Murdoch
UNIVERSITY

XML + DTD + Namespaces

Lecture 5 (A)

Assignment One

- Assignment one question sheet is available on LMS
- Submission deadline is 5pm, Friday, Week 7
- You have been advised to have finished Labs 3 and Lab 4 before attempting a final prototype solution for the assignment
  - If you have not started/finished labs 3 and lab 4, do so as soon as possible, and then start/continue working on the assignment

Assignment One

- You definitely should read the question a few times after its release and start planning and developing initial and intermediate prototypes
- All students should submit their assignment on LMS according to the instructions in the assignment question sheet AND have their working application residing under their home directory on ceto.murdoch.edu.au
- Late submission penalties will apply - refer to the assignment question sheet

Murdoch UNIVERSITY

What we have done so far

- We have covered:
  - Installation and configuration of a fully featured Web server (Apache)
  - Overview of Web Development
  - JavaScript functions and objects
  - Fundamentals of Node.js development environment
  - Web Client and Web Server Application Development using Node.js

What do we expect you to achieve?

- Gain an understanding of the following topics:
  - Web Computing and Programming
  - Development of Web Server and Client applications
  - Understanding of XML technologies
  - Understanding JSON technologies
  - Understanding jQuery basics (and tables / graphs)
  - Understanding of Web Services

What do we expect you to achieve?

- Demonstrate the ability to develop programming skills for the development of Web applications
- Demonstrate the ability to develop Web Client and Server applications that can deal with XML and JSON
- An appreciation of alternate development approaches

Introduction to XML

Lecture Objectives

- Understand why there is a need for XML
- A first look at:
  - The components of XML
  - How XML relates to other technologies

Mark-up Languages and XML

- The use of **mark-up languages** is prevalent among the computing world
  - HTML (HyperText Markup Language) is the most visible markup language
- To understand the reason for XML (eXtensible Markup Language), it is important that you have an understanding of the concept of mark-up languages
  - XML is basically about creating new mark-up languages

Style, Structure and Content

- Historically, when all documents were only in printed form, a major design consideration for a document was its **style** (or **layout** and **presentation**)
- The structure of the document was left to the reader of the printed document to work out based on the presented style

An Example Page

- Looking at this page you can decipher that:
  - The words in large blue font at the top constitute the *title*
  - The words on the right of each large dot constitute a *main point*
  - The words on the right of a smaller dot constitute *sub-points* to main points
  - The number at the top right is the *page number*

Example Page Using RTF Style Instructions


```
\par\font=44\color=blue\bold
An Example Page
\no-bold\newline\par\font=32
\bullet
Looking at this page you can decipher that:
\newline\indent\font=28
\bullet
The words in large font at the top constitute the
\italic
title
\no-italic\newline\indent\font=28
The words on the right of each large dot constitute a
\italic
main point
… etc ...
```

Murdoch UNIVERSITY

Can a Program do the Same?

- Probably, but not efficiently!!
- These days we have moved far beyond that basic approach when representing information
- The key to having vast amounts of information passed around in a network environment is to have the information processed and dealt with by **automated programs** (i.e., applications)
- If a program receives a document containing only style instructions (like the previous PowerPoint or RTF representations), how can it work out the structure of the information??

Mark-Up Languages

- So, the point of mark-up languages is to:
  - Provide information about the **structure and content in a document**, so that they can be efficiently processed by **programs automatically**
  - The most important task is to **separate** the structure and content components of the documents from the style components

Mark-Up Languages for Different Purposes

- But different problem domains will have different ways of describing their data
  - It is impossible to come up with just one set of mark-up tags, and expect them to work for all problem areas
- So mark-up **meta**-languages like XML **do not** define a specific set of tags for all purposes
- Instead, they specify a **standard way** for **defining** the tags of **completely new mark-up languages**

Why Mark-up Meta-Languages?

- The key is **standardization**, by having these meta-languages provide:
  - A **standardized method** and a **consistent set of tools** to create **new mark-up languages** appropriate for particular problem domains, and to make these new languages public
  - The ability to **parse** and **process the documents** of these new mark-up languages in **a consistent way** - this is very important for application development in the problem domain
- If the **meta-language** is defined well, it provides **an easy way** to create new mark-up languages

The Simplicity of HTML

- In the beginning, most development in document formatting concentrated on HTML, since HTML:
  - Was simple and easy to learn
  - Provided mechanisms to do some interesting things (display formatting, links, embedded graphics and other multimedia, etc.)
  - Was forgiving on errors (like forgetting a closing tag)
- However, with the need for more complex applications, developers began using HTML beyond what it was originally defined to do

Beyond HTML

- The main problem with HTML was that it was never designed to be **extensible** (that is, capable of being extended as necessary)
- As a software developer, if you want to exchange documents on the web, but find that HTML doesn't do what you need it to do, you cannot easily create and use a new language that extends HTML, and share this new language with others

Enter XML

- Standards for XML began to appear under the umbrella of the World-Wide-Web Consortium (W3C) in 1996 to address some of the above problems
- XML is actually a subset of the Standard Generalized Mark-up Language (SGML), which began as the Generalized Mark-up Language (GML) in the 1960's

So What IS XML?

- XML's foundation is the XML **recommendations**, which define what an XML document *must and must not have*
- This forms the basis for creating a new mark-up language
- New mark-up languages created following the XML specifications are called **Applications of XML**

An Example Page

- Looking at this page you can decipher that:
  - The words in large blue font at the top constitute the *title*
  - The words on the right of each large dot constitute a *main point*
  - The words on the right of a smaller dot constitute *sub-points* to main points
  - The number at the top right is the *page number*

Our Previous Example

- We can mark-up the document like this:

```
<page>
    <title>An Example Page</title>
    <main_point>
        <main_point_text>
            Looking at this page you can decipher that:
        </main_point_text>
        <sub_point>
            The words in large font at the top constitute the title
        </sub_point>
        <sub_point>
            The words on the right of each large dot constitute a main point
        </sub_point>
        <sub_point>
            The words on the right of a smaller dot constitute sub-points
        </sub_point>
        <sub_point>The number at the top right is the page number</sub_point>
    </main_point>
    <page_number>22</page_number>
</page>
```

Our Previous Examples

- Compared to our two previous documents below, writing a **program** to identify and use the different components of the page will be so much more efficient using our own mark-up

An Example

- For example, looking at this page, you as a reader can decipher that:
  - The words with the big fonts at the top ("An Example") is a *title*.
  - The words on the right of each large dot is a *main point*.
  - The words on the right of each smaller dot is a *sub-point* to the main points.
  - The characters on the top right is a *page number*.
  - etc.

```
\par\font=28\bold
An example
\no-bold\newline\par\font=24
\bullet
For example, looking at this page, you as a reader can decipher that:
\newline\indent
\bullet
The words with the big fonts at the top ("An Example") is a
\italic
title.
\no-italic\newline\indent
The words on the right of each large dot is a
\italic
main point.
...
```

A Sensible XML Document Example


```
<course>
     <name>Bachelor of Science – Mobile and Web Application
           Development
     </name>
     <duration>3 years</duration>
     <unit>
          <title>ICT375 Advanced Web Programming</title>
          <lecturer>
              <surname language="English">Xie</surname>
              <othernames language="English">Hong</othernames>
              <email>H.Xie@murdoch.edu.au</email>
          </lecturer>
     </unit>
     <unit>
          <title>ICT283 Data Structures and Abstraction</title>
          <lecturer>
              <surname>Rai</surname>
              <othernames>Shri</othernames>
              <email>s.rai@murdoch.edu.au</email>
          </lecturer>
     </unit>
</course>
```

Murdoch UNIVERSITY

The Document Type Definition

- A **Document Type Definition (DTD)** defines the **structure** of an XML document
  - It can be put at the beginning of an XML document, or can be linked to the document via an external file

```
<!DOCTYPE course [
      <!ELEMENT course (name, duration, unit+)>
      <!ELEMENT name (#PCDATA)>
      <!ELEMENT duration (#PCDATA)>
      <!ELEMENT unit (title, lecturer*, tutor*)>
      <!ELEMENT title (#PCDATA)>
      <!ELEMENT lecturer (surname, othernames?, email*)>
      <!ELEMENT tutor (surname, othernames?, email*)>
      <!ELEMENT surname (#PCDATA) >
      <!ATTLIST surname language CDATA "English">
      <!ELEMENT othernames (#PCDATA) >
      <!ATTLIST othernames language CDATA "English">
      <!ELEMENT email (#PCDATA)>
]>
```

Is that all there is to XML?

- Also associated with the core concepts of XML is a whole set of other technologies
- Some examples include:
  - Alternate ways (to the DTD) to define document types (XML Schema)
  - Programming language APIs which allow you to easily write software to parse and process XML documents
  - Specifications on how to represent an XML document at a program level (eg: DOM, SAX)

Is that all there is to XML?

- Specifications on how to transform an XML document, to another document type, or to another XML document of the same type (using XSLT)
- Specifications on how to define special elements in a document (eg: isolating XML document components using XPath)
- Special browsers and general web browsers capable of displaying XML documents
  - How do general web browsers know what to display when it doesn't know what the XML document tags "mean"?
  - This is a question we will explore as we study the various XML technologies

Popular XML Applications

- Math Markup Language
  - MathML is a low-level specification for describing mathematics as a basis for machine to machine communication
  - MathML also provides a foundation for the inclusion of mathematical expressions in Web pages
- Synchronized Multimedia Integration Language
  - SMIL defines mark-up for layout, timing, visual transitions, animations, media embedding, etc.
  - SMIL allows the presentation of media items (text, images, video, audio), as well as links to other SMIL presentations, and files from multiple web servers

Popular XML Applications

- Chemical Markup Language
  - CML is an approach to managing molecular information using XML
  - CML uses XML's portability to help developers and chemists design inter-operable documents
- electronic business XML (ebXML)
  - ebXML standardizes XML document exchange between businesses and organizations
- XHTML
  - eXtensible Hypertext Markup Language

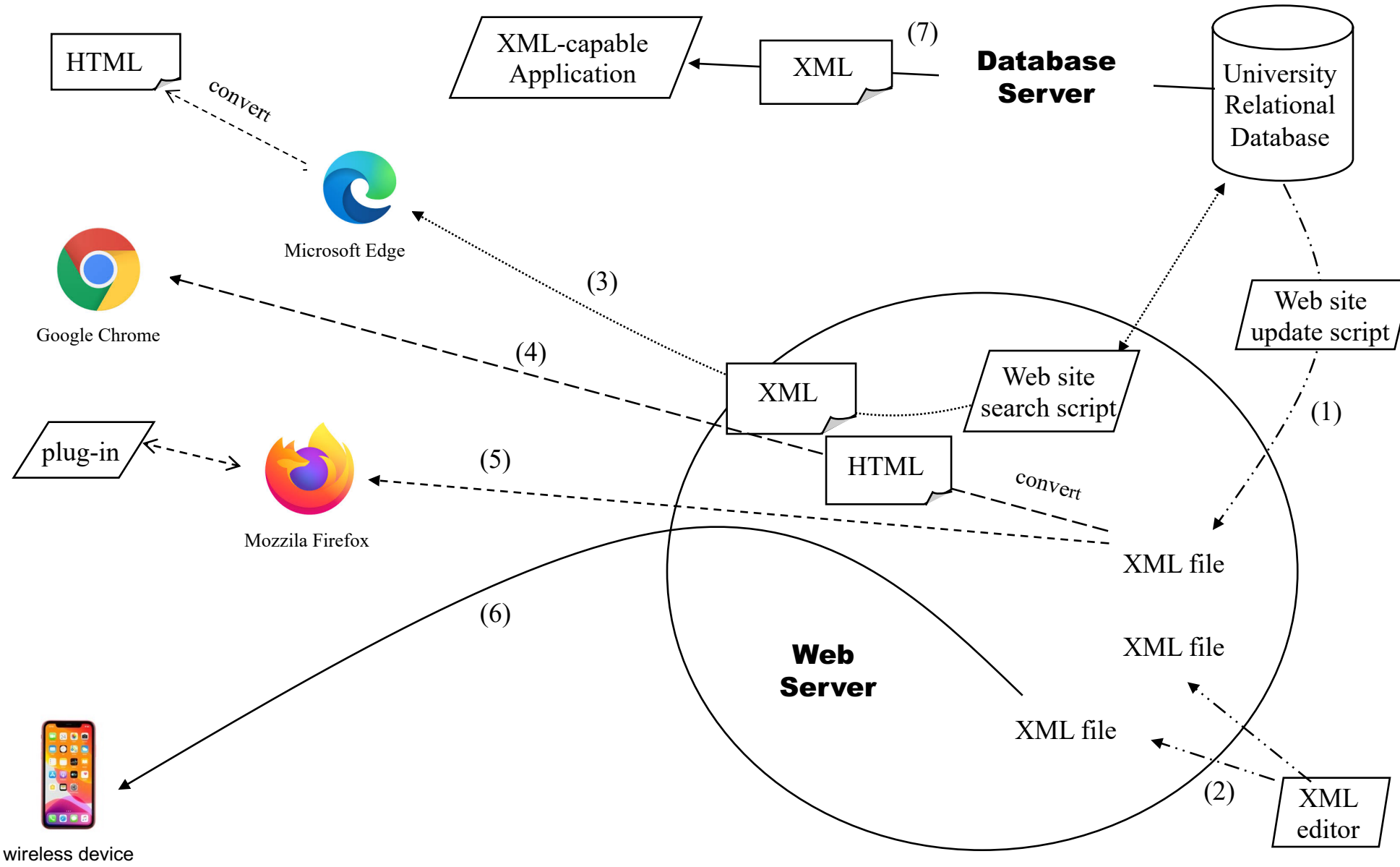
The Design Goals of XML

- As proposed by W3C:
  1. XML shall be straight-forwardly usable over the Internet
  2. XML shall support a wide variety of applications
  3. XML shall be compatible with SGML
  4. It shall be easy to write programs which process XML documents

The Design Goals of XML

5. The number of *optional* features in XML is to be kept to the absolute minimum, ideally zero
6. XML documents should be human-legible and reasonably clear
7. The XML design should be prepared quickly
8. The design of XML shall be formal and concise
9. XML documents shall be easy to create

Example Usages of Our `course` XML



Example Usages of Our `course` XML From the Previous Diagram

1. A daily update script extracts course information from a relational database and generates XML files to put on a web site
2. A data-entry person uses an XML editor to create new XML files with course information
3. A user surfing the web site uses the search facility to find some course information. The search script queries the database, gets back the results, and sends the results back to the browser in XML. The browser converts the XML to HTML and displays HTML as normal
4. A user using a web browser accesses some course information. The web server converts the XML document to HTML and sends HTML back to the browser. Browser displays HTML as normal

Example Usages of Our `course` XML From the Previous Diagram

5. A user using a web browser with a special course-info plug-in accesses some course information. The web server sends the XML document directly to the browser. The browser uses the plug-in to display the course information
6. A user on a wireless device accesses some course information. The XML document is sent to a mobile app installed on the user's wireless device, which displays the course information.
7. An XML-enabled application connects to the database and retrieves some course information. The database server sends results back in XML

Endless Possibilities

- The previous diagram demonstrates only a very small fraction of what is possible with XML technologies
- The possibilities for which software does what, when, where and how, are endless

XML Still Valuable

- Though JSON is replacing XML as a preference with Web developers for Web site development (because of its ease of use for data storage and transfer), XML is still the preference with many corporations for storing vast amounts of data
- In order to utilize the vast amounts of legacy data still available via the Internet, being able to understand and work with XML technologies is still a very valuable asset for those seeking to work in this industry

For Further Information

- Follow W3C's XML activities:
  - http://www.w3.org/
  - http://www.w3.org/XML/

Murdoch
UNIVERSITY

The XML Document, DTD, and Namespaces

Lecture 5 (B)

The XML Document

Lecture Objectives

- Understand the role of the XML document format in XML technologies
- Know the format and syntax of an XML document, as specified by W3C's XML 1.0 Recommendation
- In regard to what we are doing in the unit:
  - XML is an important set of Internet technologies for use in different solutions in different areas
  - **The XML document format is the first and most basic step in understanding how all the different XML technologies work**

Lecture Outline

- The importance of the XML document format specifications
- Definition of a well-formed XML document
- Components of a well-formed XML document

The XML Document Specification

- In the last set of lecture slides, we discussed two key factors leading to the success of XML as a set of technologies:
  1. Standardization
  2. Ease of creating new mark-up languages
- The quality of the specifications of the basic XML document format are based on these two factors

Reference

- The current base XML 1.0 Recommendation (5th Edition) was released on 26th November 2008
  - http://www.w3.org/TR/REC-xml
  - Note: the above specification was modified in place to replace broken links to RFC4646 and RFC4647 (investigate to find out more)

W3C Recommendation

Extensible Markup Language (XML) 1.0 (Fifth Edition)

W3C Recommendation 26 November 2008

This version:
http://www.w3.org/TR/2008/REC-xml-20081126/
Latest version:
http://www.w3.org/TR/xml/
Previous versions:
http://www.w3.org/TR/2008/PER-xml-20080205/
http://www.w3.org/TR/2006/REC-xml-20060816/
Editors:
Tim Bray, Textuality and Netscape <tbray@textuality.com>
Jean Paoli, Microsoft <jeanpa@microsoft.com>
C. M. Sperberg-McQueen, W3C <cmsmcq@w3.org>
Eve Maler, Sun Microsystems, Inc. <eve.maler@east.sun.com>
François Yergeau

Please refer to the **errata** for this document, which may include some normative corrections.

The previous errata for this document, are also available.

See also **translations**.

This document is also available in these non-normative formats: XML and XHTML with color-coded revision indicators.

The XML Document Specification

- The W3C's XML 1.0 Recommendation specifies two sets of constraints for an XML document:
  - A **well-formed** document: The XML document must conform to the basic syntax rules
  - A **valid** document: Having a Document Type Definition (DTD) to specify allowed components in the XML document (eg: tags, structure)
- For now, we will consider what makes an XML document **well-formed**

The Importance of the XML Document Specification

- Having well-defined documents will make it:
  - Easy for document publishers and information content creators to create new documents
  - Easy for software to parse and process the documents
  - Easy for people to read and understand the documents
- These reflect the core design goals of XML

The Design Goals of XML - Recap

- As proposed by W3C:
  1. XML shall be straight-forwardly usable over the Internet
  2. XML shall support a wide variety of applications
  3. XML shall be compatible with SGML
  4. It shall be easy to write programs which process XML documents

The Design Goals of XML - Recap

5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero
6. XML documents should be human-legible and reasonably clear
7. The XML design should be prepared quickly
8. The design of XML shall be formal and concise
9. XML documents shall be easy to create

A Well-Formed XML Document

- **ALL** XML documents **MUST** be well-formed
  - That is, they **MUST conform to all syntax definitions in the XML 1.0 Recommendation**
- Unlike HTML, XML software are not allowed to "correct" errors
  - That is, **non-well-formed documents will always cause unrecoverable errors in all XML software**

A Well-Formed XML Document

- A well-formed XML document consists of three parts:
  - Prolog (optional)
  - A root element
  - Miscellaneous parts (optional)
    - Can exist within the Prolog and the root element parts

The Prolog

- The prolog exists at the beginning of an XML document. It can consist of:
  - An XML declaration, followed by
  - Miscellaneous parts, followed by
  - A Document Type Definition (DTD) - more later
- Although formally the prolog **can** be empty, the W3C specifications recommend that no document leaves out the XML declaration part
  - Most XML documents on the Internet do have an XML declaration

The XML Declaration

- The declaration must have at least the "**xml**" keyword and the "**version**" attribute
- These keywords are tagged with the starting and ending characters <?xml … ?>
- Some example XML declarations:

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

<?xml version="1.0" standalone="yes"?>

<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
```

Root Element

- The **root element:** all of the "actual" document data resides within this element
- There can be **one and only one** root element, which can have child elements
- All **child elements** must be **properly nested** within the root element, and each other
- The effect of these conditions means that the elements form a **hierarchical** or **tree-like** structure
  - The root element (obviously) is at the root of the tree

Miscellaneous Parts

- The "Miscellaneous parts" of an XML document can consist of:
  - Comments (the same syntax as HTML comments)
    Eg:

```
<!-- Start of the main tag -->
```

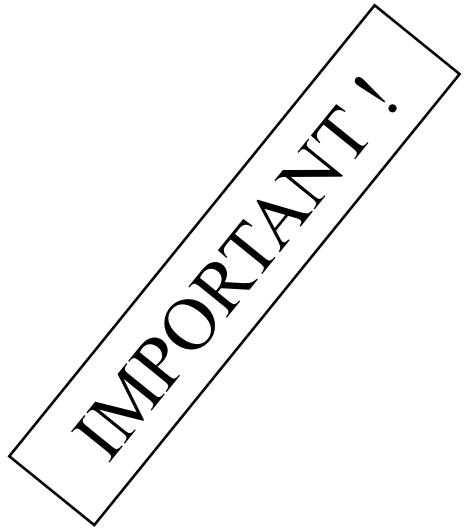
  - Processing Instructions (within <?xml … ?>)
    Eg:

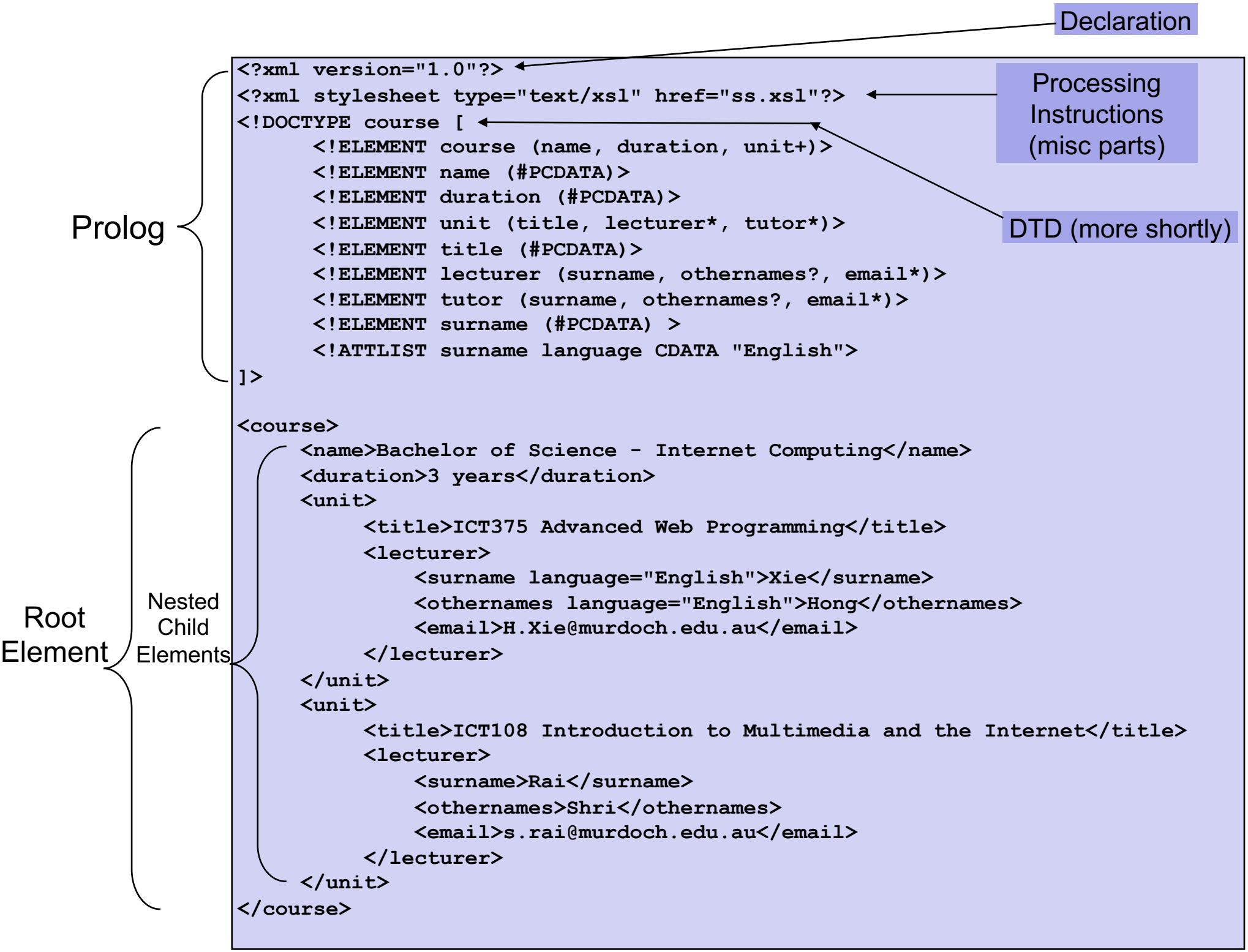
```
<?xml stylesheet type="text/css"
                  href="mycss.css"?>
```

  - White Spaces

In Summary

- A well-formed XML document consists of (in order):
  - (Optional) Prolog with
    - A <?xml …?> declaration
    - Miscellaneous parts
    - Document Type Definition
  - A required root element
    - Within which all other elements exists
  - (Optional) Miscellaneous parts




Declaration
Processing Instructions (misc parts)
DTD (more shortly)
Prolog
Root Element
Nested Child Elements
<?xml version="1.0"?>
<?xml stylesheet type="text/xsl" href="ss.xsl"?>
<!DOCTYPE course [
<!ELEMENT course (name, duration, unit+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT duration (#PCDATA)>
<!ELEMENT unit (title, lecturer*, tutor*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT lecturer (surname, othernames?, email*)>
<!ELEMENT tutor (surname, othernames?, email*)>
<!ELEMENT surname (#PCDATA) >
<!ATTLIST surname language CDATA "English">
]>
<course>
<name>Bachelor of Science - Internet Computing</name>
<duration>3 years</duration>
<unit>
<title>ICT375 Advanced Web Programming</title>
<lecturer>
<surname language="English">Xie</surname>
<othernames language="English">Hong</othernames>
<email>H.Xie@murdoch.edu.au</email>
</lecturer>
</unit>
<unit>
<title>ICT108 Introduction to Multimedia and the Internet</title>
<lecturer>
<surname>Rai</surname>
<othernames>Shri</othernames>
<email>s.rai@murdoch.edu.au</email>
</lecturer>
</unit>
</course>

Elements in XML

- The contents of an XML document (the information the author wants to convey) are broken up into units called **elements**
- Different types of elements are given different names, and are tagged with a **start-tag**, an **end-tag**, and the element name
  - For example, in the following element the content "12345678" of type "StudentID" is marked-up with the tag `<StudentID>`

```
<StudentID>12345678</StudentID>
```

Elements in XML

- **All** basic information text must be tagged in an XML file
- All elements must exist **properly** nested within the root element (and each other)
  - The left example below demonstrates overlap, and is therefore incorrect in syntax:

INCORRECT

CORRECT

Empty Elements

- Some elements do not need closing tags
  - These are called **empty** elements
  - You will have come across the concept of empty elements in HTML, with tags like `<hr>`, `<br>`, `<img>`, etc.
- In XML, tags of empty elements must end with a forward slash - “/”
  - Thus in XHTML, which is the XML compliant version of HTML, we have `<hr />` and `<br />` tags

Attributes of Elements

- Elements may also contain attributes
- The attribute **names** and **values** are defined within the element's start-tag. Eg:

```
<Student ID="12345678" status="enrolled"
                       workrate="high" />
```

- All attribute values must be enclosed in quotes
- As the `Student` element above consists of only attributes, it must be treated as an empty element, thus closed with a forward slash

Attributes vs Child Elements

- For any element, you can define information about the element by using attributes, or you can define a new child element
- For example, the following elements contain the same information:

Child elements:

```
<lecturer>
   <surname>Xie</surname>
   <othernames>Hong</othernames>
   <email>H.Xie@murdoch.edu.au</email>
</lecturer>
```

Attributes:

```
<lecturer surname="Xie" othernames="Hong"
          email="H.Xie@murdoch.edu.au" />
```

Murdoch UNIVERSITY

Attributes vs Child Elements

- Sometimes there are obvious technical reasons for using one versus the other
  - Eg: if you want to use a default value – this is easier with an attribute rather than with a child element
- In many cases the decision will be a judgment call
- If using an attribute, there should at least be some differences between the nature of the information in the attributes; otherwise define a new child element

Pre-Defined Entity References

- Since some characters like "<" and ">" have special use in XML, you cannot use them in some places. Eg:

```
<number attribute=">5"> 4 </number>
```
- Wrong syntax!

```
<number attribute="&gt;5"> 4 </number>
```
- Correct!

- There are 5 pre-defined entity references to alleviate these types of situations:

&lt;	The < character
&gt;	The > character
&amp;	The & character
&apos;	The ‘ character
&quot;	The " character

Defining New Entities

- Besides the pre-defined entity references, you may also define new entities in the XML document
- This can be useful for:
  - Reference to commonly used names
  - Multi-language support
  - Managing binary files
  - And more ...

CDATA Sections

- Since XML is very sensitive to characters like "<" and "&", you can define CDATA sections in an XML document which are **not** supposed to be parsed
  - CDATA stands for **character data**
- Eg:

```
<![CDATA[
  I'm free to use any of my own special
  characters like <&*@!]% in here!!!
]]>
```

Murdoch UNIVERSITY

In Summary

- Things to watch out for when constructing a well-formed XML document:
  - You should have an **XML declaration at the beginning**
  - You must include **at least the root element**
  - All other elements must be contained within the **root element**
  - You must correctly nest all child elements - **no overlaps**

In Summary

- Things to watch out for when constructing a well-formed XML document (cont.):
  - You must include **both start and end tags for non-empty elements**
  - You must use **"/" to close empty element tags**
  - You must use unique **attribute names**
  - You must use **quotes** for attribute values
  - You must use the **pre-defined entity references** instead of the "special" original characters

The Document Type Definition

Lecture Objectives

- Understand what a Document Type Definition (DTD) is, and its role in an XML document
- Understand the role and importance of the DTD in *extending* XML
- Know the format and syntax of DTDs as specified by W3C's XML 1.0 Recommendation

Lecture Objectives

- In regard to what we are doing in this unit:
  - XML is an important set of Internet technologies for use in different solutions in different areas
  - Having the ability to create new mark-up languages is a key factor in XML technologies
  - **The DTD is technically one of the main methods used to define the syntax of an XML document, and therefore it is a mechanism used to specify a new mark-up language**

Lecture Outline

- What is a Document Type Definition?
- What constitutes a **valid** XML document?
- The syntax of DTDs
- DTDs and Validating Parsers

The Document Type Definition

- We discussed how W3C's XML 1.0 Recommendation specifies the syntax of a well-formed XML document
- Part of this syntax allows a **Document Type Definition** to be included in the Prolog

A Valid XML Document

- The Document Type Definition defines the structure for the tags in an XML document:
  - What tags are allowed in the document
  - Which tags contain which other tags
  - Where the basic text data are located, etc…
- The XML 1.0 Recommendation defines a **valid** document to be one which:
  - Has a Document Type Definition, **AND**
  - The XML document syntax conforms to the rules of its own Document Type Definition

Creating New Mark-Up Languages

- Since DTDs define the tags and how they relate to each other, it is **this component of XML that defines a new mark-up language**
- It is up to organizations, consortiums, groups, partners, etc. to
  - Decide on the mark-ups that are most appropriate for their area
  - Define the structure of the tags in the DTD
- … And then all their shared documents can make use of the same DTD

Declaration for a DTD

- A DTD can be defined in the XML document prolog or in a separate file
- The DTD section of an XML document is called the Document Type Definition **Declaration**
  - The DTD Declaration is in the form of a `<!DOCTYPE>` tag
  - Internal DTDs are declared using the format

```
<!DOCTYPE rootname [ DTD ]>
```

Prolog

DTD

XML

```
<?xml version="1.0"?>
<?xml stylesheet type="text/xsl" href="ss.xsl"?>
<!DOCTYPE course [
      <!ELEMENT course (name, duration, unit+)>
      <!ELEMENT name (#PCDATA)>
      <!ELEMENT duration (#PCDATA)>
      <!ELEMENT unit (title, lecturer*, tutor*)>
      <!ELEMENT title (#PCDATA)>
      <!ELEMENT lecturer (surname, othernames?, email*)>
      <!ELEMENT tutor (surname, othernames?, email*)>
      <!ELEMENT surname (#PCDATA) >
      <!ATTLIST surname language CDATA "English">
      <!ELEMENT othernames (#PCDATA) >
      <!ATTLIST othernames language CDATA "English">
      <!ELEMENT email (#PCDATA) >
]>
<course>
     <name>Bachelor of Science - Internet Computing</name>
     <duration>3 years</duration>
     <unit>
          <title>ICT375 Advanced Web Programming</title>
          <lecturer>
              <surname language="English">Xie</surname>
              <othernames language="English">Hong</othernames>
              <email>H.Xie@murdoch.edu.au</email>
          </lecturer>
     </unit>
     ...
</course>
```

Defining Elements in a DTD

- The syntax to declare elements in a DTD uses a tag of the form:

```
<!ELEMENT name content_model>
```

  - Where:

    `name` is the name of the element or tag

    `content_model` is one of

    - `EMPTY` (meaning an empty element)
    - `ANY` (meaning the parser should not check content)
    - child elements, or
    - mixed (meaning `#PCDATA` or child elements)

Defining Elements in a DTD

- Some examples:

```
<!ELEMENT book_header (title, author) >

<!ELEMENT staff (name, duty) >

<!ELEMENT document EMPTY >

<!ELEMENT undefined ANY >

<!ELEMENT info (#PCDATA) >

<!ELEMENT info ((field, value) | #PCDATA) >
```

#PCDATA

- To define a section that contains the raw text data of an XML document (i.e. the document content), we use the keyword `#PCDATA`
  - `#PCDATA` stands for **parsed character data**
- The `#PCDATA` **is parsed**, so cannot contain special characters like "<"
  - This is different from the `CDATA` section mentioned earlier in the lecture, which is **not parsed**

Defining Children

- To specify that one element contains another element, we include the name of the element in the `content_model`
- Eg:

```
<!ELEMENT course (name, duration, unit+) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT duration (#PCDATA) >
<!ELEMENT unit (title, lecturer*, tutor*) >
```

  - Here the `course` and `unit` elements each have a `content_model` containing the names of other elements
  - The `content_model` is listed in parenthesis

Having Multiple Children

- To specify that one element may contain more than one child element of a specific type, we use special symbols:

x+	where + means 1 or more occurrences of element x
x*	where * means 0 or more occurrences of element x
x?	where + means 0 or 1 occurrence of element x
x, y	which means element x followed by element y (i.e. a **sequence**)
x | y	which means element x or element y, but not both

Having Multiple Children

- Some examples:

```
<!ELEMENT book_header (title?, author?, info*)>

<!ELEMENT staff (name, duty+) >

<!ELEMENT info ((field?, value) | #PCDATA) >
```

Sub-sequences with Parentheses

- As we have just seen in our last example, you can include sub-sequences of child elements by using parentheses in the child sequence

```
<!ELEMENT info ((field?, value) | #PCDATA) >
```

DTD Comments

- Comments can be put anywhere in the DTD section, using the same comment syntax as defined by a well-formed XML document
- Eg:

```
<!DOCTYPE course [
   ...
   <!-- This is the name of the course -->
   <!ELEMENT name (#PCDATA) >
   ...
]>
```

External DTDs

- DTDs not defined in the XML document itself (i.e. defined in another file), can be referred to by using a different declaration. Eg:

```
<!DOCTYPE course SYSTEM "course.dtd">
<!DOCTYPE course PUBLIC
      "-//uni consortium//Custom Course v1.0//EN"
      "http://university.edu.au/course.dtd">
```

  - SYSTEM means the file is located in the local filesystem of the computer; you can specify a path to the file
  - PUBLIC means the resource is located at the given link
  - Note in either case, the XML root element must still be defined

Prolog

```
<?xml version="1.0"?>
<?xml stylesheet type="text/xsl" href="ss.xsl"?>
<!DOCTYPE course SYSTEM "course.dtd">
```

DTD

```
<course>
     <name>Bachelor of Science - Internet Computing</name>
     <duration>3 years</duration>
     <unit>
          <title>ICT375 Advanced Web Programming</title>
          <lecturer>
              <surname language="English">Xie</surname>
              <othernames language="English">Hong</othernames>
              <email>H.Xie@murdoch.edu.au</email>
          </lecturer>
     </unit>
     ...
</course>
```

XML document

```
<!ELEMENT course (name, duration, unit+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT duration (#PCDATA)>
<!ELEMENT unit (title, lecturer*, tutor*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT lecturer (surname, othernames?, email*)>
<!ELEMENT tutor (surname, othernames?, email*)>
<!ELEMENT surname (#PCDATA) >
<!ATTLIST surname language CDATA "English">
<!ELEMENT othernames (#PCDATA) >
<!ATTLIST othernames language CDATA "English">
<!ELEMENT email (#PCDATA) >
```

course.dtd

Multiple DTDs

- You can define an XML document using elements from **multiple DTDs**
- This is done using the concept of **Namespace**s which we will look at shortly

Validating Parsers

- All XML parsers require that the documents be well-formed, but some publicly available parsers do not check for validity
- XML parsers that validate the documents they parse against the document's DTD are called **validating parsers**

Namespaces

Learning Objectives

- Understand the concept of Namespaces in XML documents
- We are studying how to use XML as an important set of Internet technologies to use as solutions in different areas
- **Namespaces** are an integral part of XML
  - It allows us to use the same name for elements in different contexts
  - It allows us to clearly identify what names we are referring to

Name Conflicts in XML

- Since any element name can be used in XML element tags, very often a name conflict will occur when two (or more) different documents attempt to use the same name (defined in their DTDs) to describe two (or more) different types of elements
- In an environment where there may be a lot of XML documents being processed automatically by software, this can create many problems and lead to disastrous results

Example Name Conflict

- We can not put these two sets of tags together in a single document because the `<table>` element will cause a name conflict

HTML

```
<table>
   <tr>
      <td>Some data</td>
   </tr>
</table>
```

Furniture Data (XML)

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

Resolving Name Conflicts Using Prefixes

- One possible way to resolve name conflicts is by using prefixes
- In our example, we can define the two different uses of the `<table>` tag with their own individual prefix followed by a colon (:)

```
<html:table>
<furniture:table>
```

- Note that the prefix must then be applied to **every start-tag and end-tag**

Resolving Name Conflicts Using Prefixes

HTML

```
<html:table>
   <html:tr>
      <html:td>Some Data</html:td>
   </html:tr>
</html:table>
```

Furniture Data (XML)

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

XML Namespaces

- XML namespaces provide a simple method for qualifying element and attribute names, used in eXtensible Mark-up Language (XML) documents, by associating them with namespaces identified by Uniform Resource Identifier (URI) references

XML Namespace Declarations


```
<html:table
   xmlns:html="http://www.w3.org/TR/html5/">
   <html:tr>
      <html:td>Entry</html:td>
   </html:tr>
</html:table>
```

A unique URI

```
<furniture:table
   xmlns:furniture="http://www.table.org/">
   <furniture:name>Coffee Table</furniture:name>
   <furniture:length>120</furniture:length>
   <furniture:width>80</furniture:width>
</furniture:table>
```

Namespaces and the **xmlns** Attribute

- Instead of using only prefixes, our two examples also include the **xmlns attribute**

```
xmlns:html=" ..Unique URI.. "
xmlns:furniture =" ..Unique URI.. "
```

  - These have been added to their respective <table> tags to give the element prefixes a **qualified name (QName)**, which associates the tags with a namespace
  - Note that **xmlns** means **XML Namespace**

Namespaces and the **xmlns** Attribute

- Although the namespace identifier is usually a valid Uniform Resource Locater (URL), it does not necessarily have to be a valid web address
  - It can be any arbitrary string to uniquely identify the particular **namespace**
  - Hence, namespaces are identified by their Uniform Resource Identifier (**URI)**

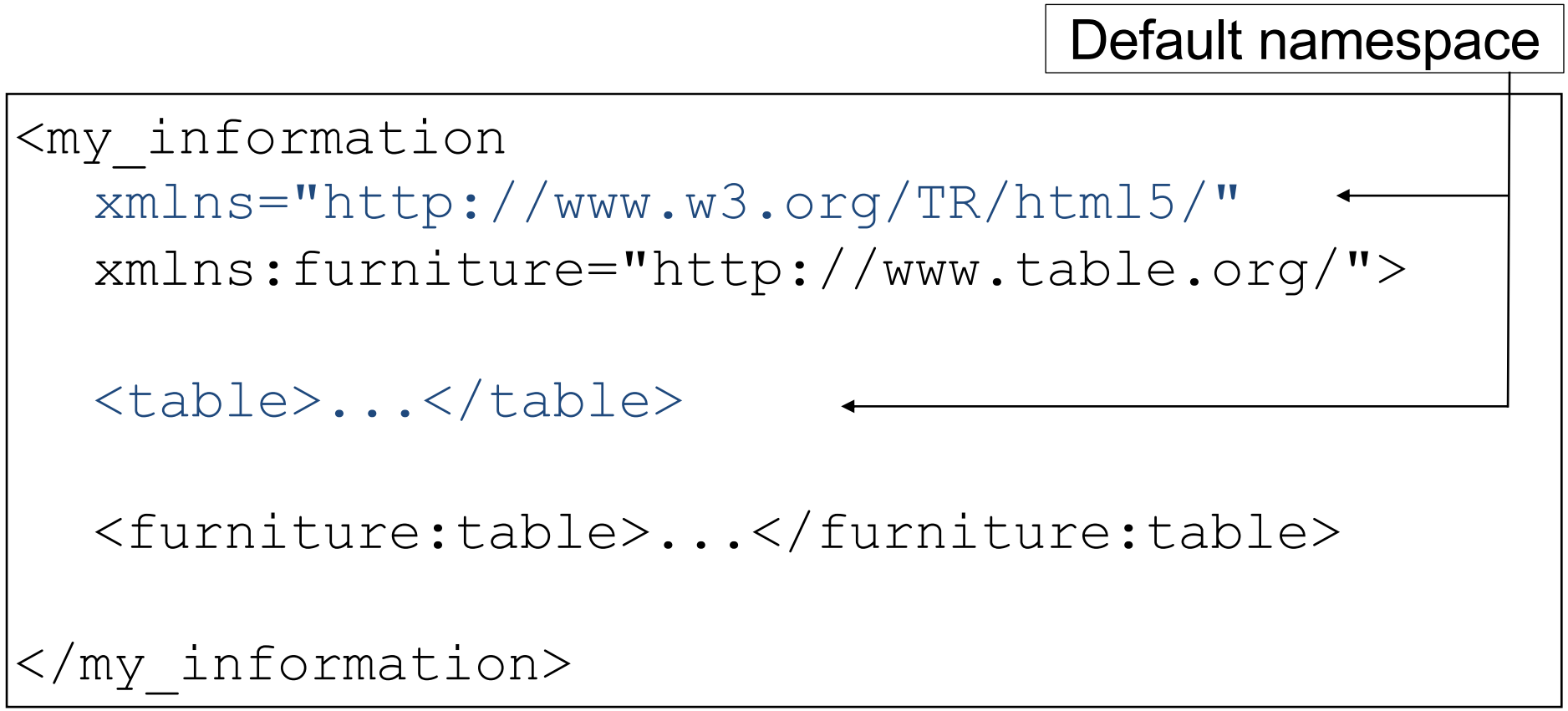
Mixing Namespaces

- We can mix namespaces by including multiple identifiers in the start tag

```
<my_information
   xmlns:html="http://www.w3.org/TR/html5/"
   xmlns:furniture="http://www.table.org/" >
   <html:table>
       ...
   </html:table>
   <furniture:table>
       ...
   </furniture:table>
</my_information>
```

Default Namespaces

- We can specify a default namespace, which will apply to all tags without prefixes



Default Namespaces for Attributes

- Unlike elements, the default namespace for attributes is the namespace of the containing element

```
<my_information
   xmlns="http://www.w3.org/TR/html5/"
   xmlns:furniture="http://www.table.org/" >
   <table> ... </table>
   <furniture:table color="red">
         . . .
   </furniture:table>
</my_information>
```

attribute name and value from the furniture namespace, only apply to this namespace

Some Well-Known Namespaces

- HTML5
  - xmlns:html="http://www.w3.org/TR/html5/"
- Schema
  - xmlns:xsd="http://www.w3.org/2000/XMLSchema"
  - xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- XSLT
  - xmlns:xsl=http://www.w3.org/1999/XSL/Transform
- Further Reading:
  - http://www.w3.org/TR/REC-xml-names/

W3C Recommendation

Namespaces in XML 1.0 (Third Edition)

W3C Recommendation 8 December 2009

This version:
http://www.w3.org/TR/2009/REC-xml-names-20091208/
Latest version:
http://www.w3.org/TR/xml-names/
Previous versions:
http://www.w3.org/TR/2006/REC-xml-names-20060816/ http://www.w3.org/TR/2009/PER-xml-names-20090806/
Editors:
Tim Bray, Textuality <tbray@textuality.com>
Dave Hollander, Contivo, Inc. <dmh@contivo.com>
Andrew Layman, Microsoft <andrewl@microsoft.com>
Richard Tobin, University of Edinburgh and Markup Technology Ltd <richard@inf.ed.ac.uk>
Henry S. Thompson, University of Edinburgh and W3C <ht@w3.org> - Third Edition

Please refer to the **errata** for this document, which may include normative corrections.

See also **translations**.

This document is also available in these non-normative formats: XML and HTML highlighting differences from the second edition.