Lecture 7 – Open Message Formats

This Week

- In previous weeks we studied about different component and RPC frameworks.
- This week we will look at XML and JSON which are used in network transfer of objects via serialization.
- Many of current RPC methods use these message formats widely in transferring data.

XML

What is XML?

not programing language. use for represent data

- eXtensible Markup Language
- Marked-up text is text that is structured by marking it with textual tags to describe the meaning of the text contents
- A markup language only defines the set of valid tags and their valid structure, not what to do with them

ASCII Data

- 10, Nimal, 56
- Data can be stored like above in ASCII format so it can be retrieved in any platform (including mainframes etc).
- But we don't know what 10, 56 mean.

XML Data

it machine readable and human readable also

- XML is self describing and also platform neutral

```
<Person>
  <Empno> 10 </Empno>
  <Name> Nimal </Name>
  <Age> 56 </Age>
</Person>
```

describe infomation

display infomation

XML vs HTML

Feature	XML (eXtensible Markup Language)	HTML (HyperText Markup Language)
Purpose	Designed for storing and describing structured data	Designed for displaying web content
Tag Rules	Tags must be properly closed and nested	Some tags can be left unclosed (e.g., `<p>`)
Case Sensitivity	Case-sensitive (`<Tag>` ≠ `<tag>`)	Not case-sensitive (`<TAG>` = `<tag>`)
Predefined Tags	No predefined tags; users define their own tags	Has predefined tags for structure and styling
Syntax Strictness	Strict syntax rules	More lenient syntax rules
Data Representation	Describes information (focus on meaning)	Displays information (focus on presentation)
Extensibility	Can define new markup languages (meta-language)	Cannot define new tags or structure
Use Case	Used for data storage, transfer, and configuration (e.g., Web Services, APIs)	Used for designing and structuring web pages
Formatting	Requires external styling (CSS, XSLT) for display	Supports built-in formatting and layout features

- HTML (HyperText Markup Language) defines the valid tags and structure for Web pages
  - Fairly inconsistent standard; eg: tags don't always need to be closed with `</tag>`, eg: `<p>` (paragraph)
- XML can be used to mark up text.
- A language used to define other languages (Meta Language)
  - XML itself has no tags
  - XML tags are not predefined. You must "invent" your own tags (new language)
- HTML is about ***displaying*** information, while XML is about ***describing*** information. (the main difference)

Why XML ?

- In its simplest form, XML is a markup language for documents that contain structured data.
- The key tenet of XML is that it can be used to describe any data in a human-readable, structured form.
- Structured information can contain both content, as well as information that defines the content.

different system can exchange information using XML

- XML is a cross-platform, software and hardware independent tool for transmitting information/data.

Where to Use XML

- XML is everywhere.
- XML is ideal for:
  - e-commerce (in particular B2B), content management, Web Services, distributed computing, peer-to-peer (P2P) networking and the Semantic Web…
  - However, it's huge in space
    - 3-20 times larger comparing to binary format

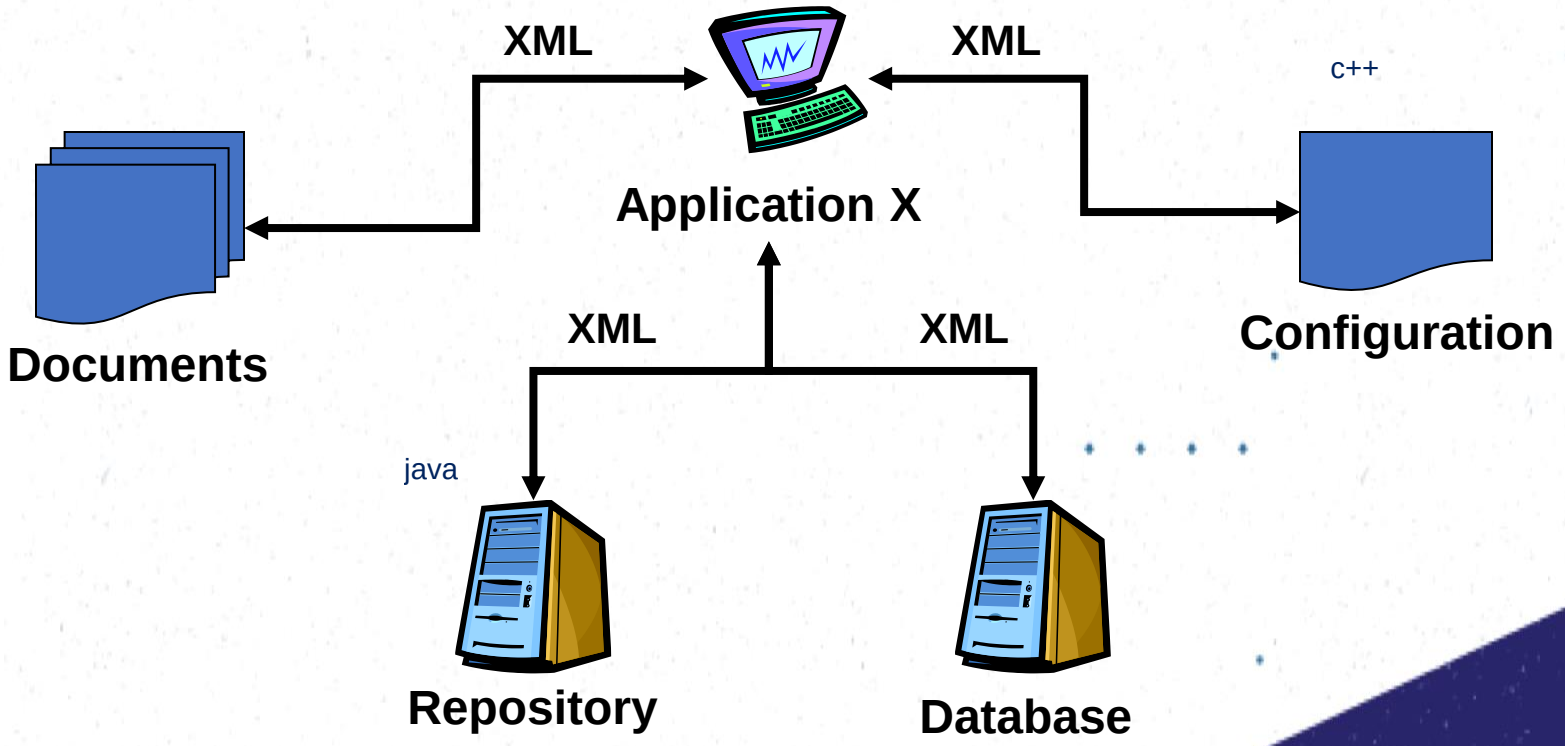
data exchange (e.g., Web Services, E-commerce, Configuration files, Distributed computing).

XML files are larger than binary files (3–20 times), their readability and flexibility make them essential.

- XML will be as important to the future of the Web as HTML has been to the foundation of the Web and that XML will be the most common tool for all data manipulation and data transmission.

Universal Language

- XML is a "use everywhere" data specification



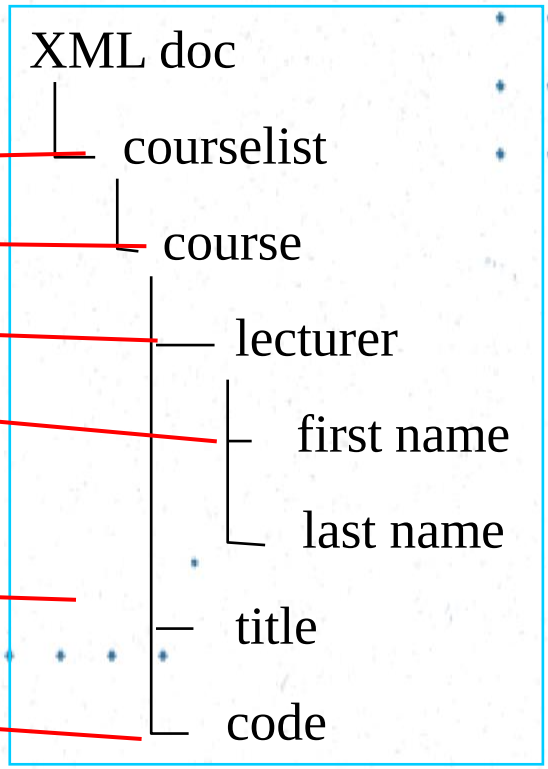
A Sample XML Document

declaration

communicate through the network data will encode binary format

tree structure

```
<?xml version = “1.0”  encoding="ISO-8859-1"?>
<!-- my first XML doc -->
<courselist xmlns="http://www.university.com" >
<course>
<lecturer>
   <firstname> H.T. </firstname>
   <lastname> Shen </lastname>
</lecturer>
<title>SOA</title>
< code>INFS3204</code>
</course>
</courselist>
```



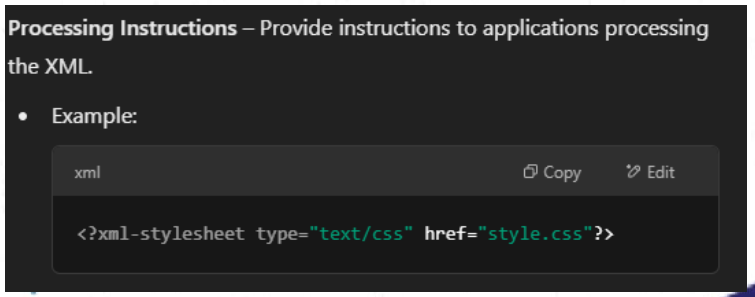
Benefits of XML

XML allows data to be exchanged seamlessly between different platforms, applications, and systems, making it ideal for web services and APIs.

- Open W3C standard
- Representation of data across heterogeneous environments
  - Cross platform
  - Allows for high degree of interoperability
- Strict rules
  - Syntax requires proper nesting, closing of tags, and following a well-defined structure
  - Structure data hierarchically, making it easy to read, parse, and process by both humans and machines.
  - Case sensitive distinguishes between uppercase and lowercase letters,

XML Syntax

- An XML document are composed of
  - An XML declaration: <?xml version = “1.0” encoding="ISO-8859-1"?>
  - Optional, but should be used to identify the XML version, namespace, and encoding schema to which the doc conforms
  - XML markup types
    1. **Elements (and attributes)**
    2. Entity references — Used to represent special characters. < <!-- Represents "<" -->
    3. Comments: `<!-- what ever you want to say -->`
    4. Processing instructions: `<?instruction options?>`
  - **Content**



Elements and Attributes

- Elements (or Tags): `<course> </ course>`
  - Primary building blocks of an XML document
  - All tags must be closed and nested properly (as in XHTML)
    - Empty tags ok (e.g., `< course />`) <image src="logo.png" />
- Attributes: `< course level = "undergraduate">`
  - Additional information about an element
  - Attribute value can be `required`, `optional` or `fixed`, and can have a default value
  - An attribute can often be replaced by nested elements
  - All values must be quoted (even for numbers, as in XHTML)

Elements and Attributes

- Elements and attributes can be named in almost any way you like
  - Should be descriptive and not confusing (human-readable!)
  - No length limit, case sensitive and ok to use the underscore (_)
  - No names are reserved for XML (namespaces can solve naming conflicts)
  - Must not start with a number or punctuation character or XML or Xml
  - Cannot contain spaces, the colon (:), space, greater-than (>) , or less-than (<)
  - Avoid using the hyphen (-) and period (.)

Element Relationship

Root Element: `<courselist>`

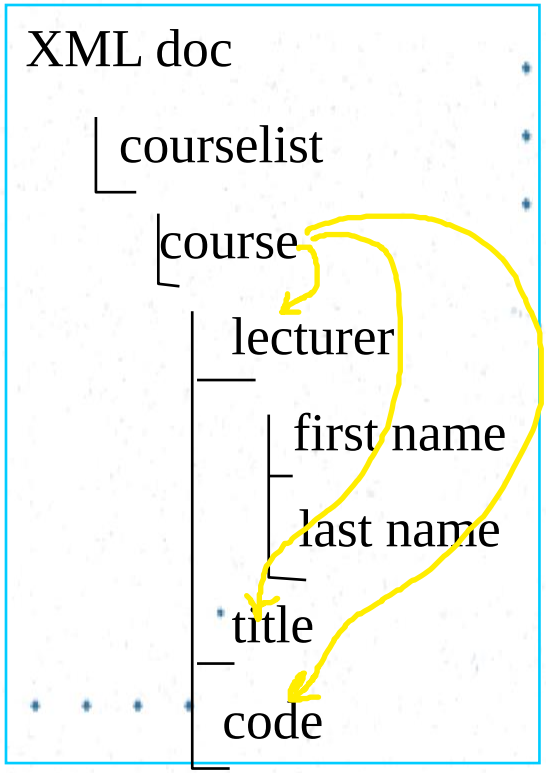
Child Elements of `<courselist>`: `<course>`

Child Elements of `<course>`: `<lecturer>`, `<title>`, `<code>`

Siblings (share the same parent `<course>`): `<lecturer>`, `<title>`, `<code>`

Nested Elements (inside `<lecturer>`): `<firstname>`, `<lastname>`

```
<?xml version = “1.0”  encoding="ISO-8859-1"?>
<!-- my first XML doc -->
<courselist xmlns="http://www.university.com" >
 <course>
  <lecturer>
   <firstname> H.T. </firstname>
   <lastname> Shen </lastname>
  </lecturer>
  <title>SOA</title>
  < code>INFS3204</code>
 </course>
</courselist>
```



courselist is the **root element**. *lecturer, title and code* are **child elements** of *course*. *lecturer, title and code* are **siblings** (or **sister elements**) because they have the same parent.

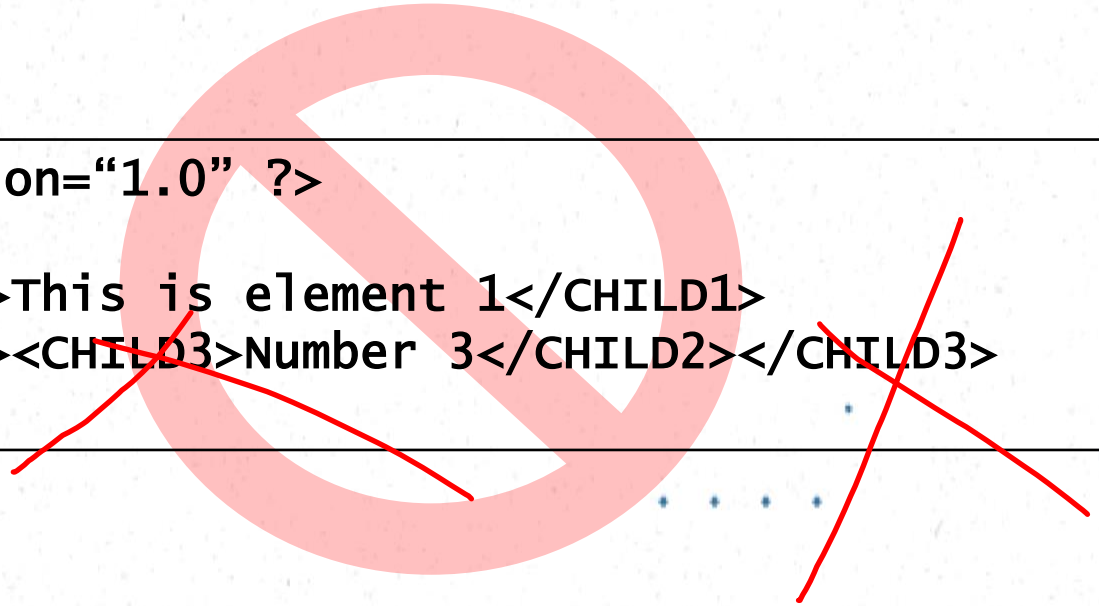
XML Content

- The text within the elements
  - So the document is structured!
- XML content can consist of any data
  - As long as the content does not confuse with valid XML metadata instructions (use entity references for special characters!)
- Every XML doc must have one and only one root element

```
<courselist>
 <course>
    <lecturer>
        <firstname>H.T</firstname>
            <lastname>Shen</lastna
            me>
    </lecturer>
    <title>SOA</title>
    < code>INFS3204</code>
  </course>
</courselist>
```

Well-formed XML 1?

```
<xml? Version="1.0" ?>
<PARENT>
    <CHILD1>This is element 1</CHILD1>
    <CHILD2><CHILD3>Number 3</CHILD2></CHILD3>
</PARENT>
```

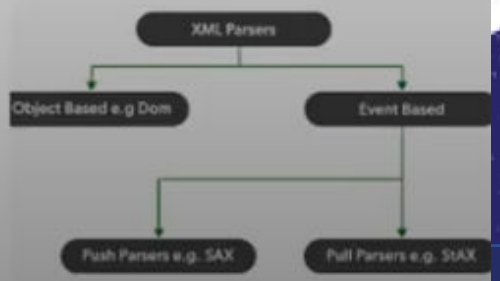


Well-formed XML 2?

```
<xml? version="1.0" ?>
<PARENT>
    <CHILD1>This is element 1</CHILD1>
    <CHILD2/>
    <CHILD3></CHILD3>
</PARENT>
```

large files suitable for SAX parser

DOM parser loads a full XML file in memory and creates a tree representation of XML document, while SAX is an event-based XML parser and doesn't load the whole XML document into memory. For small and medium-sized XML documents DOM is much faster than SAX because of in



<name> appears twice, once for Employee and once for Dependent.In large applications, different datasets might use the same tag names, causing ambiguity.

XML Namespaces

- XML tags are user defined
- Therefore it's possible that we can use the same element name to define two different things
- This will lead to a naming conflict.
- Also real world applications may have to use several XML based languages defined by various parties
- These several languages may have same element names

confuse

```
<Employee>
    <name>John Smith</name>
    <department>Admin</department>
    <salary>40000</salary>
    <dependents>
    <dependent>
        <name> Mark</name>
        <age>12</age>
    </dependent>
    </dependents>
</Employee>
```

Real-world applications use multiple XML-based languages, and some elements may overlap.

XML Namespaces

- Eg:
  - Think about a online furniture store selling furniture items from multiple vendors
  - Many vendors will have same <table> element
  - To resolve this kind of conflicts namespaces are used.

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

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

XML Namespaces

- Name conflicts in XML can easily be avoided using a name prefix.

```
<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>
```

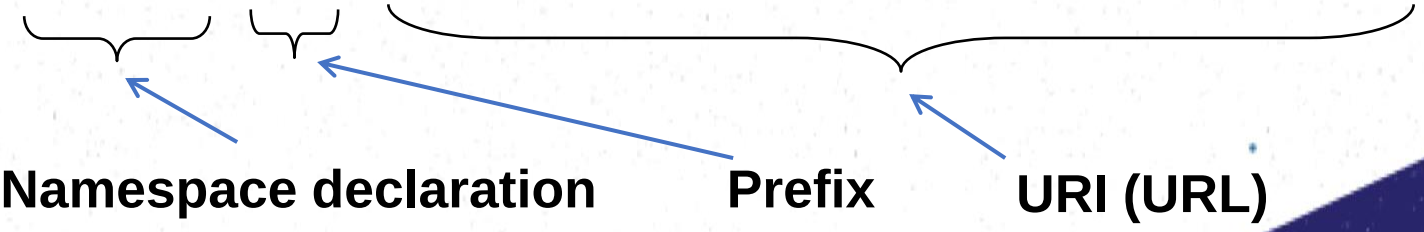
In the example above, there will be no conflict because the two<table> elements have different names.

In the exa vo
<table>
elements have different names.

XML Namespaces

- When using prefixes in XML, a **namespace** for the prefix must be defined
- The namespace is defined by the **xmlns attribute** in the start tag of an element.
- The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".
- A namespace is identified by a URI

```
xmlns: bk = “http://www.example.com/bookinfo/”
```



Namespaces: Declaration

Declaring XML Namespaces

Syntax

```xml
xmlns:prefix="URI"
```

- `xmlns` : Defines a namespace.
- `prefix` : The short identifier for the namespace.
- `URI` : The **Unique Resource Identifier**, usually a **URL** (doesn't have to be a real website).

Examples

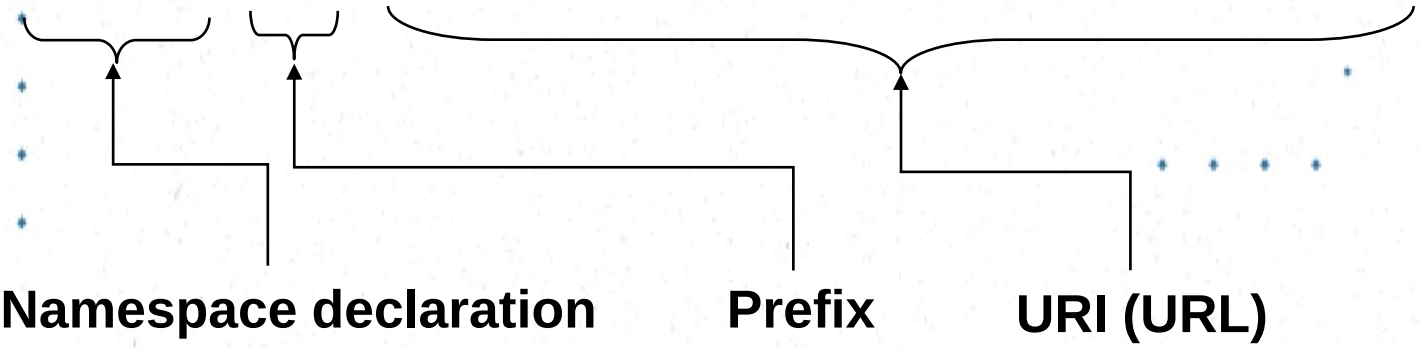
```xml
xmlns:bk="http://www.example.com/bookinfo/"
xmlns:bk="urn:mybookstuff.org:bookinfo"
```

- `bk` **is the prefix** for the **book namespace**.
- The **URI ensures uniqueness** across different XML documents.

```
xmlns: bk = "http://www.example.
```

```
xmlns: bk = "urn:mybookstuff.org
```

```
xmlns: bk = "http://www.example.com/bookinfo/"
```



Namespace declaration **Prefix** **URI (URL)**

Multiple XML Namespaces

I want to use my own furniture definitions (as the default),and two more name spaces (one is HTML and another is IKEA's furniture definitions)

```
<furniture xmlns = "http://www.itee.uq.edu.au/~zxf/furniture"
           xmlns:ikea = "http://www.ikea.com/names
           xmlns:html="http://www.w3.org/TR_REC
```

```
<furniture
    xmlns="http://www.itee.uq.edu.au/~zxf/furniture"
    xmlns:ikea="http://www.ikea.com/names"
    xmlns:html="http://www.w3.org/TR_REC-html40">

    <table> <!-- Uses the default furniture namespace -->
        <name>Classic Wooden Table</name>
        <width>100</width>
        <length>150</length>
    </table>

    <ikea:table> <!-- Uses IKEA's furniture definitions -->
        <ikea:name>Modern Glass Table</ikea:name>
        <ikea:width>120</ikea:width>
        <ikea:length>180</ikea:length>
    </ikea:table>

    <html:table> <!-- Uses HTML for layout -->
        <html:tr>
            <html:td>Product</html:td>
            <html:td>Price</html:td>
        </html:tr>
        <html:tr>
            <html:td>Table</html:td>
            <html:td>$200</html:td>
        </html:tr>
    </html:table>

</furniture>
```

- `<furniture>` element has 3 namespaces
  - The first one is the default one in this example
  - No naming conflicts anymore
    - <table>, <ikea:table>, <html:table>
  - All child elements of `<furniture>` inherit the 3 namespaces (un

Default and scope

- An XML namespace declared without a prefix becomes the default namespace for all sub-elements
- All elements without a prefix will belong to the default namespace
- Unqualified elements belong to the inner-most default namespace.
  - BOOK, TITLE, and AUTHOR belong to the default book namespace
  - PUBLISHER and NAME belong to the default publisher namespace

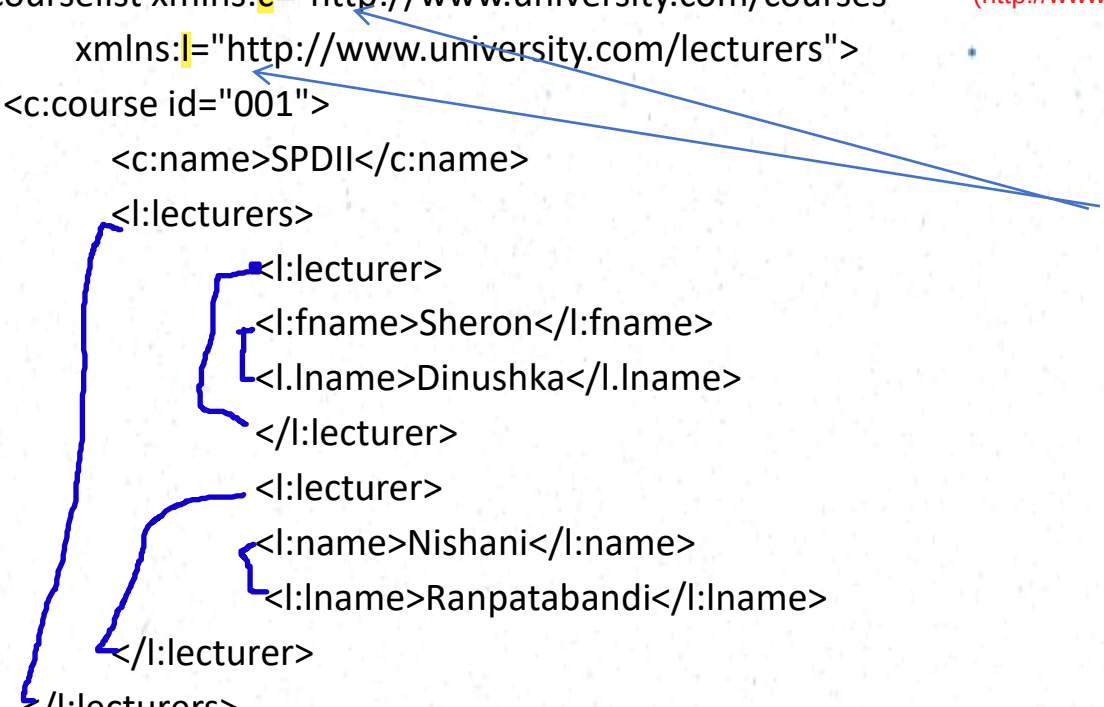
```
<BOOK xmlns="www.bookstuff.org/bookinfo">
   <TITLE>All About XML</TITLE>
   <AUTHOR>Joe Developer</AUTHOR>
   <PUBLISHER xmlns="urn:publishers:publinfo">
      <NAME>Microsoft Press</NAME>
   </PUBLISHER>
</BOOK>
```

Another sample XML document

c: prefix → Defines the course namespace (http://www.university.com/courses).

l: prefix → Defines the lecturers namespace (http://www.university.com/lecturers).

```
<?xml version="1.0" encoding="utf-8"?>
<courselist xmlns:c="http://www.university.com/courses"
        xmlns:l="http://www.university.com/lecturers">
  <c:course id="001">
        <c:name>SPDII</c:name>
        <l:lecturers>
              <l:lecturer>
              <l:fname>Sheron</l:fname>
              <l.lname>Dinushka</l.lname>
              </l:lecturer>
              <l:lecturer>
              <l:name>Nishani</l:name>
              <l:lname>Ranpatabandi</l:lname>
        </l:lecturer>
    </l:lecturers>
    </c:course>
</courselist>
```



another way…

Without the c: prefix, <course> and <name> now belong to the default namespace (http://www.university.com/courses).

The l: namespace is still used for lecturers.

```
<?xml version="1.0" encoding="utf-8"?>
<courselist xmlns="http://www.university.com/courses"
        xmlns:l="http://www.university.com/lecturers">
  <course id="001">
        <name>SPDII</name>
        <l:lecturers>
                <l:lecturer>
                <l:fname>Sheron</l:fname>
                <l.lname>Dinushka</l.lname>
                </l:lecturer>
                <l:lecturer>
                <l:name>Nishani</l:name>
                 <l:lname>Ranpatabandi</l:lname>
        </l:lecturer>
    </l:lecturers>
    </course>
</courselist>
```

Default namespace

XSD (XML Schema Definition

XML Schema (XSD)

defines the structure and rules for an XML document.

replaces DTD (Document Type Definition) for better validation.Helps XML parsers verify if an XML file follows correct structure.

- XML Schema Definition (XSD) language
  - Description of the structure of an XML document
    - XSD is itself an XML file following a fixed standard
    - Replaces the less-flexible DTD (Document Type Definition)

  - XSD used by parsers to check that an associated XML file is valid (as opposed to merely well-formed)
    - Well-formed: General XML syntax rules are followed
      - eg: Tags have end-tags, nesting is correct
    - Valid: Document follows XSD's semantics
      - eg: tags/attributes used are all defined in XSD, structure is OK

Feature	Well-Formed XML	Valid XML (XSD)
Syntax rules	✅ Yes	✅ Yes
Defined structure	❌ No	✅ Yes
Data type validation	❌ No	✅ Yes

What does XSD define?

- An XML Schema:
  - defines elements that can appear in a document
  - defines attributes that can appear in a document
  - defines which elements are child elements
  - defines the order of child elements
  - defines the number of child elements
  - defines whether an element is empty or can include text
  - defines data types for elements and attributes
  - defines default and fixed values for elements and attributes

some file format in xml

DTD
XSD

Lets look at an example (Note.xml)

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

<note
xmlns="http://www.w3schools.com/note"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">

  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XSD for Note.xml

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

<xs:schema xmlns:xs= “http://www.w3.org/2001/
targetNamespace=“http://www.w3schools.com/n
xmlns=“http://www.w3schools.com”
elementFormDefault=  "qualified">

<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>
</xs:schema>
```

XML File

xml — Copy — Edit

```
<bookstore>
    <book>
        <title>XML Essentials</title>
        <author>John Doe</author>
        <price>29.99</price>
        <isbn>123-4567890123</isbn>
    </book>
</bookstore>
```

Explain

XSD File (bookstore.xsd)

xml — Copy — Edit

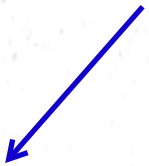
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="bookstore">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="book" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="title" type="xs:string"/>
                            <xs:element name="author" type="xs:string"/>
                            <xs:element name="price" type="xs:decimal"/>
                            <xs:element name="isbn" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

Explain

<schema> element

- The <schema> element is the root element of every XML Schema.
- The <schema> element may contain some attributes.



➢ **xmlns:xs=** **http://www.w3.org/2001/XMLSchema**

  - Indicates that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with **xs**:

<schema> element

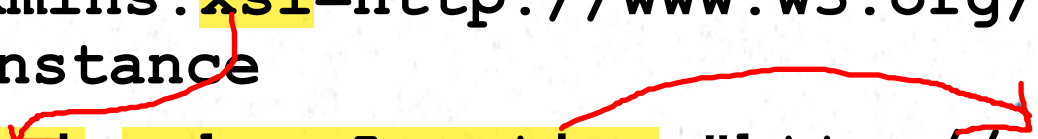
- **`targetNamespace="http://www.w3schools.com/note"`**
  - Indicates that the elements defined by this schema (note, to, from, heading, body.) belong to the target namespace.

- **`xmlns=http://www.w3schools.com`**
  - Indicates the default namespace

- **`elementFormDefault="qualified"`**
  - Indicates that elements used by the XML instance document which were declared in this schema must be namespace qualified.

Referencing a XSD in a XML doc

- `xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance`
- `xsi:schemaLocation="http://www.w3schools.com note.xsd">`



- XML documents can be linked with their schemas using the schemaLocation attribute.

- Because the schemaLocation attribute itself is in the http://www.w3.org/2001/XMLSchema-instance namespace, it is necessary to declare this namespace and map it to a prefix, usually xsi.

Complex Types in XSD

- **What is a Complex Element?**
  - A complex element is an XML element that contains other elements and/or attributes.
  - **Ex:**

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

In this case, <employee> is a complex element because it contains two child elements (<firstname> and <lastname>).

- The "employee" element can be declared directly by naming the element, like this:

```
<xs:element name="employee">
 <xs:complexType>
  <xs:sequence>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname"  type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

indicates that the employee element contains other elements.

specifies the order of the child elements.

child elements of the employee element.

Simple Elements in XSD

- The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

- Where xxx is the name of the element and yyy is the data type of the element.

- XML Schema has a lot of built-in data types. The most common types are:
  - `xs:string`
  - `xs:decimal`
  - `xs:integer`
  - `xs:boolean`
  - `xs:date`
  - `xs:time`

Simple Elements in XSD

- Ex

```
<lastname>Smith</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

- Corresponding simple element definitions:

```
<xs:element name="lastname"
 type="xs:string"/>
<xs:element name="age"
 type="xs:integer"/>
<xs:element name="dateborn"
 type="xs:date"/>
```

Attributes in XSD

- The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

  - Where xxx is the name of the attribute and yyy specifies the data type of the attribute.
  - Simple elements can't have attributes!

- Ex

```
<lastname lang="EN">Smith</lastname>
```

- corresponding attribute definition:

```
<xs:attribute name="lang" type="xs:string"/>
```

Stocks Example: XML for XSD

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="StockPortfolio.css"?>
<StockPortfolio xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
                xsi:schemaLocation="http://www.cs.curtin.edu.au/spd361
  StockPortfolio.xsd">
  <Stocks>
    <StockPurchase>
      <Ticker>GOOG</Ticker>
      <PurchasePrice>330.06</PurchasePrice>
      <NumPurchased>30</NumPurchased>
    </StockPurchase>

    <StockPurchase>
      <Ticker>MSFT</Ticker>
      <PurchasePrice>17.21</PurchasePrice>
      <NumPurchased>580</NumPurchased>
    </StockPurchase>
  </Stocks>

</StockPortfolio >
```

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="StockPortfolio.css"?>
<StockPortfolio xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
  xsi:schemaLocation="http://www.cs.curtin.edu.au/spd361 StockPortfolio.
  <Stocks>
    <StockPurchase>
      <Ticker>GOOG</Ticker>
      <PurchasePrice>330.06</PurchasePrice>
      <NumPurchased>30</NumPurchased>
    </StockPurchase>
    <StockPurchase>
      <Ticker>MSFT</Ticker>
      <PurchasePrice>17.21</PurchasePrice>
      <NumPurchased>580</NumPurchased>
    </StockPurchase>
  </Stocks>
</StockPortfolio>
```

Explain

Stocks Example: XSD

- <?xml version="1.0" encoding="UTF-8
- <xs:schema xmlns:xs="http://www.w3
- <xs:element name="StockPortfolio
- <xs:complexType>
- <xs:sequence>
- <xs:element nam
- </xs:sequence>
- </xs:complexType>
- </xs:element>
- <xs:element name="Stocks">
- <xs:complexType>
- <xs:sequence>
- <xs:element nam
maxOccurs="unb
- </xs:sequence>
- </xs:complexType>
- </xs:element>
-

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- Define StockPortfolio element -->
  <xs:element name="StockPortfolio">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Stocks"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Define Stocks element -->
  <xs:element name="Stocks">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="StockPurchase" minOccurs="0" maxOccurs="unbounde
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Define StockPurchase element -->
  <xs:element name="StockPurchase">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Ticker" type="xs:string"/>
        <xs:element name="PurchasePrice" type="xs:double"/>
        <xs:element name="NumPurchased" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Explain

```
    <xs:element name="StockPurchase">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Ticker"/>
                <xs:element name="PurchasePrice"/>
                <xs:element name="NumPurchased"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Ticker" type="xs:string"/>
    <xs:element name="NumPurchased" type="xs:int"/>
    <xs:element name="PurchasePrice" type="xs:double"/>
</xs:schema>
```

Exercise

- Write the XSD for following XML

```
<?xml version = “1.0”  encoding="IS
<courselist>
  <course>
    <lecturer>
      <firstname> H.T. </firstname>
      <lastname> Shen </lastname>
    </lecturer>
    <title>SOA</title>
    < code>INFS3204</code>
  </course>
</courselist>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Define the courselist element -->
  <xs:element name="courselist">
    <xs:complexType>
      <xs:sequence>
        <!-- Define the course element inside courselist -->
        <xs:element name="course">
          <xs:complexType>
            <xs:sequence>
              <!-- Define the lecturer element inside course -->
              <xs:element name="lecturer">
                <xs:complexType>
                  <xs:sequence>
                    <!-- Define firstname and lastname inside lecturer --
                    <xs:element name="firstname" type="xs:string"/>
                    <xs:element name="lastname" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <!-- Define title and code elements inside course -->
              <xs:element name="title" type="xs:string"/>
              <xs:element name="code" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Automating XSD Creation

- The XSD schema example was written by hand
  - Error prone
  - Time consuming
- Tools exist that will take classes and generate appropriate XSD schemas automatically
  - .NET: xsd.exe
  - Java: JAXB (Java Architecture for XML Binding) xjc.exe
  - There are others

Displaying XML

XSLT (Extensible Stylesheet Language Transformations) is used to transform XML data into a readable HTML format for display on a webpage.

- XSLT (extensible stylesheet language transformations) is used to format XML documents
- XML contains 'content'
- XSLT contains 'formatting info'

- XML + XSLT ⟶ HTML

XSLT Example (cdcatalog.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

XSL Stylesheet (cdcatalog.xsl)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

It includes formatting such as table borders and colors

specifies that the title and artist fields should be displayed in each table row.

Link the XSL stylesheet to XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
 <cd>
  <title>Empire Burlesque</title>
  <artist>Bob Dylan</artist>
  <country>USA</country>
  <company>Columbia</company>
  <price>10.90</price>
  <year>1985</year>
 </cd>
.
.
</catalog>
```

XSLT Example

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli

Resulting HTML (after applying XSLT):

After transforming the XML with the XSLT, the HTML output in a browser would look like:

html Copy Edit

Explain

```
<html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <tr>
        <td>Empire Burlesque</td>
        <td>Bob Dylan</td>
      </tr>
      <!-- More rows for other CDs will follow -->
    </table>
  </body>
</html>
```

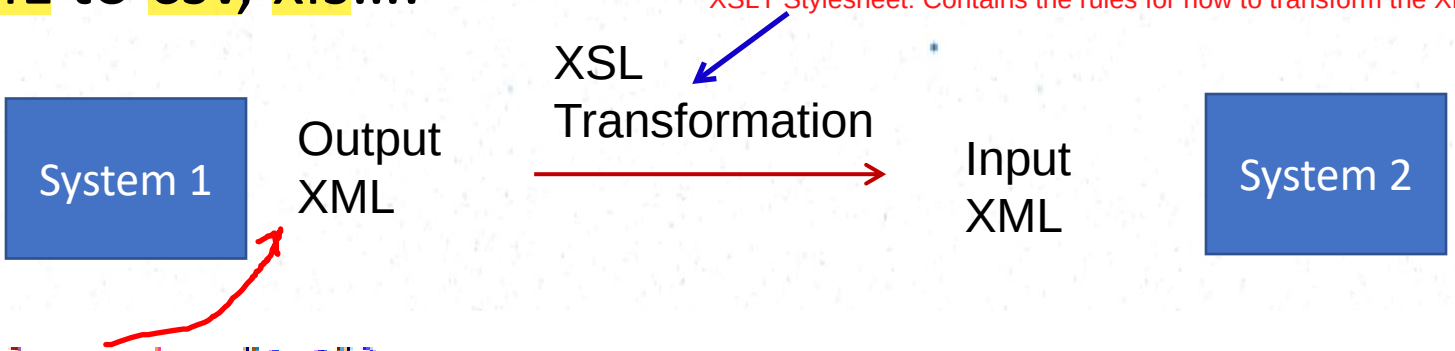
XSLT Online Editor (w3schools)

- https://www.w3schools.com/xml/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog

XSLT provides a powerful way to transform XML data to other formats like CSV, Excel, and even other XML formats, enabling easy integration between different systems that exchange data in XML.

XSLT Contd.

- XSL can be used to transform one XML to another
- XML to csv, xls....




```
<?xml version="1.0"?>
<investments>
  <item type="stock" exch="nyse"   symbol="ZCXM" company="zacx corp"
        price="28.875"/>
  <item type="stock" exch="nasdaq" symbol="ZFFX" company="zaffymat inc"
        price="92.250"/>
  <item type="stock" exch="nasdaq" symbol="ZYSZ" company="zysmergy inc"
        price="20.313"/>
</investments>
```

XSLT Contd.

```xml
<?xml version="1.0"?>
<portfolio>
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price>28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price>92.250</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price>20.313</price>
  </stock>
</portfolio>
```

XSLT Contd.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
  <portfolio>
    <xsl:for-each select="investments/item[@type='stock']">
      <stock>
        <xsl:attribute name="exchange">
            <xsl:value-of select="@exch"/>
        </xsl:attribute>
        <name><xsl:value-of select="@company"/></name>
        <symbol><xsl:value-of select="@symbol"/></symbol>
        <price><xsl:value-of select="@price"/></price>
      </stock>
    </xsl:for-each>
  </portfolio>
</xsl:template>

</xsl:stylesheet>
```

Data Formats

- A "binary" format maps data concepts to the native entities of the computer system, most of the time to bytes.
- A "text" format maps the data concepts to character strings, which must be translated to and from computer entities.
  - For example, to perform arithmetic, the string "-7.0" must be converted to an appropriate representation.
  - One advantage of text representations is that they can (to a degree) be read by a human.
    - XML has emerged as the most popular text format.

XML vs Binary

XML	Binary
'Human readable'	Requires software interpretation
Single, Universal Standard ▪ Web-friendly ▪ Massive commercial support	Many binary formats, availability varies
Stored as a stream of Unicode, in a single file ▪Organized as a tree ▪Cannot directly store 'native' numbers, must be encoded as unicode	Many organizations, ▪ store numbers in 'native' formats which are typically compact and require no translation in order to perform transfer

XML vs Binary

- XML advantages over Binary
  - XML is self describing and also platform neutral

- Binary advantages over XML
  - Size: Smaller data size, minimal overhead size (ie: no tags)
  - Speed: No need to convert between text and numeric data
  - Simpler: Parsing XML is a complicated exercise
    - And XML can be *too* flexible: multiple ways to do same thing

XML and Binary Data Transfer

- Formats (encoding) for network protocols and data transfer are traditionally binary
  - eg: JRMP (Java)

- XML has been used to define a format for data transmission entirely in text using XML tags
  - eg: SOAP: basis of Web services

Serialization

Serialization is the process of converting an object (or data) into a stream of bytes that can be transferred over a network or saved to a storage medium.

- Serialization is the problem of converting/encoding an object into a single, transportable stream
  - …including any aggregated (contained) objects
  - Used for saving to disk (persistency), network transfer, etc
- Deserialization is the process of (re)creating an object from a serialized string

  recreates the original object from the serialized data.
- Binary serialization

  Data is converted into a binary format and saved or transferred. This format is compact and efficient but typically not human-readable.
  - Common, but suffers from aforementioned problems
- XML serialization

  Data is converted into an XML format, turning objects into human-readable text. This is often used for network data transfer where human readability or compatibility with web-based systems (like web services) is essential.
  - Convert object to a single textual XML description

JavaScript Object Notation

JSON

JSON is open message format

Overview

- What is JSON?
- Comparisons with XML
- Syntax
- Data Types
- Usage

JSON is…

- A lightweight text based data-interchange format
- Completely language independent
- Based on a subset of the JavaScript Programming Language
- Easy to understand, manipulate and generate

JSON is NOT…

- Overly Complex
- A “document” format
- A markup language
- A programming language

Why use JSON?

- Straightforward syntax
- Easy to create and manipulate
- Can be natively parsed in JavaScript using **eval()**

JavaScript executes a string as code.

- Supported by all major JavaScript frameworks
- Supported by most backend technologies

JSON vs. XML

Much Like XML

- Plain text formats
- "Self-describing" (human readable)
- Hierarchical (Values can contain lists of objects or values)

Not Like XML

- Lighter and faster than XML
- JSON uses typed objects. All XML values are type-less strings and must be parsed at runtime.
- Less syntax, no semantics
- Properties are immediately accessible to JavaScript code

JSON vs. XML

Feature	JSON	XML
Syntax	Simpler	More complex (tags)
Readability	Compact	Verbose
Parsing	Faster (native in JS)	Requires parsers
Namespaces	Not supported	Supported

Knocks against JSON

Limitations of JSON

- Lack of namespaces
- No inherit validation (XML has DTD and Schemas, but there is JSONlint for checking syntax)
- Not extensible
- It's basically just ***not*** XML

Syntax

JSON Object Syntax

- Unordered sets of name/value pairs
- Begins with { (left brace)
- Ends with } (right brace)
- Each name is followed by : (colon)
- Name/value pairs are separated by , (comma)

JSON Example

```
var employeeData = {
 "employee_id": 1234567,
 "name": "Jeff Fox",
 "hire_date": "1/1/2013",
 "location": "Norwalk, CT",
 "consultant": false
};
```

Arrays in JSON

- An ordered collection of values
- Begins with [(left bracket)
- Ends with] (right bracket)
- Name/value pairs are separated by , (comma)

JSON Array Example

```
var employeeData = {
 "employee_id": 1236937,
 "name": "Jeff Fox",
 "hire_date": "1/1/2013",
 "location": "Norwalk, CT",
 "consultant": false,
  "random_nums": [ 24,65,12,94 ]
};
```

Data Types

Data Types: Strings

- Sequence of 0 or more Unicode characters
- Wrapped in "double quotes“
- Backslash escapement

Escape Sequence	Description
\"	Double quote
\\	Backslash
\/	Forward slash
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\uXXXX	Unicode character

Data Types: Numbers

- Integer
- Real
- Scientific
- No octal or hex
- No NaN or Infinity – Use **null** instead.

Type	Example	Description
Integer	`100`, `-42`	Whole numbers (positive or negative)
Real (Float)	`3.14`, `-0.75`	Decimal numbers
Scientific (Exponential)	`6.022e23`, `1.5E-10`	Numbers using exponent notation

Data Types: Booleans & Null

- Booleans: true or false
- Null: A value that specifies nothing or no value.

Data Types: Objects & Arrays

```
{
  "name": "Alice",
  "age": 25,
  "city": "New York",
  "isStudent": false
}
```

- Objects: Unordered key/value pairs wrapped in { }
- Arrays: Ordered key/value pairs wrapped in [ ]
- **Can nest objects (supports complex objects)**

```
{
  "fruits": ["Apple", "Banana", "Mango"],
  "scores": [98, 85, 91, 77]
}
```

Nested Objects

```
{
    "stuff": {
        "onetype": [
            {"id":1,"name":"John Doe"},
            {"id":2,"name":"Don Joeh"}
        ],
        "othertype": {"id":2,"company":"ACME"}
    },
    "otherstuff": {
        "thing": [[1,42],[2,2]]
     }
}
```

JSON Usage

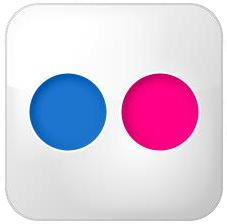
How & When to use JSON

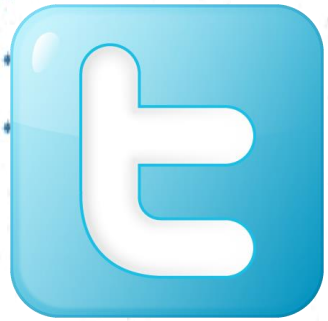
- Transfer data to and from a server
- Perform asynchronous data calls without requiring a page refresh
- Working with data stores
- Compile and save form or user data for local storage

Where is JSON used today?

- Anywhere and everywhere!




And many, many more!

Other open data/message formats

- YAML (yet another markup language)

- https://blog.stackpath.com/yaml/
- https://octopus.com/blog/state-of-config-file-formats

- Often used to define the interfaces of REST apis (will discuss later)
- Defining component/service interfaces is one key usage of open data formats (e.g. XML, YAML)

Summary

- **Binary** data formats (e.g. RMI object-streams) are more **efficient** in sending and receiving messages
- However, **Textual** data formats like XML and JSON are **Open** and **Standardized**, so they facilitate **Interoperability.**
- **XML** has rich set of features such as **namespaces** and **Schemas**.
- **JSON** is more **lightweight** and **efficient**, yet lack Schemas and namespaces
  - YAML is another example for an open data format
  - One key usage of Open data formats is to define service interfaces (e.g. XML, YAML)

Questions?