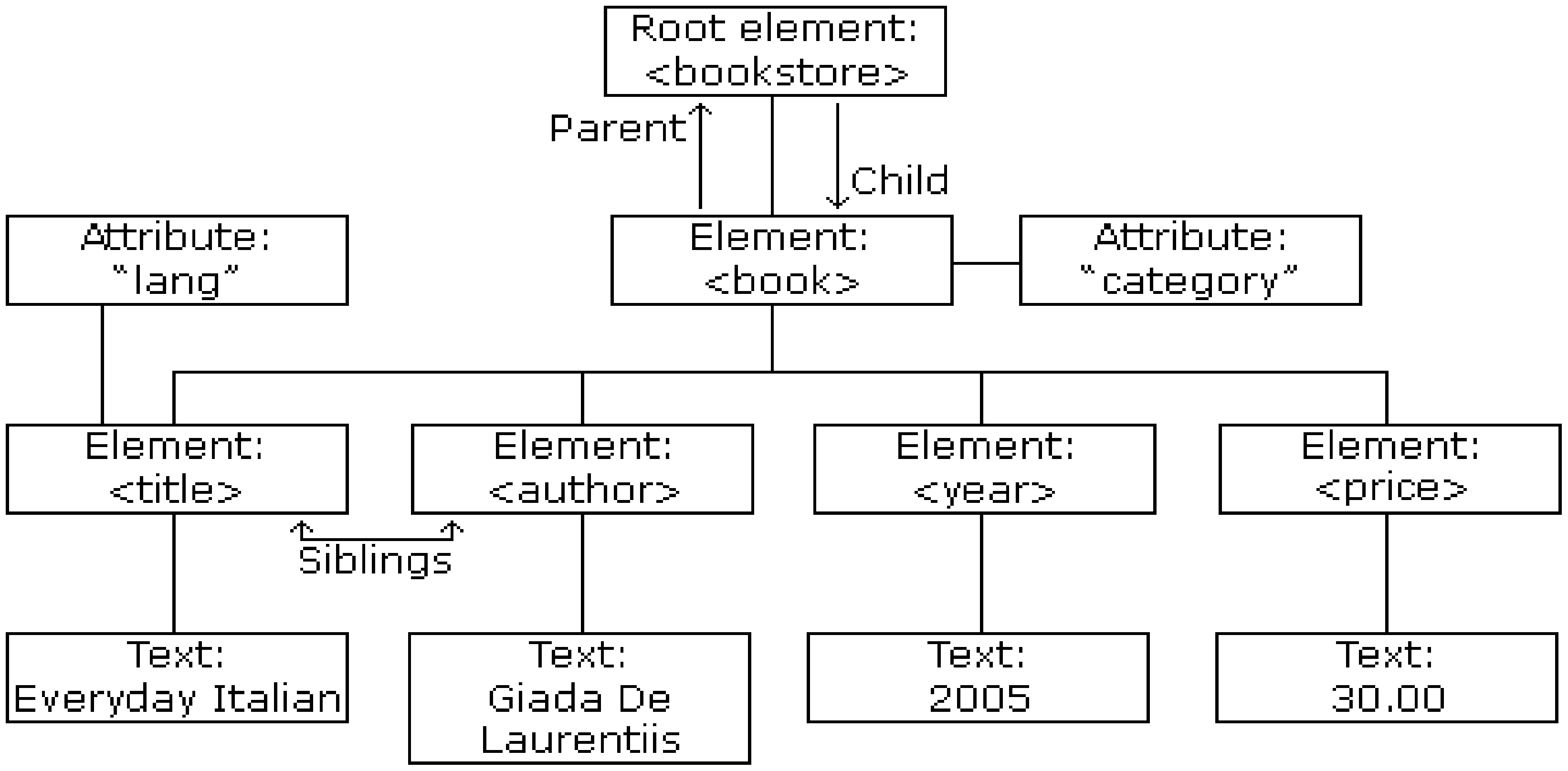
Why xml ?

- XML Holds Data, Nothing More
- XML Separates Structure from Formatting
- XML Simplifies Data Sharing
- XML Simplifies Data Transport
- XML Simplifies Platform Changes
- XML is Used to Create New Internet Languages

XML SYNTAX

Rules For Creating Wellformed XML…

Xml Node tree



Elements

```
<?xml version="1.0"?>
<person>
    <name>
        <firstname>Paul</firstname>
        <lastname>McCartney</lastname>
    </name>
    <job>Singer</job>
    <gender>Male</gender>
</person
```

Element Name Requirements

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character
- Names cannot start with the letters xml (or XML, or Xml, etc)
- Names cannot contain spaces
- Any name can be used, no words are reserved.

XML Element Attributes

- XML elements can have attributes, just like HTML.
- **Attributes provide additional information about an element.**
- <file type="gif">computer.gif</file>
- XML Attributes Must be Quoted
- <gangster name='George "Shotgun" Ziegler'>

Attributes

```
<name title="Sir">
    <firstname>Paul</firstname>
    <lastname>McCartney</lastname>
</name>
```

XML Attributes

- **Some of the problems with using attributes are:**
- Attributes cannot contain **multiple values** (elements can)
- Attributes cannot contain **tree structures** (elements can)
- Attributes are difficult to read and maintain.

XML Attributes

- XML Attributes for **Metadata**
- Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML.

XML SYNTAX RULES

Rules For Creating Wellformed XML…

Rule 1,2 :

- XML Documents Must Have a Root Element
- XML Tags are Case Sensitive

Bad Examples…

```
<h1>Sample Heading</H1>
```

XML

```
<H1>Sample Heading</h1>
```

XML

These examples are all valid under HTML but under XML they are invalid since they are in the incorrect case.

Good Examples…

```
<h1>Sample Heading
<p> text </p>
</h1>
```

XML

Rule 3

- XML Documents Must Have a Root Element
- XML Tags are Case Sensitive
- Every open tag must be closed.

Bad Examples…

```
<list>
  <listitem>tomatoes
  <listitem>lettuce
  <listitem>green onion
</list>
```

…and
</listitem>
tags.

XML

Good Examples…

```
<list>
  <listitem>tomatoes</listitem>
  <listitem>lettuce</listitem>
  <listitem>green onion</listitem>
</list>
```

XML

Rule 4

- XML Documents Must Have a Root Element
- XML Tags are Case Sensitive
- Every open tag must be closed.
- XML Elements Must be Properly Nested

Bad Examples…

```
<list>
  <listitem>tomatoes</list>
</listitem>
```

These examples are invalid since they are both examples of overlapping elements or improper nesting.

Good Examples…

```
<a>A good example of <b>nesting</b>
elements.</a>
```

XTML

```
<list>
  <listitem>tomatoes</listitem>
</list>
```

Rule 5

- XML Documents Must Have a Root Element
- XML Tags are Case Sensitive
- Every open tag must be closed.
- XML Elements Must be Properly Nested
- Attribute values must be enclosed in single or double quotes.

Bad Examples…

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

XML

This example is invalid since Elements with attributes that have values not enclosed in quotes.

Good Examples…

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

Rule 6

- XML Documents Must Have a Root Element
- XML Tags are Case Sensitive
- Every open tag must be closed.
- XML Elements Must be Properly Nested
- Attribute values must be enclosed in single or double quotes.
- If an element is empty, it still must be closed.

Bad Examples…

```
<img src="icon.png">
```

XML

```
<graphic filename="icon.png">
```

These examples are invalid since they are both examples of empty tags missing the slash (/) at the end of the tag to conclude the elements.

Good Examples…

```
<img src=”icon.png”/>
```

XML

```
<graphic filename=“icon.png”></graphic>
```

Special Characters

- There are five special characters that cannot be included in XML documents.

Isolated Markup Characters

<	&lt;
>	&gt;
'	&apos;
"	&quot;
&	&amp;

Bad Examples…

```
<h1>Jack & Jill</h1>
```

```
<equation>5 < 2</equation>
```

These examples are invalid since they are both examples including isolated markup characters that are not included as a character entity.

Good Examples…

```
<h1>Jack &amp; Jill</h1>
```

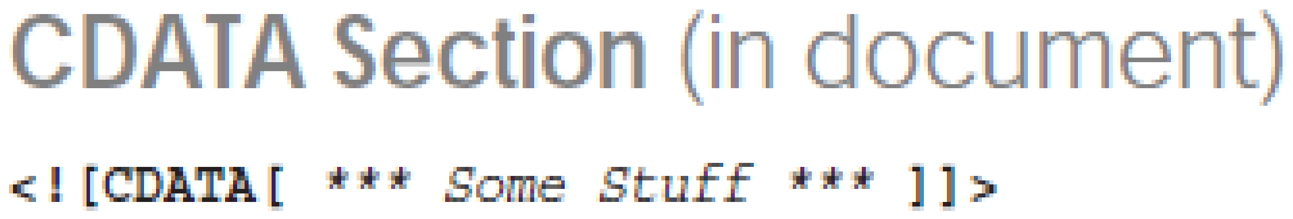
```
<equation>5 &lt; 2</equation>
```

Parsing

- PCDATA - Parsed Character Data
- CDATA - (Unparsed) Character Data

CDATA sections

- All text in an XML document will be parsed by the parser.
- But text inside a CDATA section will be ignored by the parser.



CDATA Section (in document)

```
<![CDATA[ *** Some Stuff *** ]]>
```

Summary

- There must be one and only one document element.
- Every open tag must be closed.
- If an element is empty, it still must be closed.
  - Poorly-formed: <tag>
  - Well-formed: <tag></tag>
  - Also well-formed: <tag />

Summary

- Elements must be properly nested.
  - Poorly-formed: <a><b></a></b>
  - Well-formed: <a><b></b></a>
- Tag and attribute names are case sensitive.
- Attribute values must be enclosed in single or double quotes.

Well Formed XML

- **Well Formed** - The logical structure is not validated against the DTD. A well formed document follows a set of rules to qualify as "well formed".

Xml Syntax

An optional prolog.

A document element, usually containing nested elements.

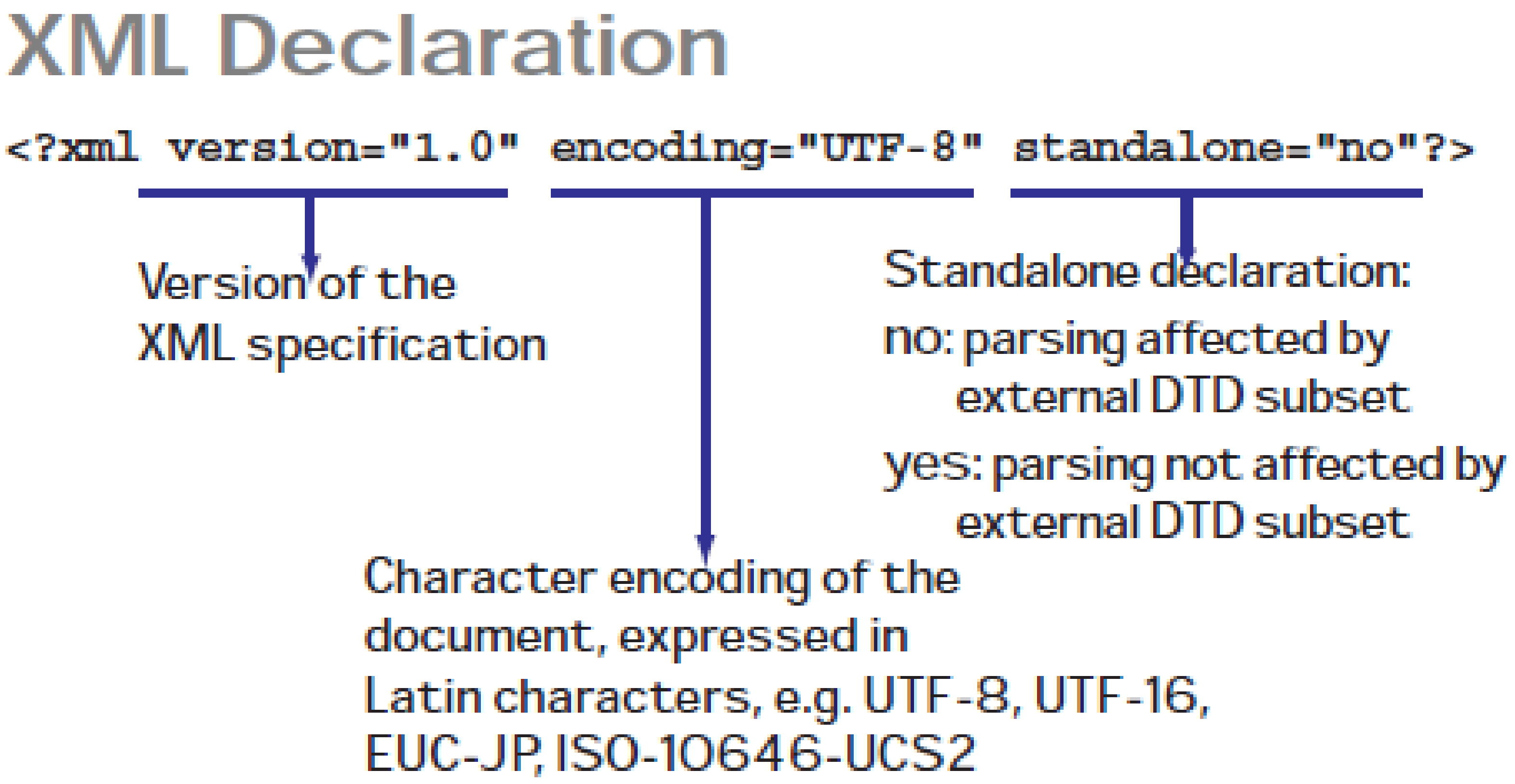
An optional prolog.

Xml Declaration

Processing Instructions

A Document Type Declaration

Xml Declaration



XML Declaration

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Version of the XML specification

Character encoding of the document, expressed in Latin characters, e.g. UTF-8, UTF-16, EUC-JP, ISO-10646-UCS2

Standalone declaration:

no: parsing affected by external DTD subset

yes: parsing not affected by external DTD subset

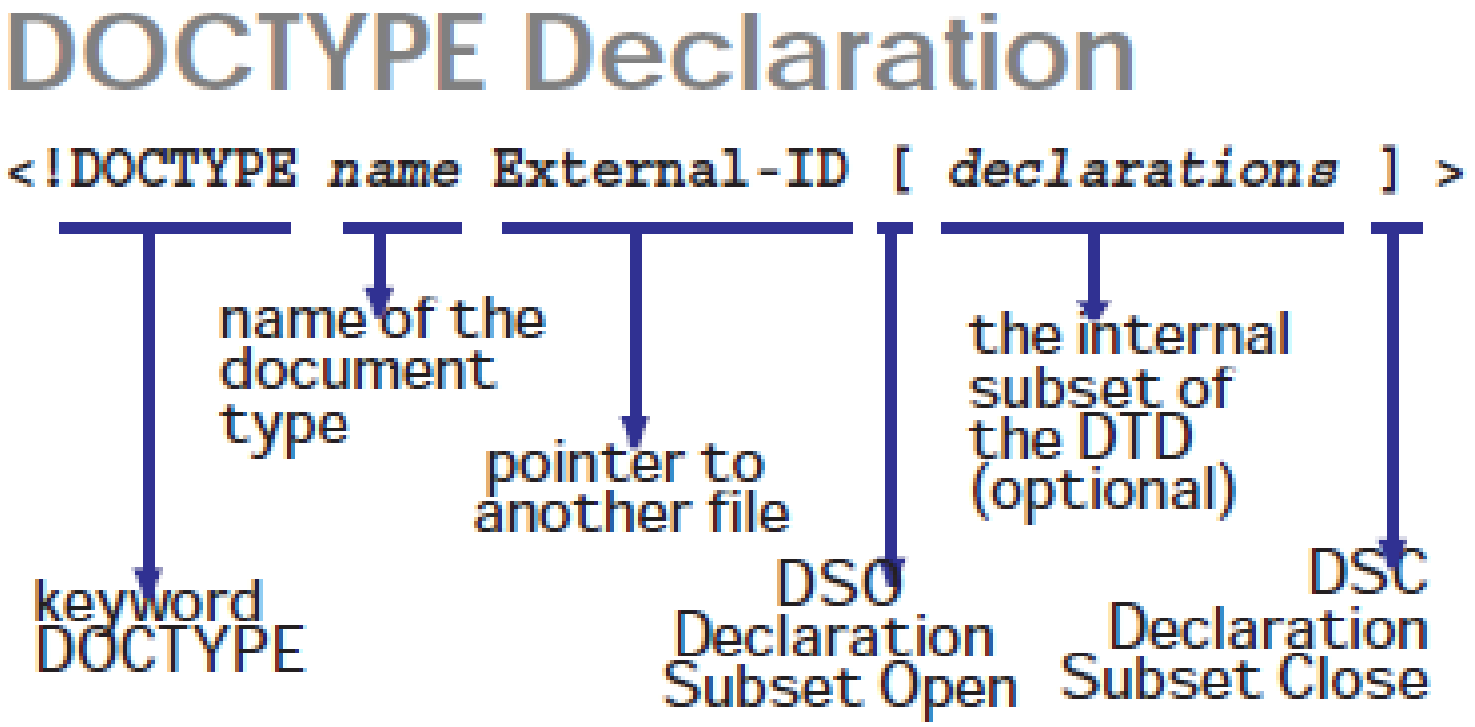
Processing Instructions

- <?xml-stylesheet type="text/css“ href="file.css"?>

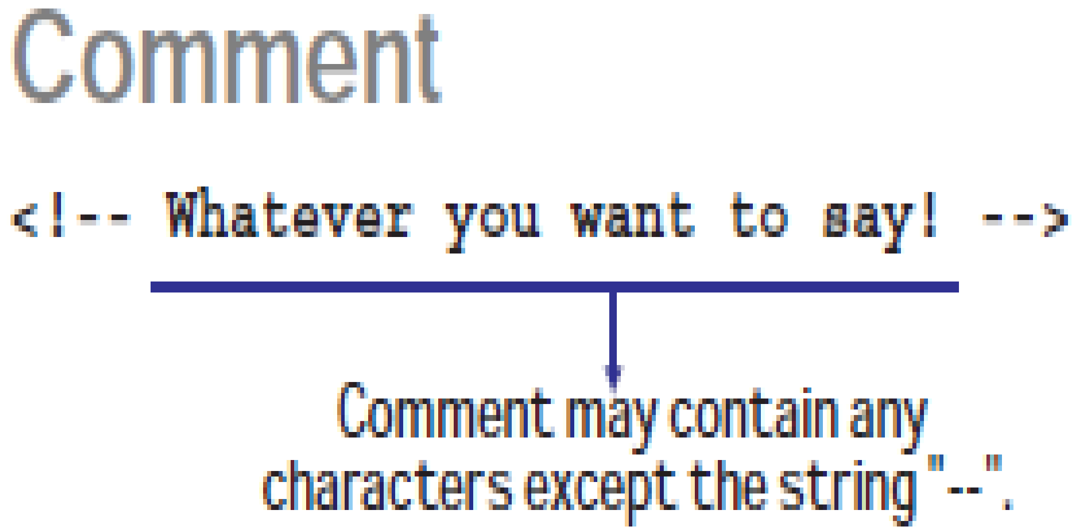
A Document Type Declaration (DTD)

- The purpose of a Document Type Definition or DTD is to define the **structure of a document** encoded in XML (extended Markup Language).
  - It’s file that **constrains** or restricts certain elements and attributes to exist in XML document.

DTD


DOCTYPE Declaration
<!DOCTYPE name External-ID [declarations] >
keyword DOCTYPE
name of the document type
pointer to another file
DSO Declaration Subset Open
the internal subset of the DTD (optional)
DSC Declaration Subset Close

Comments


Comment

```
<!-- Whatever you want to say! -->
```

Xml Syntax

An optional prolog.

A document element, usually containing nested elements.

Thank you !