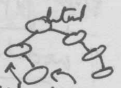
XML essentials

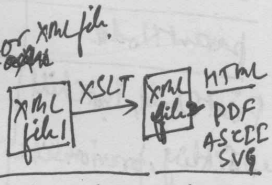
XML (self-describing well structured data)

- you get to make your own tags (or you can use an existing tag set to solve a particular problem)
- to structure and describe information



Related XML technologies –

- XPath (eXtensible Path language) = like query language, SQL (XQuery / XPointer) /html/head/title
  - used to extract data from inside an XML file
  - uses a path-like syntax ("drive:/folder/folder/file")
- XSLT (eXtensible Stylesheet Language Transformations)
  - styling language that take XML file and transforms it into something else



- XQuery – used to perform query functions on XML data, similar to SQL for databases.
- XPointer and XLink – used for creating hyperlinks to XML documents and arbitrary points within XML documents

Tools –
- notepad, TextPad, BBEdit.
- Adobe Dreamweaver
- Altova XMLSpy
- MS Visual Web Developer Express

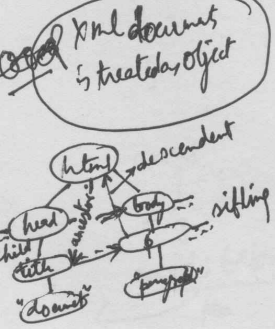
working with data islands = (a feature = embed XML info. directly into webpage and connect/bind it to layout elements)

- XML itself does so also.
- uses <XML> tag to enclose the data
- datasrc and datafld attributes are used to define a binding to XML data.
- a variety of elements can be used to bind to the content of Data islands (A, APPLET, BUTTON, DIV, FRAME, IFRAME, IMG, INPUT (button, checkbox, hidden, password, radio, text), LABEL, LEGEND, MARQUEE, SELECT, SPAN, TABLE, TEXTAREA)
- can control the amount of data displayed by (datapagesize) attribute
- can also be bound to non-table elements and display just one record at a time

Document Object Model – DOM – independent of platform, browser, language

- a tree structured document and allows access to objects in tree
- all XML content (elements, comments, processing instructions, CDATA, ...) as object.
- object (node) has parent-child relationship
- DOM uses terms (parent, child, sibling, ancestor, descendant, ... to relate nodes)

XML document is treated as object



used in XPath ⇒ (axis = relationship between context node and nodes selected by expression.
predicate = further refinement you can make to selection process.)

= Important DOM node properties =

Property/Function	Description
nodeName	"#comment" "#text"
nodeValue	(value)
nodeType	(in integer)
childNodes	Array of child nodes
parentNode	- parent node
firstChild, lastChild	properties that return first and last child
nextSibling, previousSibling	" " " next and previous siblings (Sibling = sit at the same level)
attributes	array of attributes for current node
appendChild()	add new node as child
removeChild()	remove " "
insertBefore()	insert new " before a child
replaceChild()	replace node with a new node

= Important document properties =

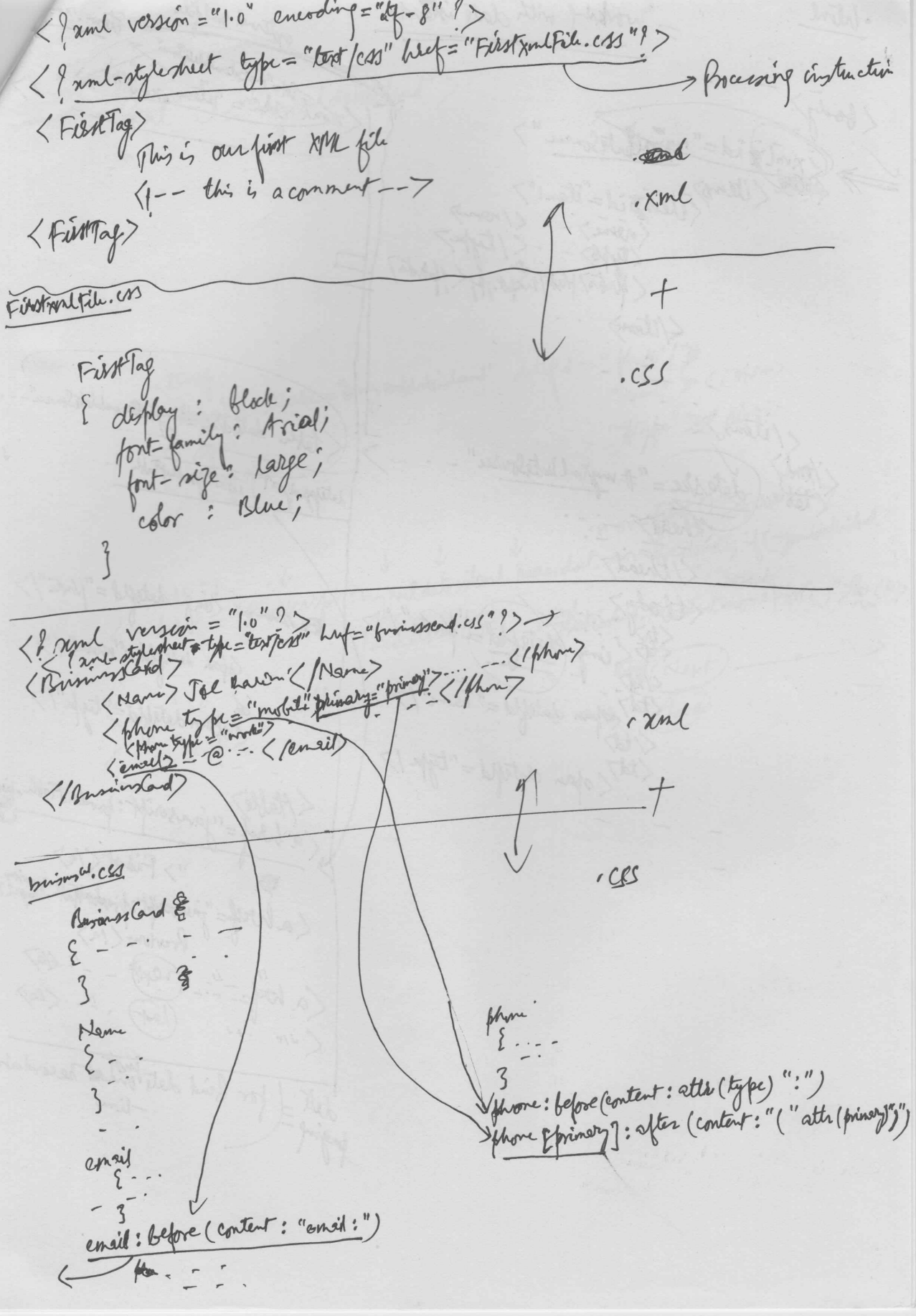
Property / function	description
documentElement	document's root node
createElement()	create new element to be inserted into document
createTextNode()	create new textNode
createComment()	create new comment node
createCDATASection()	create new CDATA section node
getElementsByTagName()	Returns array of element nodes that match given tag name
getElementById()	Returns reference to element object in document given an ID to look for.

for elements:

getAttribute = a reference to document's root node
setAttribute = create a new element
removeAttribute = create new text node
getElementsByTagName() = returns array of element nodes that match the given tag name
innerHTML = HTML content inside of the given element

for Text, comment, and CDATA nodes:

data = text data of the node


<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/css" href="FirstXmlFile.css" ?>
Processing instruction
<FirstTag>
This is our first XML file
<!-- this is a comment -->
</FirstTag>
.xml
+
.css
FirstXmlFile.css
FirstTag
{ display : block;
font-family : Arial;
font-size : large;
color : Blue;
}
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" href="businesscard.css" ?>
<BusinessCard>
<Name> Joe Karlsson </Name>
<phone type="mobile" primary="primary">...</phone>
<phone type="work">...</phone>
<email>...@...</email>
</BusinessCard>
.xml
+
.css
businesscard.css
BusinessCard
{ }
Name
{ }
email
{ }
email : before (content : "email :")
phone
{ }
phone : before(content : attr(type) ":")
phone [primary] : after (content : "(" attr(primary)")")

• html = working with data islands

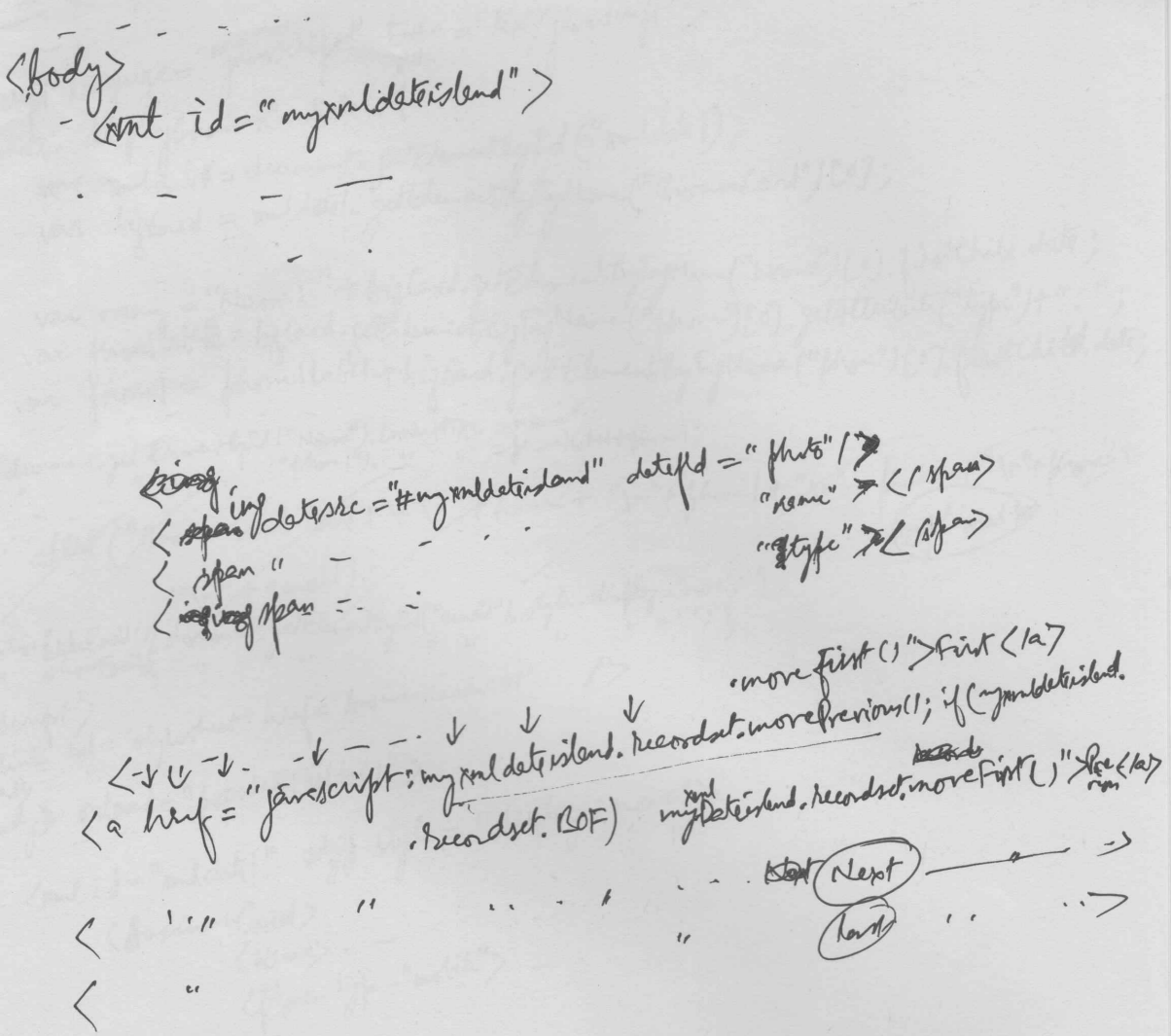
external xml file

```
<body>
  <xml id="myXmlDataSource">
    <items>
      <item id="item1">
        <name> </name>
        <type> </type>
        <photo> photo/image.jpg </photo>
      </item>
    </items>
  </xml>
  <table datasrc="#myXmlDataSource"
         datapagesize="4" id="productTable">
    <thead> </thead>
    <tbody>
      <tr>
        <td> <img datafld="photo" /> </td>
        <td> <span datafld="name" /> </td>
        <td> <span datafld="type" /> </td>
      </tr>
    </tbody>
  </table>
  <a href="javascript:productTable.firstPage()"> First </a>
  <a href="javascript:productTable.previousPage()"> Previous </a>
  <a href="javascript:productTable.nextPage()"> Next </a>
  <a href="javascript:productTable.lastPage()"> Last </a>
</body>
```

items.xml

```
<xml id="myExternalDataSource" src="items.xml" />
```

data paging = for find data just one record data - time


Next
Last

xhtml

```
<script language="javascript" type="text/javascript">
function displayBusinessCardData() {
    var xmldata = document.getElementById("xmldata");
    var bizCard = xmldata.getElementsByTagName("BusinessCard")[0];

    var name = "Name: " + bizCard.getElementsByTagName("Name")[0].firstChild.data;
    var phone1Label = bizCard.getElementsByTagName("phone")[0].getAttribute("type") + ": ";
    var phone1 = phone1Label + bizCard.getElementsByTagName("phone")[0].firstChild.data;

    document.getElementById("Name").innerHTML = name;
    document.getElementById("phone1").innerHTML = phone1Label + phone1;

    alert("BusinessCard Data: \n\n" + name + "\n" + phone1 + "\n" + phone2 + "\n" + phone3 +
          "\n" + email);
}
function hideEmail() { document.getElementById("email").style.display = "none"; }
function showEmail() { document.getElementById("email").style.display = ""; }
</script>
<link rel="stylesheet" href="businesscard.css" />
</head>
<body onload="populateFields()">

<xml id="xmldata" style="display:none">
    <BusinessCard>
        <Name></Name>
        <phone type="mobile"></phone>
    </BusinessCard>
</xml>

<a href="javascript:displayBusinessCardData()">...</a>

<div class="BusinessCard">
    <div class="Name" id="Name"></div>
    <div class="phone" id="phone1"></div>
</div>

<a href="javascript:hideEmail()">Hide email</a>
<a "  "  showEmail()  ....</a>
```

optional

XPath examples –

/ ⇒ find root tag in document

/root_tag ⇒ " ", but only if it is named "root_tag"

// element_a ⇒ find all element_a tags, wherever they appear in the file

text() ⇒ selects text content of current node

@ name ⇒ selects "name" attribute of current node

/doc/chapter[5]/section[2] ⇒ select second section of fifth chapter of the doc

body/p[last()] ⇒ select last "p" tag in the "body" tag

.. ⇒ select parent of current node

/html/body/p[@class="a"] ⇒ select all "p" tags in body under html tag that have class attribute whose value is "a"

//p[@class and @style] ⇒ select every "p" tag in document that has both class attribute and style attribute.

XSLT – similar to CSS but a little bit different

– CSS use rules that are applied to elements

// = XSLT uses templates that transform a source tree into a result tree.

XSLT element =

– <xsl:stylesheet>

– <xsl:template name="name" match="xpath">

– <xsl:value-of select="xpath">

– <xsl:attribute>

– <xsl:text>

– <xsl:for-each select="xpath">

– <xsl:if test="condition">

– <xsl:choose>

– <xsl:when>

– <xsl:otherwise>

– <xsl:sort select="xpath">

– <xsl:call-template name="template name">

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/SimpleTag">
    <html><head>
      <title>XSLT Example</title>
      </head>
      <body>
        <h1><xsl:value-of select="text()"/></h1>
      </body></html>
  </xsl:template>
</xsl:stylesheet>
```

← XSLT template example

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="simple.xslt"?>
<SimpleTag>Hello!</SimpleTag>
```

.xml

DTD (Document Type Declaration)

```
<?xml version="1.0"?>
<!DOCTYPE BusinCard SYSTEM "file.dtd">
```

save in (file.dtd) →

```
<!ELEMENT BusinCard (Name, phone+, email?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ATTLIST phone type (mobile | fax | work | home) #REQUIRED>
<!ELEMENT email (#PCDATA)>
```

```
<BusinCard>
    <Name> Joe </Name>
    <phone type="mobile"> 417- - - </phone>
    <phone type="work"> 870- - - </phone>
    .
    .
    .
    <email> joe@gmail.com </email>
</BusinCard>
```

= XML Schema (definition) = .XSD

- alternative to using DTD
- allows you to constrain content of XML documents, just like DTD
- more powerful (much finer level of control)
- written using XML syntax

(schema and xml files are always stored separately)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLScheme">
    <xsd:element name="root_tag">
        other content definitions
        <xsd:attribute name="attrName" type="type" />
    </xsd:element>
</xsd:schema>
```

Schema elements:

- xsd: string
- xsd: boolean
- xsd: decimal
- xsd: integer
- xsd: positiveInteger
- xsd: negativeInteger
- xsd: date
- xsd: time
- xsd: dateTime
- xsd: gMonth
- xsd: gYear
- xsd: gDay
- xsd: gYearMonth
- xsd: anyURI
- xsd: restriction
- xsd: enumeration
- xsd: pattern
- xsd: anyType
- xsd: complexType
- xsd: attribute name= ... />
- xsd: sequence ... >

minOccurs = "1" } for occurrence of elements
maxOccurs = "5" }
"unbounded"

SAX (Simple API for XML)

- takes one pass over document from start to finish
- once SAX starts, it goes through all the way to the end
- SAX tells you each time it encounters a certain kind of document content

1. write handler functions that will process each type of XML data (element, CDATA, etc.)
2. Create SAX parser instance
3. load XML data to be parsed and hand it off