University of Glasgow

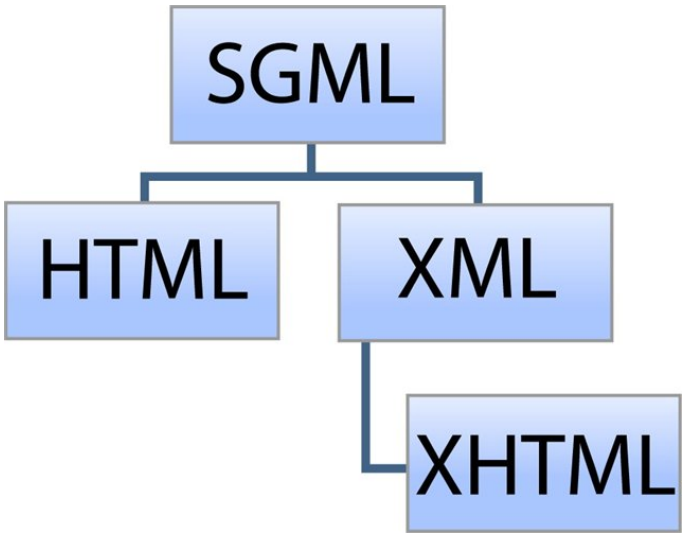
XML and JSON

Web Application Development 2

So what is XML, anyway?

- XML stands for "**eXtensible Markup Language**"
- XML is developed by the W3C; 1.0 in 1998
  - W3C is a consortium with hundreds of members including the major vendors and users of the web
    - AT&T, BBC, Citibank, Microsoft, Oracle Xerox…
    - and quite a few Universities
    - founded and led by Sir Tim Berners-Lee
- XML is designed to **transport** and **store** data
- Design goals of XML emphasise simplicity, generality, and usability

Mark up Languages



XML Design Goals

- **Why did the W3C design XML?**
  - Mark-up for the web was not being properly supported
    - Standard Generalized Mark-up Language (SGML) was too complex
    - While, HTML was too limited and mixed format with structure
- XML aimed to:
  - Provide a **simpler markup language** (easier than SGML)
  - **Separate format from structure** (Separate Concerns)
  - Be **extensible** and provide support for a host of applications
  - **Transport and store data**

The Role of XML

- To describe the structure of **semi-structured documents**
- A mechanism for **sharing, transporting and storing annotated data**
- To be a general purpose language for **data description** and **interchange**.
  - i.e., forms the basis of other languages
- XML has:
  - Emerged as a dominant standard
  - Developed a number of **vocabularies** for specific disciplines
  - Additional tools for additional layers of processing, such as:
    - the separate(!) ability to add **formatting** to XML documents
    - Querying XML documents, transforming XML documents, etc.

Extensions of XML

- XML can be extended to describe the data within specific domains, for example:
  - **XHTML –** web pages
  - **Wireless Markup Language** (WML) a specialisation of XML for Wireless Application Protocol – for early mobile data
  - **MathML –** The Language of Mathematics
  - **Chemical Markup Language** – "HTML with Molecules"
  - **SOAP** - for describing distributed method parameters
  - lots of other things can be built on top of XML

<?xml?>

WHAT DOES IT LOOK LIKE?

Sample XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
<food>
   <name>Belgian Waffles</name>
   <price>$5.95</price>
   <description>
  Two of our famous Belgian Waffles with plenty of real maple syrup
  </description>
   <calories>650</calories>
</food>
<food>
   <name>Strawberry Belgian Waffles</name>
   <price>$7.95</price>
   <description>
   Light Belgian waffles covered with strawberries and whipped cream
   </description>
   <calories>900</calories>
</food>
<food>
   <name>Berry-Berry Belgian Waffles</name>
   …
</food>
</breakfast_menu>
```

Example from https://www.w3schools.com/xml

Sample HTML file

```
<!doctype html>
<html class="no-js" lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>University of Glasgow - Schools - School of Computing Science</title>
    <meta http-equiv="X-UA-Compatible" content="IE=EDGE" />
    <link rel="canonical" href="https://www.gla.ac.uk/schools/computing/" />
    <!--standard meta tags-->
    <meta name="keywords" content="Computer Science, Russell Group, Glasgow, Scotland, Top UK education and
research, Computing Science " />
    <meta name="description" content="Computer Science, Computing Science, University of Glasgow" />
...

</head>
<body style="visibility: hidden;" onload="loaded()">
    <!--noindex-->
            <div role="navigation" id="skip-to-content">
              <ul>
                <li>
                    <a href="#content-container">Skip to main content</a>
                 </li>
                </ul>
          </div>
    <div class="row" id="container">
            <div id="leftNav" class="show-for-small-only">
                <div id="leftNavlist">
              <div id="navigation-global-ss">.</div>
                <div id="search" style="display:none;">
<form action="//www.gla.ac.uk/search" method="get" id="cse-search-box-mobile">
    <input type="text" id="ssKeywordsMobile" name="query"  title="search" placeholder="Search" /><br />
  <input type="hidden" name="collection" value="staff-and-website-meta" />
```

More specifically,…

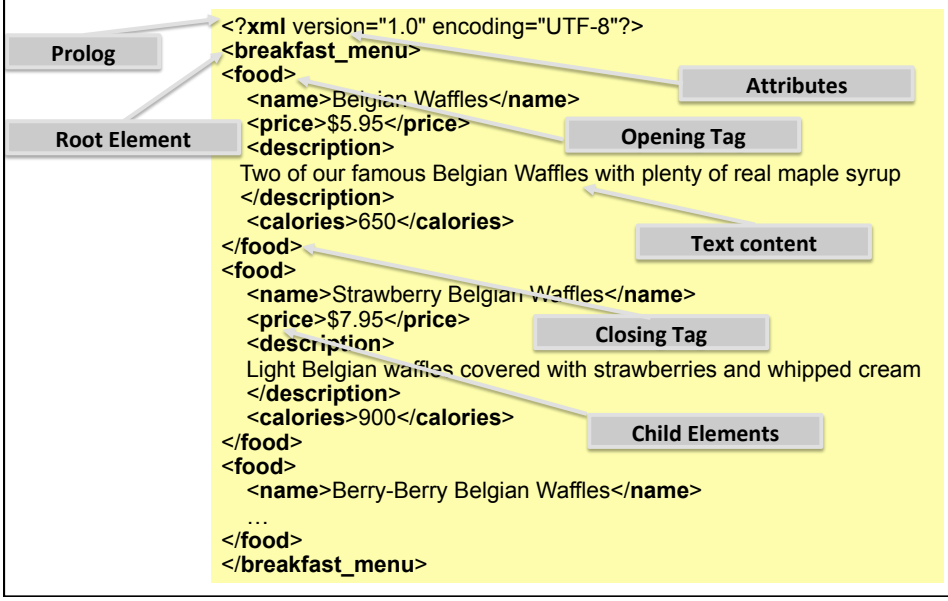
- HTML was designed to *display* data
  - HTML elements mix format and structure with content and presentation
  - While in XML, tags define structure and any presentation is handled separately
- The **structure of XML is tightly controlled**:
  - Tags are **case sensitive** and variable values must be **quoted**
  - If there is a **start tag**, there must be an **end tag**
  - A hierarchical structure of elements is enforced
  - These are not strictly enforced in the case in HTML
- XML provides flexibility
  - **New tags** (and variables) can be created, i.e., user-defined

XML Document Structure

An XML document consists of three parts:

- an optional **prolog** – **XML declaration**:
  <?xml version="1.0" encoding="UTF-8"?>
  - version – must be 1.0 or 1.1
  - encoding – how characters are encoded in the file
  - standalone – "yes" if this document is entirely self-contained, "no" if it has an external DTD or Schema. ("No" is default)
- the **body** – containing the document elements and data
- an optional **epilog** – containing **comments** and **processing instructions**
  <!-- This XML document is over -->

XML Document Anatomy



XML Elements

- Elements are the basic building blocks of XML
- An XML element is everything from (including) the element's start tag to (including) the element's end tag
- An element can contain:
  - text
  - attributes
  - other elements
  - or a mix of the above
- Element names are case sensitive
- Closed elements consist of both opening and closing tags:

```
<Url>http://www.gla.ac.uk/</Url>
```

- Elements can be nested
  - All elements must be nested within a single root element
  - Nested elements are child elements
- Empty elements are denoted by: `<Url></Url>` or just `<Url />`

XML Attributes and Values

- Attributes are characteristics of elements
- Attributes are case sensitive
- Attributes have values – they must be in quotes!
- All values are text strings

```
<ResultSet type="web" totalResultsAvailable="211000000"
totalResultsReturned="10" firstResultPosition="1" > … </ResultSet>
```

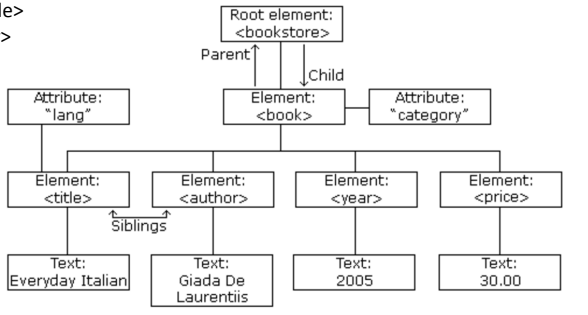
- Values can contain most characters and whitespace
  - Take care when using special characters esp. <,>,”, etc. Use escape values e.g. for **<** use **<**

Well Formed XML

- An XML document is *well-formed* if the markup satisfies:
  - **XML tags are Case Sensitive**
  - **Corresponding tags:** for every start tag there is an end tag
  - **Hierarchically structured:** An XML parser will be able process it and make use of the tree structure
    - e.g., **`<a><b>some text</a></b>`** *is not* well-formed
    - i.e., not properly nested
  - XML attribute values must be quoted
  - XML documents have to have a root element

XML Tree Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
 <book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
 </book>
 <book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
 <book category="web">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
 </book>
</bookstore>
```



Predefined and Valid XML

- To share XML **a pre-defined structure can be used:**
  - These describe the tags which can appear, and can be done using:
    1. **Document Type Definitions** (DTD), or
    2. **XML Schemas and XML Namespaces**
  - The XML can be checked according to the definitions and validated.
  - These structures are references either at the top of the file or provided separately.
- An XML document is **Valid** if it is Well-Formed and also conforms to the rules in the DTD or Schema
- Many XML validators available
  - E.g., https://www.xmlvalidation.com

Example DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Bob</to>
<from>Alice</from>
<heading>Reminder</heading>
<body>Don't forget to cook dinner</body>
</note>
```

The root of the document is the element "note"

The note element must contain the elements "to," "from", "heading", "body"

#PCDATA means "parseable character data"

Referencing an external DTD

Put following in note.dtd:

```
<?xml version="1.0"?>
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Then in note.xml:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Bob</to>
  <from>Alice</from>
  <heading>Reminder</heading>
  <body>Don't forget to cook dinner</body>
</note>
```

Example Schema

Part of note.xsd:

```
<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>
```

Defines the element called "note"

the "note" element is a complex type

the complex type is a sequence of elements

The element "body" is of type string

Referencing an external schema

```
<?xml version="1.0" encoding="UTF-8"?>
<note
xmlns="https://www.dcs.gla.ac.uk/thisNameSpace"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
        "https://www.dcs.gla.ac.uk/thisNameSpace   note.xsd">
  <to>Bob</to>
  <from>Alice</from>
  <heading>Reminder</heading>
  <body>Don't forget to cook dinner</body>
</note>
```

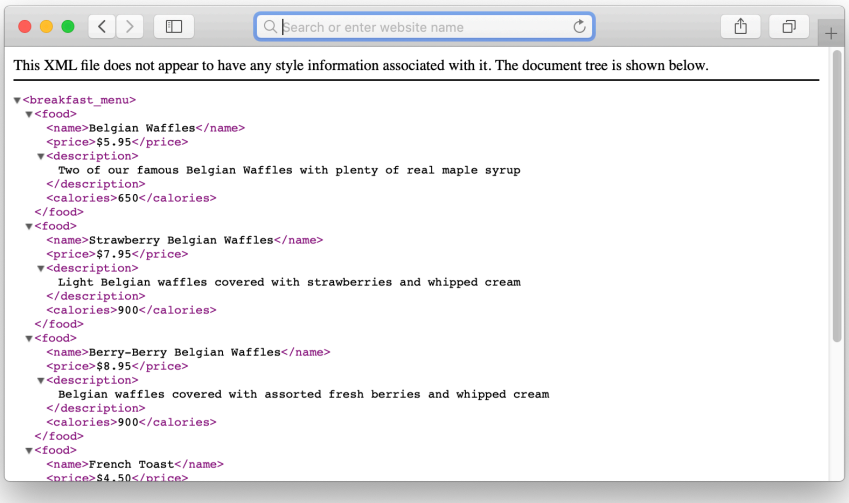
DTDs vs Schemas

- XML schemas are more powerful than DTDs:
  - XML schemas are written in XML
  - XML schemas are extensible to additions
  - XML schemas support data types
  - XML schemas support namespaces
- Why use an XML schema?
  - With XML schema, your XML files can carry a description of its own format
  - With XML schema, independent groups of people can agree on a standard for interchanging data
  - With XML schema, you can verify data

Formatting XML

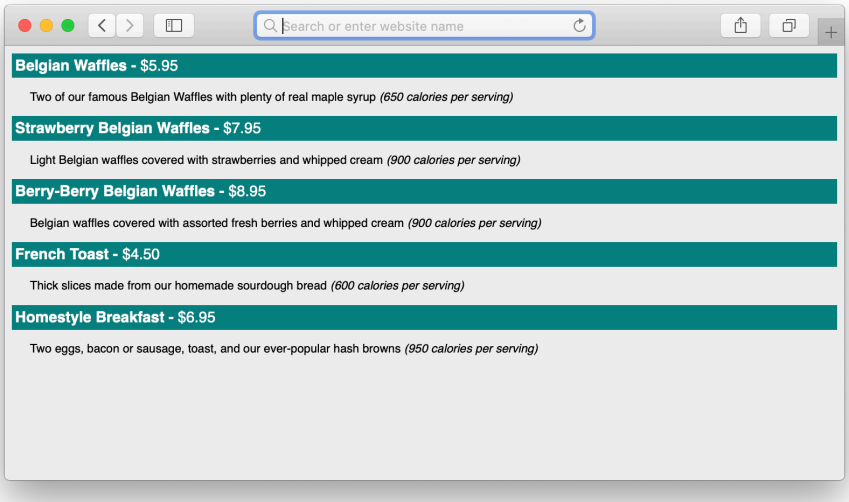
- XSL (EXtensible Stylesheet Language) is a styling language for XML
- XSLT stands for XSL Transformations
  - XSLT can be used to transform XML documents into other formats (e.g., XML -> XHTML)
- Process:
  - Start with a raw XML document
  - Create an XSL Style Sheet
  - Link the XSL Style Sheet to the XML Document
    - E.g., <?xml-stylesheet type="text/xsl" href=”menus.xsl"?>
- XSL is outside of the scope of this course

View – No XSL



View – With XSL

Top of xml: <?xml-stylesheet type="text/xsl" href="menu.xsl"?>



Strictly Speaking…

XHTML Strict – descends from XML, so rules to follow

- **Separates visual rendering** from the **content**
  - No style tags
- Strict set of rules enforced on markup
  - e.g. Hierarchy enforced strictly, tags all lower case, restricted placement of elements
- An XHTML Strict Document will work in many different environments:
  - visual browsers, braille readers, text based browsers, print
- It is highly configurable by the user
- And highly maintainable by the developer

How XML Differs from HTML

- XML was designed to **transport** and **store** data
- HTML was designed to **display** data
- **Carrying** information vs **displaying** information

Java Script Object Notation (JSON)

JSON Introduction

- Lightweight data interchange format
  - "Easy" for humans to read and write
  - Easy for machines to parse and generate
  - Less boilerplate, so more information per byte
- JSON is built on two universal data structures
  - A collection of name/value pairs
    - Often realized as a object, record, struct, dictionary, hash..
  - An ordered list of values
    - Often realized as an array, vector, list..
- JSON is language independent

Comparison of XML and JSON

- XML:

```xml
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

- JSON:

```json
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

JSON and JavaScript

- JSON uses JavaScript syntax, but the JSON format is text only, just like XML
- Text can be read and used as a data format by any programming language
- JSON evaluates to JavaScript Objects
  - The JSON format is syntactically identical to the code for creating JavaScript objects.
  - Instead of using a parser (like XML does), a JavaScript program can use standard functions to convert JSON data into native objects

JSON Syntax

- JSON syntax is derived from JavaScript object notation syntax:
  - Data is in name/value pairs in the form "name" : "value"
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays
- Example JSON object:

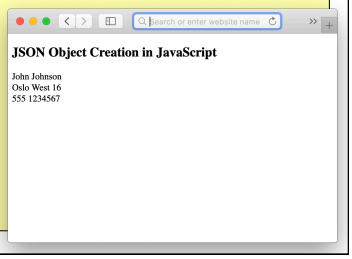
```
{"firstName":"John", "lastName":"Doe"}
```

- Example JSON array:

```
[ {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter","lastName":"Jones"} ]
```

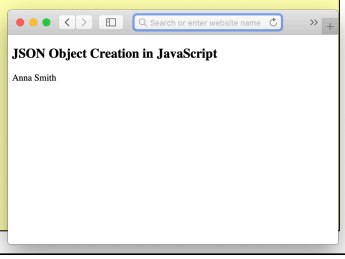
Display JSON

```
<!DOCTYPE html>
<html> <body>

<h2>JSON Object Creation in JavaScript</h2>

<p id="demo"></p>

<script>
var text = '{"name":"John Johnson","street":"Oslo West 16",
"phone":"555 1234567"}';

var obj = JSON.parse(text);

document.getElementById("demo").innerHTML =
  obj.name + "<br>" +
  obj.street + "<br>" +
  obj.phone;
</script>
</body> </html>
```



Display JSON (2)

```
<!DOCTYPE html>
<html> <body>

<h2>JSON Object Creation in JavaScript</h2>
<p id="demo"></p>

<script>
var text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';

var obj = JSON.parse(text);

document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " +
obj.employees[1].lastName;
</script>
</body> </html>
```



JSON in Python

json-demo.py:

```
import json

json_string = '{"employees":[{"firstName":"John","lastName":"Doe"},
{"firstName":"Anna","lastName":"Smith"},
{"firstName":"Peter","lastName":"Jones"}]}'

parsed_json = json.loads(json_string)

employees = parsed_json['employees']

for emp in employees:
   print(emp['firstName']+" "+emp['lastName'])
```

JSON versus XML

- JSON and XML are similar because:
  - both JSON and XML are "self describing" (human readable)
  - both JSON and XML are hierarchical (values within values)
  - both JSON and XML can be parsed and used by lots of programming languages
  - both JSON and XML can be fetched with an XMLHttpRequest (see AJAX lecture)

- JSON and XML are different because:
  - JSON doesn't use end tags
  - JSON is shorter
  - JSON is quicker to read and write
  - JSON can use arrays