XML

```
Course Code: ELEE1146

Course Name: Mobile Applications for Engineers

Credits: 15

Module Leader: Seb Blair BEng(H) PGCAP MIET MIEEE MIHEEM FHEA
```

Download as a PDF

Markup Languages (ML)

- Originally used in text formatting and typesetting
- Markup – tagging e-documents
- For two purposes:
  - Modify the look and formatting of the text
    - e.g `<B>`, `<I>`, `<BR>`, `<FONT>` in HTML
  - Establish the structure and meaning of a document for output
    - e.g. `<TITLE>`, `<HEAD>`, `<BODY>`, `<UL>` in HTML
- used in many systems: $\TeX$, RTF word format

XML as an ML

- XML: e**X**tensible **M**arkup **L**anguage
- Defined by the WWW Consortium (W3C) in 1998
- A universal method for representing information
- Fundamental shift in the way the information is distributed over the Internet
- The goal was to replace HTML as the language for publishing documents on the Web
- Both a document markup language and a database language

XML as an ML

- Derived from **SGML** (**S**tandard **G**eneralized **M**arkup **L**anguage)
    - established in 1986 as a document exchange format
- Provides structural and semantic information to data
- *Extensible*, unlike HTML
    - Users can add new tags, and separately specify how the tag should be handled for display
- *Metalanguage* – can be used to create other languages

Motivation for XML

- Used to be that C structs were used to access stored data, and that data was always stored as binary and then is interpreted.

```
struct dataStore
(
  char author[] = {'A','.','Name'};
  int age = 50;
  char ISBN[10] = {'I',...} ;
)
```

Motivation for XML

- HTML is not flexible, SGML is too complex
- HTML - insufficient for electronic data interchange
- Need for more powerful linking and style sheets
    - XML Linking Language (XLink)
    - XML Pointer Language (XPointer)
- Each application area has its own set of standards for representing information
- XML allows *application-specific* document types

XML vs HTML

- XML
    - Markup for *describing* data of virtually any type
    - Uses new, *user-defined* tags
    - Focus on how information is *structured*
- HTML
    - Markup for *displaying* data on the Web
    - Uses *fixed* number of tags
    - Focus on how information is *displayed*

XML Features

- Allows you to separate data from their representation (visualisation)
- XML document represents a blueprint of an all class of data
  Seb
  Blair
- XML helps exchanging data bases over the Internet

XML Example

```xml
<book>
  <author>
    <fname>William</fname>
    <lname>Pardi</lname>
  </author>

  <title>XML in Action</title>
  <publisher>Microsoft Press </publisher>
  <year>1999</year>
  <ISBN>0735605629</ISBN>
</book>
```

Structure of XML Data

- **Tag**: label for a section of data
  - Start (opening) Tag: `<tagname>`
  - End (closing) Tag: `</tagname>`
- **Element**: section of data beginning with `<tagname>` and ending with matching `</tagname>`
- Elements can have attributes
  - Attributes are specified by `name=value` pairs inside the starting tag of an element

Attributes

- Each element may have one or more attributes, but each attribute name can only occur once

```
<author fname="Rick",lname="Rolled"></author>
```

- Elements without sub-elements or text content can be abbreviated by ending the start tag with a /> and deleting the end tag

```
<author fname="Rick" lname="Rolled"/>
```

Attributes Vs. Sub-elements

- Same information can be represented in two ways

```
<author>
  <fname>William</fname>
  <lname>Pardi</lname>
</author>
```

OR

```
<author fname=“William” lname=“Pardi”></author>
```

- Attributes are part of markup, while sub-element contents are part of the basic document contents
- Suggestion: use **attributes** for *identifiers of elements*, and use **sub-elements** for *contents*

XML Names

- XML is case sensitive like Java and unlike HTML
- A name (for a tag or attribute) used in XML must:
  - begin with a *letter*, *underscore* or *colon*
  - followed by any number of additional *letters*, *digits*, *underscores*, *hyphens*, *periods* and *colons*
  - not begin with the letters *XML* (in any combination of upper- and lowercase) – they are reserved by W3C

Nesting

- Nesting is the process of embedding one element within another
    - Example:
        - html documents – opening and closing tags
- Nesting sets up a parent/child relationships
- Children are also called sub-elements
- Every child element **must** resides completely within its parent element

Nesting 2

- Elements must be properly nested
- Every start tag must have a unique matching end tag unlike HTML
- Exceptions from this rule:
    - Every document must have a single top-level element
    - Special tag – the empty-element tag `<tagname/>`
        - Example: `<amount></amount>` OR `<amount/>`

Nesting: Example

Proper nesting

```
<DOCUMENT>
  <PARENT>
    <CHILD></CHILD>
    <CHILD></CHILD>
  </PARENT>
</DOCUMENT>
```

Improper nesting

```
<DOCUMENT>
  <PARENT>
    <CHILD></CHILD>
    <CHILD></CHILD>
  </DOCUMENT>
</PARENT>
```

XML Document Structure

An XML document is made up of:

- declarations
- processing instructions
- comments
- elements

Declarations

- The first element in an XML document is called prolog
- The prolog is optional
- The prolog consists of two basic declarations that are optional too
- The XML Declaration
- The Document Type Declaration (DTD)

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

<!DOCTYPE customer_list
[
<!ELEMENT customer_list (customer)>
<!ELEMENT customer (name,location)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT location (#PCDATA)>
]>
```

Declarations 2

- An XML document should start with the XML declaration – gives the XML version being used
- `<?xml version="1.0"?>`
- All letters are lower case
- The document type declaration comes next – indicates the file with the grammar rules for a particular document, optional

Document Elements

- Each XML document starts with a prolog followed by the **root element**
- Each XML document contains one and only one root element
- Each element might have one or more **sub-elements** or **children**
- An element that contains sub-elements is called **container element** or **parent**

Processing Instructions, Comments, Entities

- **Processing instructions** are instructions which are passed on to applications processing the document.
- Processing instructions are enclosed in tags which begin with `<?` and end with `?>`
- **Comments** are enclosed between `<!-- and -->`, `<!--changed 10/1/01-->`
- **Entities** used to "escape" meaning of XML special symbols, only 5 entities are built-in:
    - `&amp;` (&),
    - `&lt;` (<),
    - `&gt;` (>),
    - `&quot;` ("),
    - `&apos;` ('),

XML Example

```xml
<?xml version =“1.0”?>
<!-- A Lecture formatted using XML -->

<lecture>
  <title>XML</title>
  <place>Nelson 128 C</place>
  <date>14 November 2023</date>
    <time>9:00-11:00am</time>
  <lecturer>
    <fname>Seb </fname>
    <lname>Blair</lname>
  </lecturer>
  <comment>
    An introductory lecture on XML, delivered to UG
    students at the School of Engineering, University of Greenwich
  </comment>
</lecture>
```

Namespaces

- XML data are exchanged between organizations
- Same tag name may have different meaning in different organizations, causing confusion on exchanged documents
- Namespace identify document type declarations so that they do not conflict with other element type declarations with the same name created by other programmers
- A solution is to use pairs
  - `namespace:local-name`
  - e.g. `unique-name:element-name`
- A unique name might be the company URL

Namespaces

- Namespaces help avoiding using long unique names all over a document
- The attribute xmlns defines an XML name space

```
<!--xmlns = xml namespace-->
<bank xmlns:MB=‘http://www.MyBank.com’>
…
  <MB:branch>
    <MB:branchname>MyTown</MB:branchname>
    <MB:branchcity> MyCity</MB:branchcity>
  </MB:branch>
…
</bank>
```

XML Applications

- XML 1.0 is currently approved as a recommendation by W3C
- Several applications of XML or enhancements to the language are under consideration
- For more information on XML visit http://www.w3.org/XML/

XML Applications: Scientific Data Interchange

- Many applications of XML have been developed as customized data languages for structuring data in various fields
- Mathematics: **MathML** for marking up mathematical formulas

Chemistry: **ChemML** – for marking up the molecular structure of chemicals

- Genetics: **BSML** (Bio Sequence Markup Language)

Whay about other structures.. JSON and YAML?

- YAML stands for “YAML Aint Markup Language“.
- JSON stands for “JavaScript Object Notation“.
- YAML uses indentation to define structured data. So each block in the YAML is differentiated by the number of white spaces.
- All three mentioned serialization language has same extension as their name. (.yaml for YAML, .json for JSON, .xml for XML). So it is easy to remember.
- In fact, file extensions are arbitrary for all the three data serialization standard. It is useful for the application and users to know what files format, type of the content and their data structure.

JSON

```json
{
"EmpRecord": {
 "Employee": [
     {
       "-id": "emp01",
       "name": "Alex",
       "job": "Developer",
       "skills": "python, C/C++, paskal"
     },
     {
       "-id": "emp02",
       "name": "Bob",
       "job": "Tester",
       "skills": "lips, forton, REST APIs"
     }
   ]
 }
}
```

XML

```xml
<?xml version="1.0"?>
<EmpRecord>
<Employee id="emp01">
<name>Alex</name>
<job>Developer</job>
<skills>python, C/C++, paskal</skills>
</Employee>

<Employee id="emp02">
<name>Bob</name>
<job>Tester</job>
<skills>lips, forton, REST APIs</skills>
</Employee>

</EmpRecord>
```

YAML

```yaml
---
EmpRecord:
  Employee:
  -
    "-id": emp01
    name: Alex
    job: Developer
    skills: python, C/C++, paskal
  -
    "-id": emp02
    name: Bob
    job: Tester
    skills: lips, forton, REST APIs
```

XML Adv and Disadv

Advantages of XML

- XML is a generalized language that easily allows different formats to be realized from a common syntax.
- Schemas exist for validation and creation of custom types, while namespaces avoid collisions between elements.
- XPath and XQuery facilitate complex queries into XML documents.

Disadvantages of XML

- The XML language is verbose and often contains redundant syntax.
- Higher verbosity increases storage capacity and bandwidth needs.
- Not considered easily human-readable due to the descriptive nature of elements.

JSON Adv and Disadv

Advantages of JSON

- In contrast to XML, far more human-readable due to a more compact syntax.
- Simpler syntax with limited markup.
- Quick for systems and languages to parse.
- JSONPath, like XPath, is available for complex queries.

Disadvantages of JSON

- Limited data type support containing only strings, numbers, JSON object, array, boolean, and null.
- No namespace, comment, or attribute support.
- Simple structures may not support complex configurations.

YAML Adv and Disadv

Advantages of YAML

- Exceptionally human-readable syntax.
- Compact syntax, which uses Python-style indentation to denote structure.
- Supports language-independent object types. Scalers such as `int`, `binary`, `str`, and `bool` or collections such as `map`, `set`, `pairs`, and `seq`.

Disadvantages of YAML

- Indentation format is prone to syntax and validation errors.
- Portability with certain types may not exist due to a lack of features across all languages.
- Debugging is difficult, due to the declarative nature of YAML.
- Breakpoints and similar functionality do not exist.