Week 1: Lecture Notes

Introduction to XML, services and applications

Since XML was first developed by an XML Working Group formed under the auspices of the World Wide Web Consortium (W3C) in 1996, a number of papers, applications and what have come to be called XML services have been developed or proposed. It would be difficult to have had anything to do with the Web in the last few years and not have heard of XML. However, unless you are a professional working closely with the developing standards, it is difficult to sometimes see the direction and place all these standards and technologies in perspective. In this first week, we will briefly discuss XML and provide a brief overview to XML Web services, their architecture and the developing standards and technologies which are just becoming known as XML applications.

What is XML?

XML is an acronym for Extensible Markup Language and it is a subset of the Standard Generalised Mark-up Language (SGML), developed by Charles F. Goldberg. The first working draft of which was published in 1980 and in 1986, it was accepted as a standard (ISO 8879:1986) for the structure and content of electronic documents. HTML (Hypertext Markup Language) and XML have both been defined using SGML, and conform to SGML rules. SGML, which was designed for the world of publishing, is too complex for the fairly simplistic world of the Web, so we have been using HTML, developed by Tim Berners-Lee at CERN (the European Organization for Nuclear Research) in 1990, to create the first Web pages. HTML was specifically designed to display format data on black and white monitors and it has a limited set of predefined tags. XML, on the other hand, is used to define the data as distinct from the formatting.

These are the fundamental differences between HTML and XML. HTML is intended to present hyperlinked formatted information to humans using dumb terminals, but it led to the development of the first browser, lynx and Mosaic. The former was designed for dumb terminals and the latter for graphics terminals. XML is for both computers and humans, and while it does not provide formatting information, it can provide validation and is extensible so that the number of tags is not limited. It has much in common with a hierarchical database like IBM's IMS and indeed an XML document can be looked as a hierarchical database and there have been some XML databases developed.

XML is relatively simple, yet (probably because of the simplicity) the range of applications and potential for XML is enormous. As a markup language solely to describe data, XML is ideal for the interchange of data in the distributed environment of the Web and other Networks and between diverse applications, including hierarchical and Relational databases.

There were electronic exchange methods before XML, using amongst others Electronic Data Interchange (EDI) developed in the early 1970s by the Transport Data Coordinating Committee and running over the X.25 network which preceded the Internet. Both EDI and X.25 and its successor Frame Relay are still well used. EDI was developed for use with mainframe computers, often in secure environments of government and commerce. Unlike the Internet, X.25 is a reliable network that guarantees accurate delivery and as such, was just right for delivering important numeric data between government and business. However, the systems were not as flexible and although there were International standards, the national implementations often had significant differences so that cross-border data exchange was problematic.

EDI is still in use, as well as X.25 networks and the high speed Frame Relay for secure private commercial networks. However, XML-based systems running over the Internet are rapidly becoming the method of choice.

XML services

The X.25 and Frame Relay networks were built as reliable networks that guaranteed delivery over well-defined infrastructure. The Internet and the Web is not like that. The Internet delivery mechanism is unreliable and errors have to be corrected at the transport layer on an end-to-end basis, leading to order of data delivery. Moreover, the Internet utilizes a diverse infrastructure where any computer can connect, so no one company controls the overall system.

XML allows us to exchange well-formatted messages between systems where the format is described within the document itself. This allows message exchanges between systems regardless of the environment on each system in the exchange.

With the thousands of applications that were developed using XML, some XML applications were so general-purpose that they developed into a standard in their own right, and many of these have become known as XML services.

What is a Web service?

> A Web service is a modular application that is self-describing, and can be published, located and invoked for anywhere on the Web or a local area network. The provider and the consumer of an XML Web service don't have to worry about the environment on either machine as the service is based on open Internet standards such as XML and HTTP.
>
> —Cauldwell, 2001

In simpler terms, an XML service is an application accessible to other applications over the Internet. XML service takes a new approach to the old business problem of application integration, and more recently the question of software reuse. Software reuse has always been one of the fundamental goals of object-oriented architectures, but has never been quite attainable. Coding for software reuse is a good idea and should save money in the long term; however, this assumes that the

business requirements of the environment in which such software will operate are stable. This is not the case, and attempting to design components to handle future requirements is not only difficult, but usually results in wasted effort.

Similarly, software integration is also difficult to achieve. The business world is awash with legacy systems and modern applications, a result of business-critical older applications that are too big to replace and have been layered with years of updates and modifications using newer technology. One way to achieve integration is with XML services, which will replace the current models of application design with a more flexible architecture. This should result in software systems that are more flexible in a changing environment.

We shall look at Web services in more detail in a later lecture, but for the moment, we need to learn a little about the underlying XML.

Markup versus content

An XML document contains two types of elements: content and markup. The content is the actual data while as mentioned, the markup adds meaning to the data and surrounds it. A markup elements looks as follows:

```
<city>
```

Thus, a markup element appears between an opening angle bracket (less than sign) and a closing angle bracket (greater than sign). When we say that markup surrounds the content (data), we mean that the content would appear in the XML document as follows:

```
<city>Manchester</city>
```

In the above, "Manchester" is data (or content), and by itself does not mean anything. The context, or meaning, is provided by the surrounding markup, the **<city>** and **</city>** markup, also referred to as tags. The data is given context by the markup. It should be clear that in the above, Manchester is a city. Thus, the XML document is self-describing. Notice also that the markup tags come in pairs. The **<city>** is an opening tag followed by the data the tag is giving context to, and **</city>** is the closing tag which delineates the end of the data associated with the **<city>** tag. For each opening tag, there must be an associated closing tag. This is necessary for an XML document to be well-formed. We will discuss well-formed documents later.

An example

The best way to begin learning about XML in detail is to begin to see examples. The following listing (Listing 1) shows an XML document that contains a payroll record for a particular employee. Now obviously, this is not a complete payroll record as might appear in an actual payroll system; only a small sample of data that might be kept is used to highlight important aspects.

Listing 1: XML payroll record sample

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

<pay_record>
    <employeeNum>125-34</employeeNum>
    <name>
        <firstName>John</firstName>
        <lastName>Barlow</lastName>
    </name>
    <address>
        <street>12 City Road</street>
         <city>Manchester</city>
        <state>England</state>
        <zip>M70 2PP</zip>
    </address>
    <status>Full Time</status>
    <position>
        <title>Tool Designer</title>
        <description>Designs custom tools for Manufacturing</description>
        <workdays>
            <day>Monday</day>
            <day>Wednesday</day>
            <day>Friday</day>
        </workdays>
    </position>
    <salary currency="GBP">32000</salary>
    <married />
</pay_record>
```

Now in Listing 1, there are a number of things to note. First, as mentioned previously, every markup tag comes in pairs, the opening and closing tags. The data is hierarchical. For example, payroll record contains elements such as **name**, **address**, **status** et cetera, but elements like **name** contain further subelements such as **firstName**, **LastName** and so on. The document is also extensible as we can add further elements to the existing definition of pay record, and subelements to some of the elements. Also, the inclusion of the metadata (markup) adds context to all the content. For example, without the markup, the document would look as follows:

Listing 2: Payroll record without markup

```
125-34
John
Barlow
12 City Road
Manchester
England
M70 2PP
```

Full Time
Tool Designer
Designs custom tools for Manufacturing
Monday
Wednesday
Friday
32000

Without the metadata, there is nothing to give meaning to the rest of the data. In this application, you could probably determine that John Barlow is a name, which is followed by something that looks like an address, but what about the rest of the data? What is 125-34 and 32000, and what is the significance of Monday, Wednesday and Friday? Also, is the name and address for a customer, bank account or holiday booking? As you can determine, an XML document is self-describing.

In Listing 1, notice how parts of the document are indented. For example, the **firstName** and **lastName** tags are indented to the right between the **<name>** and **</name>** tags. This is not a requirement of XML, but is usually done solely for the purposes of highlighting the hierarchical nature of the content. It is only for the human reader that this is done, in the same way that programming source code is indented. However, when creating XML documents by hand, it is always a good idea to indent, if only for yourself to keep track of what you are doing.

Now one last thing to note before we discuss the specific parts of the XML document in detail: As an XML document is a markup document, it is a plain ASCII text document, designed to be created and read in a plain text document reader such as Windows Notepad. This is because it was the intention that XML documents be easy to create and read. This also means that XML documents which are passed and transmitted over the Internet are very expensive in terms of storage and bandwidth. Twenty years ago, such storage and transmission would be unheard of; however, storage space is a cheap commodity today, and bandwidth is not such a problem, so XML has evolved as a *product of the times*.

XML document structure

An XML document has a definite structure that it must conform to, as can be seen in Listing 1.

An XML document is structured into the following sections:

- the XML declaration
- the document type declaration, or DOCTYPE (optional)
- the XML document body which contains markup and content

The XML declaration

All XML documents begin with an XML declaration, which is a processing instruction, to ensure that XML documents are read properly by XML compliant tools. Editors, browsers and document processors use document declarations to determine how a document should be processed. XML declarations MUST always start at the beginning of the first line in the XML document. Any space, even non-displaying space, may cause the parser to fail. Although strictly not required, the declaration explicitly identifies this document as an XML document, and every XML document should use one.

The XML declaration has the following format:

```
<?xml version= "xxx"  standalone="xxx"  encoding="xxx" ?>
```

The **standalone** and **encoding** components are optional, and of course are not used in Listing 1.

Version: shows the version in use. V1.0 is common; V1.1 has relaxed criteria; V2.0 is under discussion.

Encoding; states the alphabet in use, which may be UTF-8, UTF-16, ISO 8859-1 to ISO 8859-9, UCS-4, Windows-1252, etc.

XML 1.0 is the most common version in use. The later version, 1.1, provides for non-Unicode character sets and is not generally required.

If the **encoding** attribute is omitted, the default is the standard US-ASCII, but as this is a limited character set, it is generally important to specify the correct encoding. UTF-8 and UTF-16 are the most common and support single-byte and double-byte Unicode, respectively.

The **standalone** attribute indicates whether other documents will be used in determining what is valid for the current document. XML documents can use Document Type Definitions (DTDs) to define the set of valid tags and so determine if the content of the current document is valid. We will cover DTD in detail later, but if the **standalone** attribute is set to "yes," then the document is standalone and doesn't use an external document to validate the content. A standalone document should have an internal DTD included. If it doesn't, then the document is not validated against any criteria, and all the markup tags within the document are accepted as is, provided they conform to well-formed rules, which we'll cover later. If the **standalone** attribute is omitted, then the default is **standalone="yes"**.

The Document Type Declaration

After the XML declaration, we can optionally include a document type declaration. If the XML document has an associated DTD (either external or internal), then we'll need to include one. The XML document in Listing 1 doesn't have a DTD, so no

document type declaration was included. A document type declaration looks like the following

<!DOCTYPE *root-element* SYSTEM "*file*" [ ] >

Now not all of the attributes in the above type declaration are always needed. However, if a type declaration is used, the **root-element** must be included. The **root-element** is the first tag that appears in the XML document body. In Listing 1 above, it would be **pay_record**. We will discuss the tree-like structure of the document body shortly.

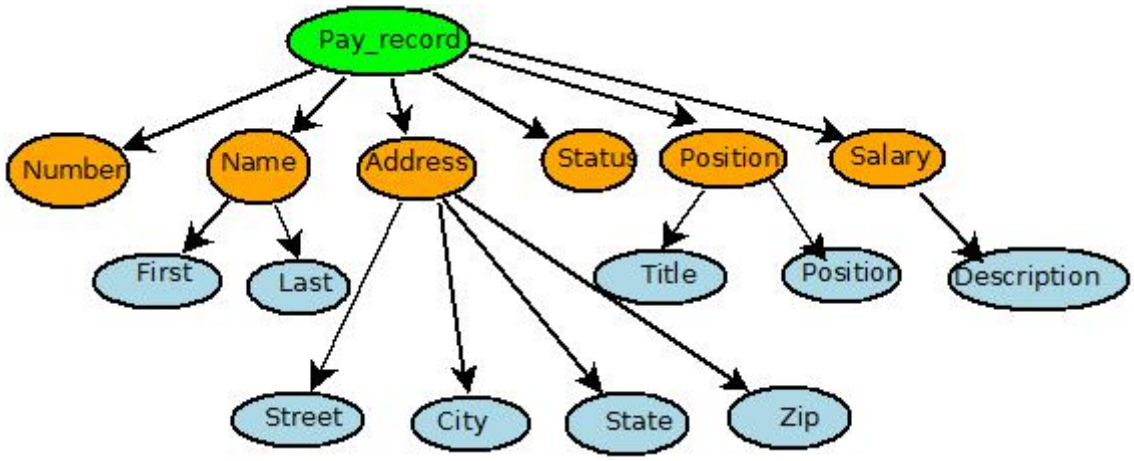
If there is an associated external DTD, then the **SYSTEM** attribute is included, followed by **file**, which is the URL (Uniform Resource Locator) of the external file containing the DTD for this XML document.

If the XML document has an internal DTD, then it is included in the square brackets **[ ]**. An XML document can have an external DTD, an internal DTD or both. If it contains both, then the internal DTD will override the external DTD where there is a conflict.

The XML document body

The remainder of the XML document, the document body, is made up of the markup tags and the content which is formatted in a tree-like structure. Figure 1 below represents the structure of the document we just created in Listing 1. An XML document has a single root element, which contains all of the document's other elements.

Figure 1: XML document structure



The structure of an XML document is essentially a tree. The root element is the top-level element (in this case, the **<pay_record>** element). Its descendants (the other elements) branch out from there. When creating XML documents, it is important to remember you are creating a tree structure. XML is not about display; it is about data

and its hierarchical organization. The root element **<pay_record>** is the root element that appears in a document type declaration if one is included. An XML document can only contain one root element.

An XML element, as mentioned previously, is made up of a start and an end tag with content in between. The tags describe or add context to the data contained within, which is essentially the value of the element:

```
<street>12 City Road</street>
```

This example has **street** as the element and "12 City Road" as the value. This allows you to mark the value of "12 City Road" so that you can differentiate this data from other related data. There are also empty elements which contain no data but may contain attributes. Examples include the XHTML tags **
** and ****. The trailing **/>** indicates that the XML processor should not expect a separate end-tag.

An XML element can contain attributes in the start tag, as in the **<img>** tag above, to provide extra information about the data. For example, in Listing 1, note the line:

```
<salary currency="GBP">32000</salary>
```

In the **salary** element, we have the attribute **currency**. The **currency** attribute gives us extra information about the content—in this case, the currency base of the salary, which is in pounds sterling, GBP. This could markedly change the meaning of the content, say, if the attribute was set to US dollars (USD). Attributes often provide us information which is not part of the data. There are some simple syntactical rules to follow about using attributes.

Attribute values must always be enclosed in quotes, but either single or double quotes can be used. For example, both of the following lines are valid

```
<salary currency="GBP">32000</salary>
<salary currency="GBP">32000</salary>
```

Double quotes are the most common, but sometimes (if the attribute value itself contains quotes), it is necessary to use single quotes, like in this example

```
<firstName alias="'Jonners"'>John</firstName>
```

There is always a debate over whether data should be stored in child elements or in attributes. For instance, in the **<name>** element in Listing 1, if we also wanted to indicate the sex of the employee, we could use either an attribute or a child element as in:

```
<name>
        <firstName>John</firstName>
        <lastName>Barlow</lastName>
```

```
        <sex>Male</sex>
</name>

<name sex="Male">
        <firstName>John</firstName>
        <lastName>Barlow</lastName>
</name>
```

Both methods are acceptable and valid XML syntax. But which is the more acceptable? There are no specific rules governing this, so a lot will depend on the actual application being developed. A general rule of thumb to use is 'if it feels like data, then use elements'. In other words, use elements to describe data. If the attribute value is really part of the data you're describing in your XML document, then you should be using elements. However, there are always exceptions to any rule, but just don't end up with something like this

```
<name sex="Male" birthdate="27/5/1960" eyeColor="Blue" hairColor="Brown"
middleInitial="F">
        <firstName>John</firstName>
        <lastName>Barlow</lastName>
</name>
```

Now a couple of last points about elements (or markup tags). XML does not limit you to a set library of tags. When marking up documents in XML, you can choose the tag name that best describes the contents of the element. For example, in the **address** element of Listing 1, I've used the **<zip>** tag to indicate the postal area code as in:

```
<zip>M70 2PP</zip>
```

However, I am free to use something else, such as

```
<post_code>M70 2PP</post_code>
```

XML markup tags are *case sensitive*. This means that XML distinguishes between upper case and lower case characters in the tags. Thus, the following tags are all different:

```
<name>
<Name>
<naMe>
```

You must be careful and consistent because something like the example that follows would lead to an invalid XML document as the opening and closing tags don't match.

```
<name>
  :
</Name>
```

‘What are the naming rules for tags’? XML element names can contain letters, digits and other characters, but names cannot start with a punctuation symbol or a digit. Names cannot contain a space since XML would then be confused over whether you actually have a space in the name or are defining an attribute. XML elements cannot start with the letters ‘XML’.

XML content can consist of any data at all, including binary data. As long as it doesn’t violate any rules that would then confuse the data with the metadata (or markup tags).

It is important to remember that XHTML tags are always lower case, so that **
** is correct and **
** is not acceptable in either XHTML or XML.

XML comments

Sometimes you want to place comments inside your XML document that are only for the human readers and are not to be processed or passed on by any application reading the XML document. Comments can be added to your XML document as follows:

```
<!-- XML example of a comment -->
```

The above example shows the syntax of comments. In between <!-- and -->, you can write any comment you want. Comments can be placed between markups anywhere in your document and they are ignored by any application processing your document.

Well-formed and valid XML

Well-formedness is a fundamental requirement of all XML documents. A well-formed XML document is one that has been constructed according to the syntax rules we have been discussing:

- All XML elements must be contained with matched start and end tags.
- XML tag names are case sensitive.
- XML elements must be correctly nested; interleaving is not allowed.
- All XML documents must have a single root element.
- Attribute values must always be within a start tag or an empty tag.

These brief definitions are based on the standard, which goes into considerably more detail, but these are sufficient for the moment

A valid XML document is a different matter. A well-formed document may not necessarily be valid, because there is nothing to validate it against. For instance, the XML document in Listing 1 is well-formed but not valid. If we used this in some payroll processing application, the use of the **<salary>** tag may be invalid because the payroll processing package does not accept attributes such as **currency** that we have used. Thus, although the document is well-formed according to all the syntax

rules of XML, it's not valid for the applications. A valid XML document must contain a proper Document Type Declaration with information about the DTD. And if all the constraints expressed in the DTD are satisfied, then the XML document is valid. Again, we will discuss DTDs and validity later.

Parsing XML

Before we can use any XML document that we create in an application, the XML document needs to be parsed. Parsing an XML document is a process of reading the XML document by software and building up a computer-based representation of the document (the computer equivalent of the tree-structured model shown in Figure 1) and checking that the document is well-formed, (conforms to all the syntax rules) and is valid if there is an associated DTD. If the XML document passes all these checks, then it has been parsed and is ready for processing or manipulation.

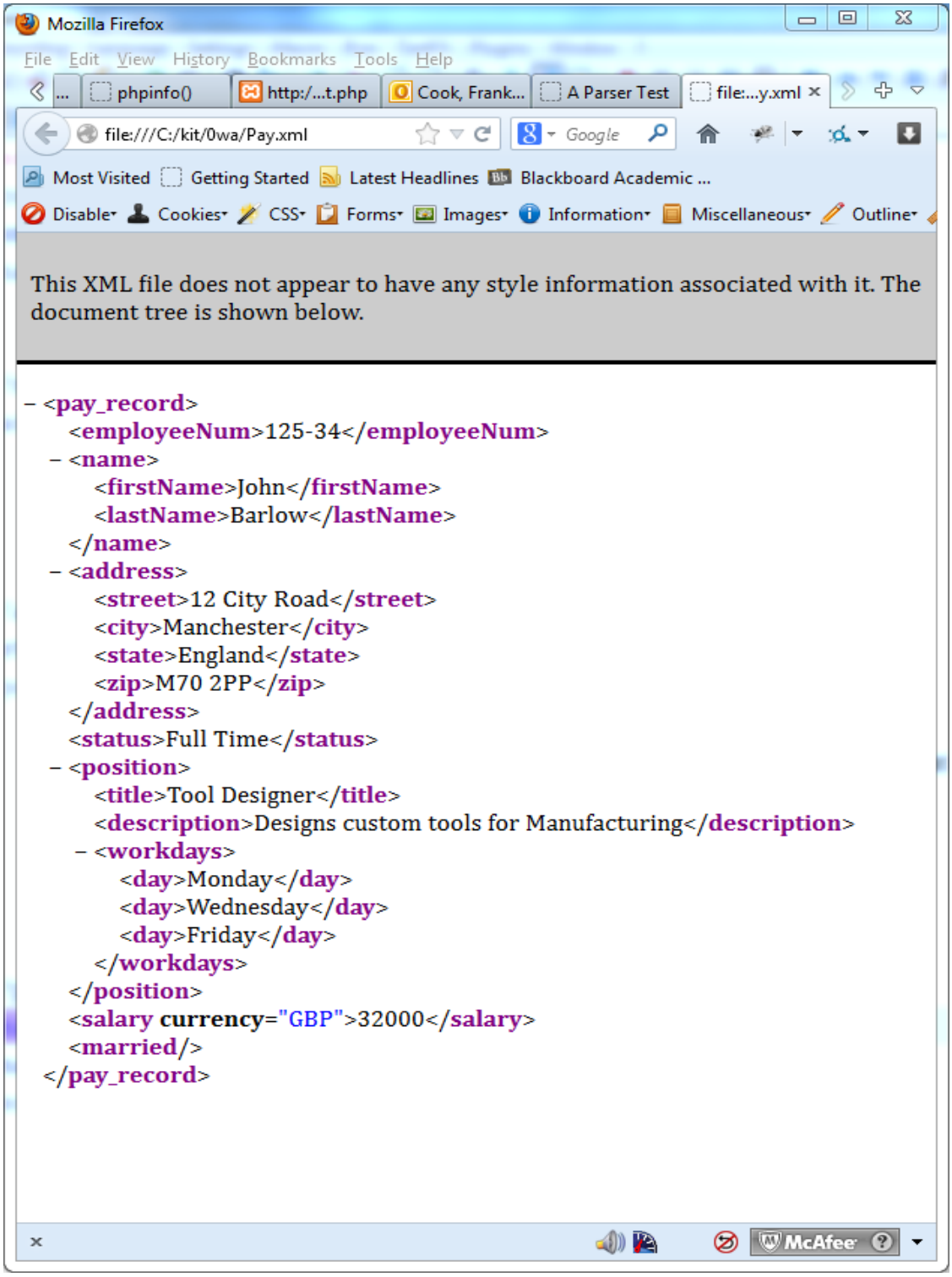
All XML applications contain an XML parser to parse XML documents before they are used to check that they are in fact well-formed or valid. Now we don't have an XML application ready to use the **pay_record** we created in Listing 1, but we could obtain just an XML parser to check the well-formedness of the document. There are a number of XML parsers both commercial and free, but most modern browsers—Opera, Firefox, Chrome, Safari and IE—have an XML parser built in. Be aware that these are all different and may render your XML differently.

Copy the code from Listing 1 above and paste it into a document called *pay_record.xml* (all XML files have the extension *xml*). This should load in your chosen browser and display the XML file or an error message if there is a problem.

```
<?xml version="1.0" ?>
- <pay_record>
    <employeeNum>125-34</employeeNum>
  - <name>
      <firstName>John</firstName>
      <lastName>Barlow</lastName>
    </name>
  - <address>
      <street>12 City Road</street>
      <city>Manchester</city>
      <state>England</state>
      <zip>M70 2PP</zip>
    </address>
    <status>Full Time</status>
  - <position>
      <title>Tool Designer</title>
      <description> Designs custom tools for Manufacturing </description>
    - <workdays>
        <day>Monday</day>
        <day>Wednesday</day>
        <day>Friday</day>
```

```
        </workdays>
    </position>
    <salary currency="GBP">32000</salary>
    <married />
</pay_record>
```

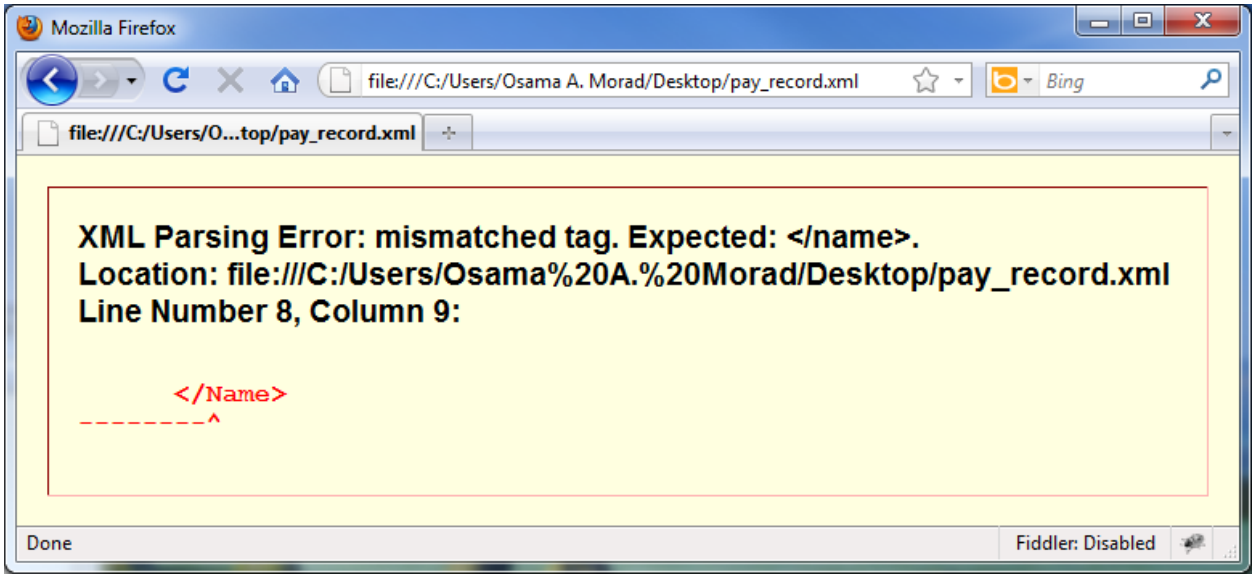
Listing 3: Output from browser of *pay_record.xml*



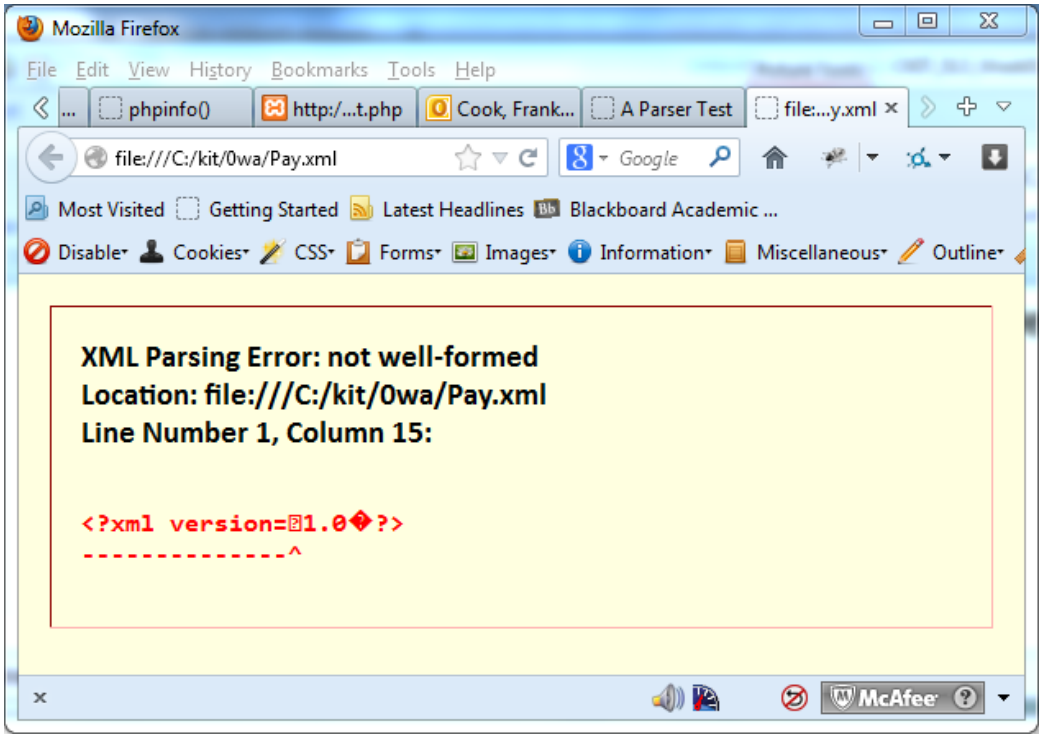
Listing 3 shows the output from the Firefox browser; your display may differ according to your choice of browser. We will see later some problems that arise trying to make XML and XHTML files cross-browser compliant.

If you have an invalid XML document, you will get an error message indicating approximately where the problem is. For example, change the line **</name>** to **</Name>** and see what happens. This is the display in Firefox. Other browsers will be similar:

> The XML page cannot be displayed
> Cannot view XML input using XSL style sheet. Please correct the error and then click the Refresh button, or try again later.
>
> ---
>
> **End tag 'Name' does not match the start tag 'name'. Error processing resource 'file:///C:/Users/Osama A. Morad/Desktop/pay_record.xml'. Line 8, Column 9:**
>
> </Name>
> -----^



A similar problem can occur if you use a word processor to compose the file. The word processor will use typographer's inverted commas, 'like this', and in Firefox this will result in a response like:



The display in other browsers will differ and may not be so helpful.

XML namespaces

Before we continue with another example, we should discuss the use of *namespaces* in XML. Since element names in XML are not fixed, a name conflict will occur when two different documents use the same names describing two different types of elements, as in the example that follows.

The following XML document carries information about fruit in a table:

```
<table>
  <tr>
  <td>Pears</td>
  <td>Peaches</td>
  </tr>
</table>
```

This XML document also carries information about a table (using a piece of furniture):

```
<table>
  <name>Walnut Table</name>
  <width>100</width>
  <length>200</length>
</table>
```

If for some reason these two XML documents were combined together, there would be an element name conflict because both documents contain a **<table>** element with different content and definition. We could solve these problems by the use of a prefix on the elements in each of the documents:

This XML document carries fruit information in a table:

```
<h:table>
  <h:tr>
  <h:td>Pears</h:td>
  <h:td>Peaches</h:td>
  </h:tr>
</h:table>
```

This XML document carries information about a piece of furniture (in a table):

```
<f:table>
  <f:name>Walnut Table</f:name>
  <f:width>100</f:width>
  <f:length>200</f:length>
</f:table>
```

Now the element name conflict is gone because the two documents use a different name for their **<table>** element (**<h:table>** and **<f:table>**). By using a prefix, we have created two different types of **<table>** elements. Now an XML namespace is an extension of this concept using a URI (Uniform Resource Identifier).

The fruit table example:

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
  <h:td>Pears</h:td>
  <h:td>Peaches</h:td>
  </h:tr>
</h:table>
```

... and the furniture table example:

```
<f:table xmlns:f="http://www.kit.com/furniture">
  <f:name>Walnut Coffee Table</f:name>
  <f:width>100</f:width>
  <f:length>200</f:length>
</f:table>
```

Instead of using only prefixes, an **xmlns** attribute has been added to the **<table>** tag to give the element prefix a qualified name associated with a namespace.

The **namespace** attribute is placed in the start tag of an element and has the following syntax:

```
xmlns:namespace-prefix="namespace"
```

In the examples above, the namespace itself is defined using an Internet address, such as:

```
xmlns:f="http://www.ohecampus.com/furniture"
```

The W3C namespace specification states that the namespace itself should be a URI. When a namespace is defined in the start tag of an element, all child elements with the same prefix are associated with the same namespace. The address used to identify the namespace is not used by the parser to look up information. The only purpose is to give the namespace a unique name. The Web page specified by the URI does not even need to exist.

However, sometimes organizations may use the namespace as a pointer to a real Web page containing information about the namespace (e.g., http://www.w3.org/TR/html4/), but this is not very helpful as the parser will not look up the webpage. The URI is strictly used to define a prefix.

A URI is a string of characters which identifies an Internet Resource. The most common URI is the Uniform Resource Locator (URL), which identifies an Internet domain address. There is another form of URI, but in our examples, we will only use URLs. Since our furniture example above uses an Internet address to identify its namespace, we can be sure that the namespace is unique.

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax (note that no namespace prefix is used):

```
<element xmlns="namespace">
```

The two examples above now become the following when using default namespaces:

```
<table xmlns="http://www.w3.org/TR/html4/">
  <tr>
  <td>Pears</td>
  <td>Peaches</td>
  </tr>
</table>

<table xmlns="http://www.w3schools.com/furniture">
  <name>Walnut Table</name>
  <width>100</width>
  <length>200</length>
</table>
```

Editing XML documents

Creating XML documents is easy, and can generally be done with something as simple as Windows Notepad or gedit.

There are several XML editors available, some free and some commercial, such as Cooktop (free), Altova XMLSpy (30-day trial available) or Stylus Studio, but these are generally more complicated than needed here.

The best XML editors are, in my opinion, Notepad++ for Windows or Vim, which is included in Linux/Mac operating system (OS); but if you prefer an Integrated Development Environment (IDE), then XML Notepad (free) from Microsoft—simple yet feature rich—is the best choice. Serna Free is a cross-OS WYSIWYG editor for windows, Linux and Mac OS and is an open-source free version of the Serna editor.

I recommend that you avoid using Office products or HTML editors that are not designed for XML development as they may introduce non-printing characters that aren't compatible with the parser.

XHTML versus HTML5

Finally, a brief note about using HTML: The current standard is HTML5; the earlier version comes in two varieties; HTML 4 and XHTML, the latter which is designed to work with XML and differs from HTML 4 in that it is case sensitive; all tags are in lower case and there are a few tags which are handled differently. As we move to HTML5, there is as yet no corresponding version of XHTML although there are several proposals. The most important of these proposals is polyglot HTML, which is designed to be delivered either as HTML5 or XHTML.

In this module, we will use both XHTML and HTML5, but to be compatible with XML, all HTML tags must be in lower case and must be closed.

Note that while the tags for (X)HTML must be lower case, the tags that you will create for XML may be upper or mixed case, but should always be consistent. **<Director>** Smith **</Director>** is valid, but **<Director>** Smith **</director>** is not.

References

Cauldwell, P. et al. (2001) *Professional XML Web services*. Birmingham, UK: Wrox Press Ltd.

Other reading

HTML unleashed. The emergence of XML: introduction
http://webreference.com/dlab/books/html/38-0.html

Getting started with XML: a manual and workshop
http://www.infomotions.com/musings/getting-started/getting-started.html

W3Schools: XML tutorials
http://www.w3schools.com/xml/default.asp

Understanding XML
http://msdn.microsoft.com/en-us/library/aa468558.aspx

Polyglot HTML: HTML-compatible XHTML documents
http://dev.w3.org/html5/html-xhtml-author-guide/

Specifications

XML 1.0
http://www.w3.org/TR/2008/REC-xml-20081126/

XML 1.0: an annotated version
http://www.xml.com/axml/testaxml.htm

XML 1.1 for information only
http://www.w3c.org/TR/2004/REC-xml11-20040204/

XHTML 1.0
http://www.w3.org/TR/xhtml1/

Polyglot HTML
http://dev.w3.org/html5/html-xhtml-author-guide/

Article sites

IBM Developer Works (lots of articles/tutorials)
http://www-106.ibm.com/developerworks/xml/

Microsoft MSXML
http://msdn.microsoft.com/en-gb/data/bb190600.aspx

Introduction to tree-based XML programming guide for Cocoa Apple
https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/NSXML_Concepts/NSXML.html

Tools

XML Notepad
http://www.microsoft.com/en-us/download/details.aspx?id=7973

Notepad++
http://notepad-plus-plus.org/

Serna Free
http://sourceforge.net/projects/sernafree.mirror/

Other online resources

XML description
http://en.wikipedia.org/wiki/XML

XHTML description
https://en.wikipedia.org/wiki/XHTML

Polyglot markup description
https://en.wikipedia.org/wiki/Polyglot_markup

Web service protocols
http://en.wikipedia.org/wiki/List_of_web_service_protocols

Description of Web services architecture
http://en.wikipedia.org/wiki/Web_service
http://www.w3.org/TR/ws-arch/

XML home page
http://www.w3.org/XML/

XML tutorial
http://www.w3schools.com/xml/default.asp

Web services home
http://www.w3.org/2002/ws/
Web services tutorial
http://www.w3schools.com/webservices/default.asp

Note: Wikipedia is not an acceptable academic reference, but it is a good place to start looking. Then you can search further and find more reliable sources.