XML
eXtensible Mark-up Language

Semistructured Data

- Another data model, based on trees.
- Motivation: flexible representation of data.
  - Often, data comes from multiple sources with differences in notation, meaning, etc.
- Motivation: sharing of *documents* among systems and databases.

2

Tags

- Tags, as in HTML, are normally matched pairs, as <FOO> … </FOO> .
- Tags may be nested arbitrarily.
- XML tags are case sensitive.

3

Example: Well-Formed XML

```
<?xml version = "1.0" standalone = "yes" ?>
<BARS>
  <BAR><NAME>Joe's Bar</NAME>
      <BEER><NAME>Bud</NAME>
            <PRICE>2.50</PRICE></BEER>
      <BEER><NAME>Miller</NAME>
            <PRICE>3.00</PRICE></BEER>
  </BAR>
  <BAR> ...
</BARS>
```

A NAME subobject

A BEER subobject

4

XML

- XML allows the content to be structured so that it is easy for a machine to extract meaningful data from an XML page.
- It can be used to structure data in a database, or as a communication language
- It can be formatted using a style sheet language called XSL (like CSS for HTML)

Example

```
<?xml version="1.0"?>
  <email date="30/09/04">
      <to> fred</to>
      <from>sue</from>
      <subject> xml example</subject>
      <message>This is the message</message>
  </email>
```

- All tags have a start and end
- Tags must be correctly nested as a tree
- Tags can have attributes

Example - better

```
<?xml version="1.0"?>
  <email>
      <to>fred</to>
      <from>sue</from>
      <date>
            <day>30</day>
            <month>9</month>
            <year>2004</year>
      </date>
      <subject>xml example</subject>
      <message>this is the message</message>
  </email>
```

Elements ..

Logically every element has three key pieces:

- A name
- The attributes of the element
- The content of the element can be:
  - text,
  - comments,
  - more tagged info or
  - Processing Information
    - **<?xml-stylesheet type="text/xml" href="limited.xsl"?>**
    - This is meta info about the document

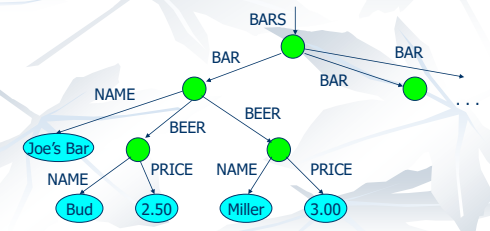
Example Document

```
<BARS>
  <BAR name = "JoesBar">
      <SELLS theBeer = "Bud">2.50</SELLS>
      <SELLS theBeer = "Miller">3.00</SELLS>
  </BAR> …
  <BEER name = "Bud" soldBy = "JoesBar
      SuesBar …"/> …
</BARS>
```

9

Example

- The <BARS> XML document is:



10

XML element might come in seven flavors:

Pattern	XML
1	<e/>
2	<e>text</e>
3	<e name="value" />
4	<e name="value"> text </e>
5	<e> <a> text text </e>
6	<e> <a> text <a> text </e>
7	<e> text <a>text </e>

Semi-Structured XML

- *Semi-structured elements*, are elements with mixed content of text and child elements

```
<e>
  some textual

  <a>content</a>
</e>
```

JSON

JAVASCRIPT OBJECT NOTATION

JSON:

- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.

```
{ "name":"John", "age":31, "city":"New York" }
```

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data - A Name and a Value

- JSON data is written as name/value pairs.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

  "name":"John"
- In JSON, *keys* must be strings, written with double quotes:

  { "name":"John" }

- JSON is built on two internal structures:
  - A collection of name/value pairs with unique names (*associative array*)
  - An ordered list of values (*array*)
- The file type for JSON files is ".json"

JSON Values

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

Valid Data Types

- JSON Strings:
  - Strings in JSON must be written in double quotes. { "name":"John" }
- JSON Numbers:
  - Numbers in JSON must be an integer or a floating point. { "age":30 }
- JSON Objects:
  - Values in JSON can be objects.

```
{
    "employee":{ "name":"John", "age":30, "city":"New York" }
}
```

Valid Data Types

- JSON Arrays
  - Values in JSON can be arrays.

```
{
    "employees":[ "John", "Anna", "Peter" ]
}
```

- JSON Booleans:
  - Values in JSON can be true/false.

```
{ "sale" : true }
```

- JSON null:
  - Values in JSON can be null.

```
{"middlename" : null }
```

Object Syntax

- JSON objects are surrounded by curly braces {}.
- JSON objects are written in key/value pairs.
- Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null).
- Keys and values are separated by a colon.
- Each key/value pair is separated by a comma.

```
{ "name":"John", "age":30, "car":null }
```

Arrays in JSON objects

```
[ "Ford", "BMW", "Fiat" ]
```

- Arrays in JSON are almost the same as arrays in JavaScript.
- In JSON, array values must be of type string, number, object, array, boolean or *null*.

```
{
    "name":"John",
    "age":30,
    "cars":[ "Ford", "BMW", "Fiat" ]
}
```

XML vs. JSON

- JSON is Like XML 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 XMLHttpReques

XML vs. JSON

- JSON is Unlike XML Because
  - JSON doesn't use end tag
  - JSON is shorter
  - JSON is quicker to read and write
  - JSON can use arrays
- The biggest difference is:
  - XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

Why JSON is Better Than XML

- XML is much more difficult to parse than JSON.
- JSON is parsed into a ready-to-use JavaScript object.

Converting JSON to XML

Ex.	XML	JSON
1	<e/>	"e": null
2	<e>text</e>	"e": "text"
3	<e name="value" />	"e": {"@name": "value"}
4	<e name="value"> text </e>	"e": { "@name": "value", "#text": "text" }
5	<e> <a> text text </e>	"e": { "a": "text", "b": "text" }
6	<e> <a> text <a> text </e>	"e": { "a": ["text", "text"] }
7	<e> text <a>text </e>	"e": { "#text": "text", "a": "text" }

Conversion of Arrays

	JSON	XML
Empty array:	[]	nothing in the XML-
Single item array:	[{ "name": "val" }]	<name> val </name>
Multiple item array:	["Arm", "Leg"]	<item> Arm </item> <item> Leg </item>

Conversion Examples:

An attempt to map a structured XML element to the following JSON object, yields an invalid result, since the name "a" is not unique in the associative array.

```
<e>
  <a>some</a>
  <b>textual</b>
  <a>content</a>
</e>
```

```
"e": {
  "a": "some",
  "b": "textual",
  "a": "content"
}
```

Conversion Examples:

So we need to collect all elements of identical names in an array. Using the patterns 5 and 6 above yields the following result:

```
<e>
  <a>some</a>
  <b>textual</b>
  <a>content</a>
</e>
```

```
"e": {
  "a": [ "some", content" ],
  "b": "textual"
}
```

Conversion general rules of thumb

- A structured XML element can be converted to a **reversible** JSON structure, if:
  - all subelement names occur exactly once, or …
  - subelements with identical names are in sequence.

Conversion of Semi-Structured XML

- *XML semi-structured elements*, are elements with mixed content of text and child elements

```
<e>
  some textual

  <a>content</a>
</e>
```

we can apply pattern 7 :

```
"e": {

  "#text": "some textual",

  "a":  "content",

}
```

Arrays in XML and JSON

- JSON arrays are specified using square brackets.
- XML arrays are represented by repeating an element name multiple times.
- XML arrays can also be represented as zero or more elements of the same name wrapped in another element:

```
<Languages>
    <item> JavaScript </item>
    <item> Java </item>
    <item> Python </item>
</Languages>
```

Array Conversion

```
<Languages>
    <item> JavaScript </item>
    <item> Java </item>
    <item> Python </item>
</Languages>
```

{"Languages" : {"item" : ["JavaScript","Java","Python"]}}

More examples:

```
<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>
```

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