XML Basics

I. Introduction

- XML: A W3C standard to complement HTML
- Two facets of XML: document-centric and data-centric
- Motivation
    - HTML describes presentation
    - XML describes content

- User defined tags to markup “content”
- Text based format
- Ideal as “Data Interchange” format.
- Ideal for “distributed” applications (client-server)
- All major database products have been retrofitted with facilities to store and construct XML documents.
- XML is closely related to object-oriented and so-called semi-structured data.

II. Semistructured Data

An HTML document (student list) to be displayed on the Web

```
<dl>
  <dt>John Doe</dt>
  <dd>Id: s111111111</dd>
  <dd>Address:
    <ul>
       <li>Number: 123</li>
       <li>Street: Main</li>
     </ul>
  </dd>
  ...
  ...
</dl>
```

To make the previous student list suitable for machine consumption on the Web, it should have the following characteristics:

- Be **object-like**
- Be **schemaless** (not guaranteed to conform exactly to any schema, but different objects have some commonality among themselves
- Be **self-describing** (some schema-like information, like attribute names, is part of data itself)

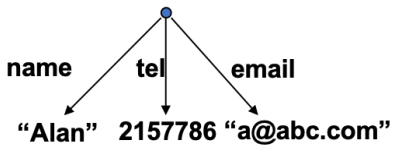
Data with these characteristics are referred to as **semistructured**.

Set of label-value pairs.

```
{ name: "Alan",
  tel: 2157786,
  email: "a@abc.com"
}
```

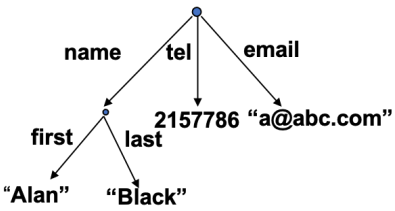
Graph Model:

Nodes represent objects connected by labeled edges to values



The values themselves may be structures.

```
{ name: {first: "Alan", last: "Black" },
  tel: 2157786,
  email: "a@abc.com"
}
```



Duplicate labels allowed

```
{ name: "Alan",
  tel: 2157786,
  tel: 2498762
}
```

The syntax is easily generalized to describe sets of objects

```
{ person: { name: "Alan", tel: 2157786, email: "a@abc.com" },
  person: { name: "Sara", tel: 2136877, email: "sara@abc.com" },
  person: { name: "Fred", tel: 7786312, email: "fred@abc.com" }
}
```

All objects within a set need not have the same structure

```
{ person:{name: "Alan",tel: 2157786,email: "a@abc.com" },
  person:{name: {first: “Sara”,last: “Black”},email: “s@abc.com”},
  person:{name: “Fred”, tel: 7786312, height: 168}
}
```

Relational Data is easily represented

```
{
  r1: { row: {a: a1, b: b1, c: c1},
        row: {a: a2, b: b2, c: c2}
      },

  r2: { row: {c: c2, d: d2},
        row: {c: c3, d: d3},
        row: {c: c4, d: d4}
      }
}
```

Object-oriented data is also naturally represented (each node has a unique object id, either explicitly mentioned or system generated)

```
{
  person: &o1{ name: "Mary", age: 45, child: &o2, child: &o3 },
  person: &o2{ name: "John", age: 17, relatives: { mother: &o1, sister: &o3 } },
  person: &o3{ name: "Jane", country: "Canada", mother: &o1 }
}
```

Semistructured Data Model

Formal syntax for semi-structured data model

```
    <ssd-expr>     ::== <value> | oid <value> | oid
    <value>        ::== atomicvalue | <complexvalue>
    <complexvalue> ::== { label:<ssd-expr>, ..., label:<ssd-expr> }
```

- An oid value is said to be DEFINED if it appears before a value; otherwise it is said to be USED
- An ssd-expression is CONSISTENT if
  - an oid is defined at most once, and
  - If an oid is used, it must also be defined.

A flexible and powerful data model that is capable of representing data that does not have to follow the strict rules of databases.

What is Self-describing Data?

Non-self-describing (relational, object-oriented):

```
    Data part:
         (#12345,  ["Students",   {["John Doe",   s111111111, [123, "Main St"]],
                                    ["Joe Public", s222222222, [321, "Pine St"]] }
                   ]
         )

   Schema part:
      PersonList[ ListName: String,
                  Contents: [ Name: String,
                              Id: String,
                              Address: [Number: Integer, Street: String]  ]
                ]
```

Self-describing:

Attribute names embedded in the data itself, but are distinguished from values.

Doesn't need schema to figure out what is what (but schema might be useful nonetheless)

```
(#12345,
     [ ListName: "Students",
       Contents:  { [ Name: "John Doe",
                      Id:  "s111111111",
                      Address: [ Number: 123, Street: "Main St" ] ] ,
                    [ Name: "Joe Public",
                      Id:  "s222222222",
                      Address: [ Number: 321, Street: "Pine St" ] ]  }
     ]
)
```

III. XML: eXtensible Markup Language

- Suitable for semi-structured data and has become a standard
- Used to describe content rather than presentation
- Differs from HTML in following ways:
  - New tags may be defined at will by the author of the document (extensible)
  - No semantics behind tags. For instance, HTML's `<table>…</table>` means: render contents as a table; in XML: doesn't mean anything special.
  - Structures may be **nested arbitrarily**
  - XML document may contain an optional schema that describes its structure
  - Intolerant to bugs; Browsers will render buggy HTML pages but XML processors will reject ill-formed XML documents.

XML Elements

element: piece of text bounded by user-defined matching tags:

```
<person>
  <name>Alan</name>
  <age>42</age>
  <email>agb@abc.com</email>
</person>
```

Note:

- Element includes the start and end tag
- No quotation marks around strings; XML treats all data as text. This is referred to as PCDATA (Parsed Character Data).
- Empty elements: `<married></married>` can be abbreviated to `<married/>`

Collections are expressed using repeated structures.

Ex. The collection of all persons on the 4th floor:

```
<table>
  <description>People on the 4th floor</description>
  <people>
    <person>
      <name>Alan</name><age>42</age<<email>agb@abc.com</email>
    </person>
    <person>
      <name>Patsy</name><age>36</age><email>ptn@abc.com</email>
    </person>
    <person>
      <name>Ryan</name><age>58</age><email>rgz@abc.com</email>
    </person>
  </people>
</table>
```

XML Attributes

Attributes define some properties of elements

Expressed as a name-value pairs

```
<product>
  <name language="French">trompette six trous</name>
  <price currency="Euro">420.12</price>
  <address format="XLB56" language="French">
    <street>31 rue Croix-Bosset</street>
    <zip>92310</zip>
    <city>Sevres</city>
    <country>France</country>
  </address>
</product>
```

As with tags, user may define any number of attributes

Attribute values must be enclosed within quotation marks.

Attributes vs Elements

- A given attribute can occur only once within a tag; Its value is always a string
- On the other hand, tags defining elements/sub-elements can repeat any number of times and their values may be string data or sub-elements
- Same data may be encoded using attributes or elements or a combination of the two

```
<person name="Alan" age="42">
  <email>agb@abc.com</email>
</person>
```

or

```
<person name="Alan">
  <age>42</age>
  <email>agb@abc.com</email>
</person>
```

XML References

Use `id` attribute to define a reference (similar to oids)

Use `idref` attribute (possibly within an empty element) to refer to a previously defined reference.

Use `idrefs` attribute to refer to a set of references

```
 <state id="s2">          -- defines an id or a reference
    <scode>NE</scode>
    <sname>Nevada</sname>
  </state>

 <city id="c2">
   <ccode>CCN</ccode>
   <cname>Carson City</cname>
   <state-of idref="s2"/>    -- refers to object called s2;
 </city>
```

Mixing Elements and Text

XML allows us to mix PCDATA and sub-elements within an element.

```
<person>
  This is my best friend
  <name>Alan</name>
  <age>42</age>
  I am not sure of the following email address
  <email>agb@abc.com</email>
</person>
```

This seems un-natural from a database perspective, but from a document perspective, this is quite natural!

Order

The semi-structured data model is based on unordered collections, whereas XML is ordered. The following two pieces of semi-structured data are equivalent:

```
    person: {fname: "John", lname: "Smith:}
    person: {lname: "Smith", fname: "John"}
```

but the following two XML data are not:

```
    <person><fname>John</fname><lname>Smith</lname></person>
    <person><lname>Smith></lname><fname>John</fname></person>
```

To make matters worse (-:, attributes are NOT ordered in XML; Following two are equivalent:

```
    <person fname="John" lname="Smith"/>
    <person lname="Smith" fname="John"/>
```

Other XML Constructs

Comments:

```
    <!-- this is a comment -->
```

Processing Instruction (PI):

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="classes.xsl"?>
```

Such instructions are passed on to applications that process XML files.

CDATA (Character Data):

used to write escape blocks containing text that otherwise would be considered markup:

```
<![CDATA[<start>this is not an element</start>]]>
```

Entities:

`<` stands for `<`

Well-Formed XML Documents

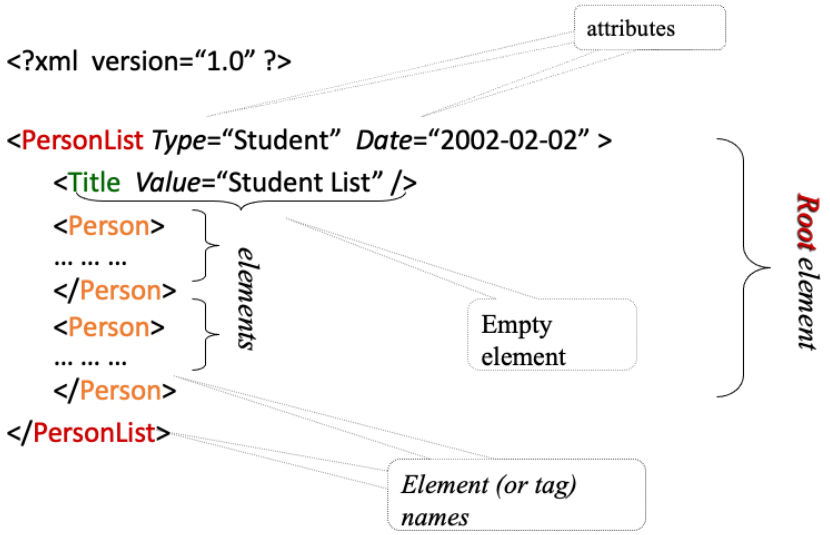
An XML document is **well-formed** if

- Tags are syntactically correct
- Every tag has an end tag
- Tags are properly nested
- There is a root tag
- A start tag does not have two occurrences of the same attribute

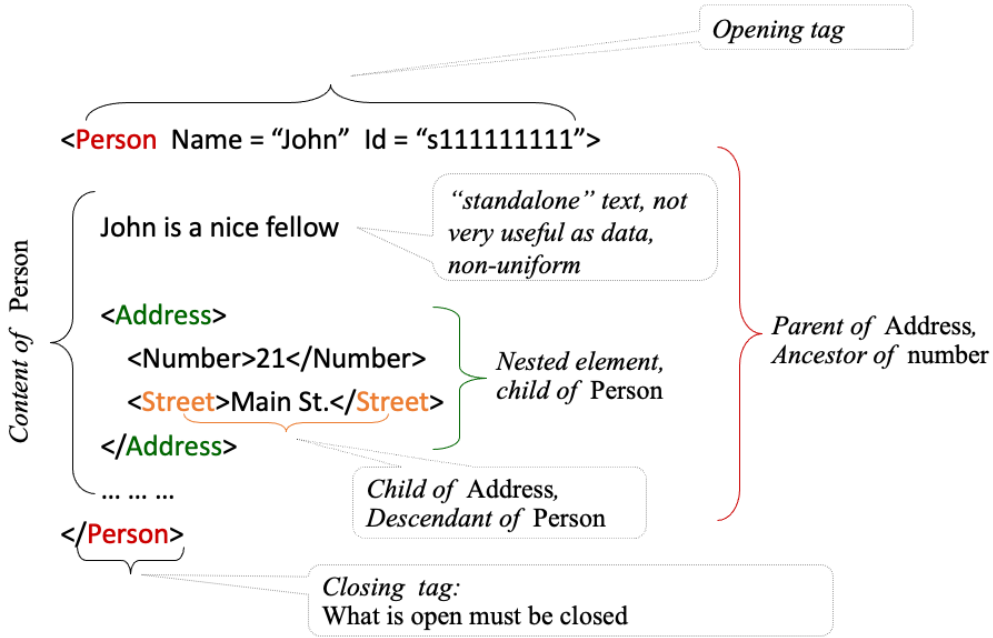
An XML document must be well-formed before it can be processed.

A well-formed XML document will parse into a node-labeled tree

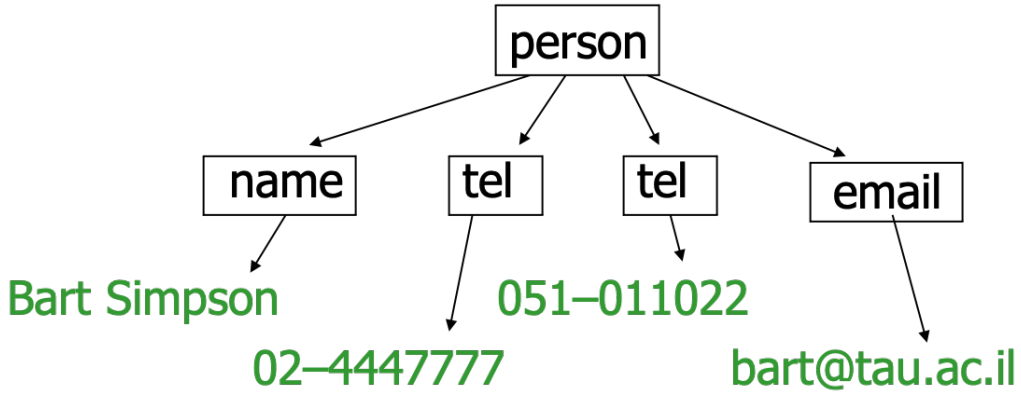
Terminology



- Elements are nested
- Root element contains all others



XML Data Model (DOM Tree)



- Document Object Model (DOM) – DOM Tree
- Leaves are either empty or contain PCDATA
- Unlike ssd tree model, nodes are labeled with tags.