The eXtensible Markup Language (XML)

Part 1: Background for XML

- An Extensible Markup Language (XML) document describes the *structure of data*
- XML and HTML have a similar syntax … both derived from SGML
- XML has no mechanism to specify the format for presenting data to the user
- An XML document resides in its own file with an ‘.xml’ extension

Main Components of an XML Document

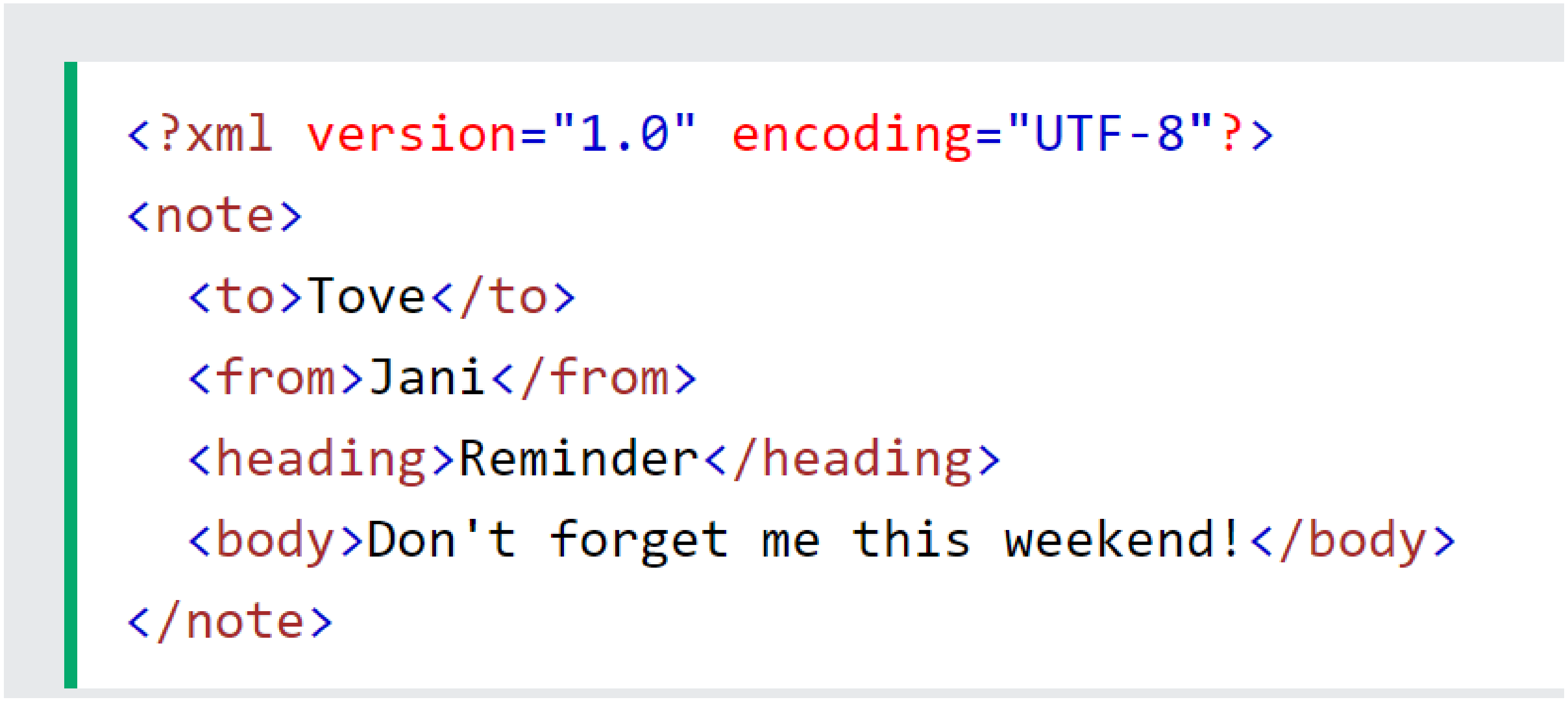
- Elements: <hello>
- Attributes: <item id=“33905”>
- Entities: &lt; (<)
- Advanced Components
  - CData Sections
  - Processing Instructions

An Example XML Document

- An example of an well-commented XML document

The Basic Rules

- XML is case sensitive
- All start tags must have end tags
- Elements must be properly nested
- XML declaration is the first statement
- Every document must contain a root element
- Attribute values must have quotation marks
- Certain characters are reserved for parsing



```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Elements vs. Attributes

Take a look at these two examples:

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <gender>female</gender>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

In the first example, gender is an attribute. In the last example, gender is an element. Both examples provide the same information.

There are no rules about when to use attributes or when to use elements in XML.

- Links:

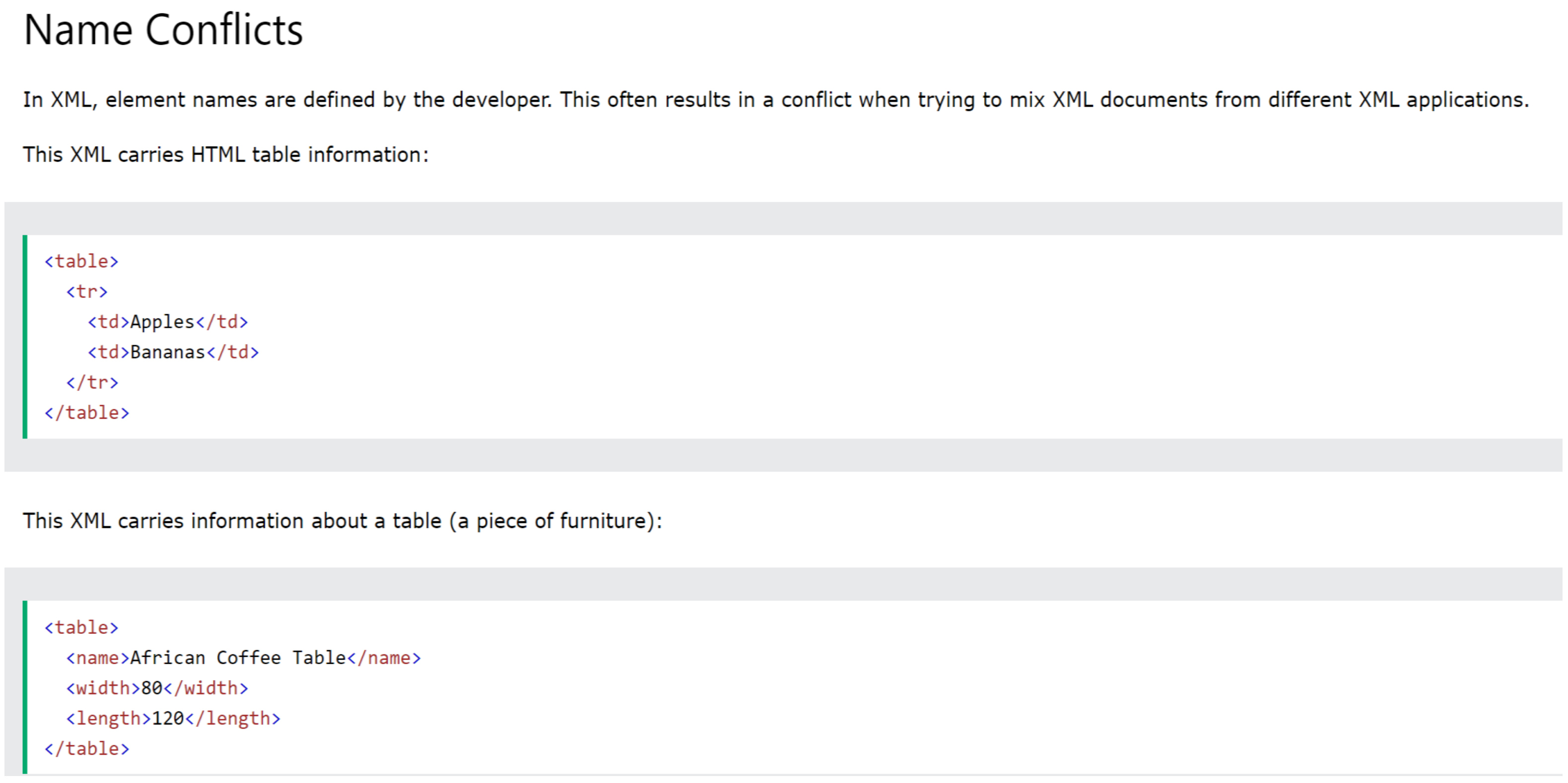
1. https://www.freeformatter.com/xml-formatter.html#ad-output

2. https://www.webtoolkitonline.com/xml-formatter.html

3. https://codebeautify.org/xmlviewer

Namespace

- XML Namespaces provide a method to avoid element name conflicts.



Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

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

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

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

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

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a **namespace** for the prefix must be defined.

The namespace can be defined by an **xmlns** attribute in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

```xml
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

In the example above:

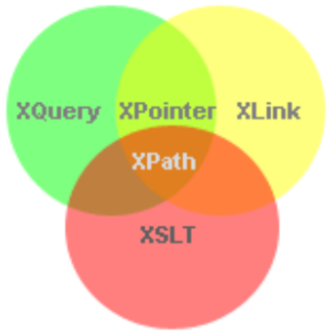
The xmlns attribute in the first <table> element gives the h: prefix a qualified namespace.

The xmlns attribute in the second <table> element gives the f: prefix a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

XPath

- XPath is a major element in the XSLT standard.
- XPath can be used to navigate through elements and attributes in an XML document.



- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT and in XQuery
- XPath is a W3C recommendation

XSD Complex Elements

- A complex element is an XML element that contains other elements and/or attributes.
- There are four kinds of complex elements:
- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Examples of Complex Elements

A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

A complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

A complex XML element, "food", which contains only text:

```
<food type="dessert">Ice cream</food>
```

A complex XML element, "description", which contains both elements and text:

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

Common Errors for Element Naming

- Do not use white space when creating names for elements
- Element names cannot begin with a digit, although names can contain digits
- Only certain punctuation allowed – periods, colons, and hyphens

Walking through an Example

- Modify the computer.xml document
  - Add a new element named “software” with an attribute named “language”
  - The attribute’s value should be the name of a programming language
  - Create another XML element called “IFStatment”
  - Use the IFStatment element to tag the following data: if (a < b && b >= 0)
  - Close the “software” tag
- After you have added these new items into the XML document, parse it again to ensure that it is still well formed. Use the feedback to correct any errors.

Part 2: Legal Building Blocks of XML

- A Document Type Definition (**DTD**) allows the developer to create a set of rules to specify legal content and place restrictions on an XML file
- If the XML document does not follow the rules contained within the DTD, a parser generates an error
- An XML document that conforms to the rules within a DTD is said to be **valid**

Why Use a DTD?

- A single DTD ensures a common format for each XML document that references it
- An application can use a standard DTD to verify that data that it receives from the outside world is valid
- A description of legal, valid data further contributes to the interoperability and efficiency of using XML

Some Example DTD Declarations

- <u>Example 1: The Empty Element</u>

```
<!ELEMENT Bool (EMPTY)> <!--DTD declaration of empty element-
   ->
<Bool Value="True"></Bool> <!--Usage with attribute in XML
   file-->
```

- <u>Example 2: Elements with Data</u>

```
<!ELEMENT Month (#PCDATA)> <!--DTD declaration of an element->
<Month>April</Month> <!—Valid usage within XML file-->


<Month>This is a month</Month> <!—Valid usage within XML file-->
<Month> <!—Invalid usage within XML file, can’t have children!-->
<January>Jan</January>
<March>March</March>
</Month>
```

Some Example DTD Declarations

- Example 3: Elements with Children

  To specify that an element must have a single child element, include the element name within the parenthesis.

```
<!ELEMENT House (Address)> <!—A house has a single address-–>
<House> <!—Valid usage within XML file-–>
<Address>1345 Preston Ave Charlottesville Va 22903</Address>
</House>
```

An element can have multiple children. A DTD describes multiple children using a *sequence,* or a list of elements separated by commas. The XML file must contain one of each element in the specified order.

```
<!--DTD declaration of an element-–>
  <!ELEMENT address (person,street,city, zip)>
  <!ELEMENT person  (#PCDATA)>
  <!ELEMENT street  (#PCDATA)>
  <!ELEMENT city    (#PCDATA)>
  <!ELEMENT zip     (#PCDATA)>
  <!—Valid usage within XML file-–>
  <address>
  <person>John Doe</person>
  <street>1234 Preston Ave.</street>
  <city>Charlottesville, Va</city>
  <zip>22903</zip>
  </address>
```

Cautions concerning DTDs

- All element declarations begin with <!ELEMENT and end with >
- The ELEMENT declaration is *case sensitive*
- The programmer must declare all elements within an XML file
- Elements declared with the #PCDATA content model can not have children
- When describing sequences, the XML document must contain exactly those elements in exactly that order.

Walking Through an Example

- Using the file “music.xml” contained in the extras folder, create a Document Type Definition that describes all of the elements. The goals for this exercise are to:
  - map out the elements
  - define each element
  - pick the best content model for each element
  - correctly order the elements using sequences
- Internally embed the DTD within the XML document

Part 3: XML and Supplementary Technologies

- The W3C Document Object Model (DOM)
  - an API that allows developers to programmatically manage and access XML nodes
  - allows programmers to update and change XML documents within an application
  - reads the whole XML file and then stores a hierarchical tree structure containing all elements within the document
  - This tree has a single root node, which is the root element, and may contain many children, each of which represents an XML element

W3C DOM with JavaScript

- Example 1: Loading the XML document: DOMDocument
  - The programmer can use a Microsoft Active X object to parse an XML file

```
//Instantiate DOMDocument object
var XMLfile = new ActiveXObject("Msxml2.DOMDocument");
XMLfile.load("newspaper.xml");
var rootElement = XMLfile.documentElement;
document.write("The root node of the XML file is: ");
document.writeln("<b>" + rootElement.nodeName +"</b>");
```

W3C DOM with JavaScript

- Example 2: Accessing the Children Elements
  - The *childNodes* member of any element node gives the programmer access to all of the sibling nodes of that element

```
 //traverse through each child of the root element
//and print out its name
for (i=0; i<rootElement.childNodes.length; i++) {
    var node = rootElement.childNodes.item(i);
    document.write("The name of the node is ");
    document.write("<b>" + node.nodeName + "</b>");
}
```

W3C DOM with JavaScript

- Example 3: Getting Element Attributes

```
//traverse through each child of the root element
//and print out its name
   for (i=0; i<rootElement.childNodes.length; i++) {
        //get the current element
        var elementNode = rootElement.childNodes.item(i);
        document.writeln("Processing Node: " +
                elementNode.nodeName + "<BR>");

        var attributeValue;
        //get an attribute value by specific name
        attributeValue = elementNode.getAttribute("articleID");
        //print it out
        document.writeln("Attribute value: <b>" + attributeValue +
                " </b><br>");
   }
```

Cautions with DOM

- Make sure that the XML file resides in the same directory as the html file with the JavaScript code

- The Attribute node does not appear as the child node of any other node type; it is not considered a child node of an element

- Use caution when outputting raw XML to Internet Explorer.If the programmer uses the *document.writeln* method, IE attempts to interpret the XML tags and jumbles the text. Instead, use an alert box when debugging.

Walking through an Example

1. Create an HTML file with notepad. Insert some JavaScript code that will parse newspaper.xml into a DOM tree. Print out the attribute values for each articleID.