

# **XSL Transformations (XSLT)**

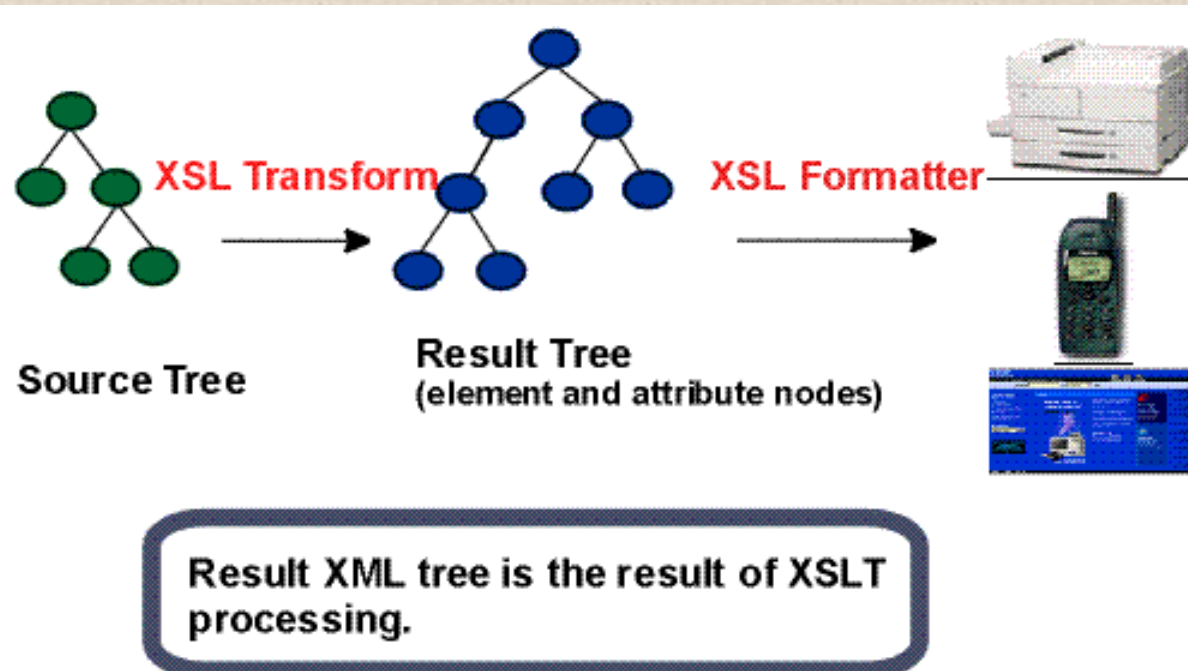
**Meghasyam Bokam**  
**April'1<sup>st</sup>, 2002**

# Background

**XSL** (Extensible Stylesheet Language) : is a language for expressing style sheets. It consists of two parts:

- a language for transforming XML documents - **XSLT**
- an XML vocabulary for specifying formatting semantics - **XSLFO**

XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.



# What is XSLT?

- is a language for transforming XML documents into other XML documents.
- is designed for use as part of XSL
- describes rules for transforming a source tree into a result tree.
- uses the expression language defined by XPath for selecting nodes.
- is designed primarily for XML-to-XML and XML-to-HTML transformations.

# Data Model

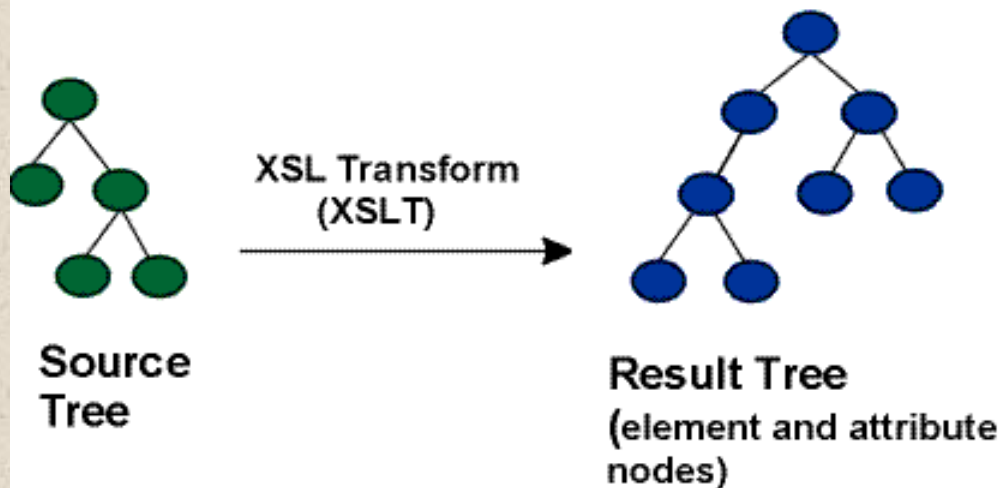
- The data model used by XSLT is the same as that used by Xpath.
- The tree structure of XML document consists of 7 nodes:
  - The root node
  - Element nodes
  - Text nodes
  - Attribute nodes
  - Namespace nodes
  - Processing Instruction nodes
  - Comment nodes
- XSLT operates on source, result and stylesheet documents using the same data model.



# How Tree Transformation Works?

- a transformation expressed in XSLT is called a style sheet.
- this style sheet describes rules for transforming a source tree into a result tree.
- an XSLT processor reads both an XML document and an XSLT style sheet, and it outputs a new XML document or fragment.
- the transformation is achieved by associating patterns with templates.
- a pattern is matched against elements in the source tree.
- a template is instantiated to create part of the result tree.

With tree transformation, the structure of the result tree can be quite different from the structure of the source tree



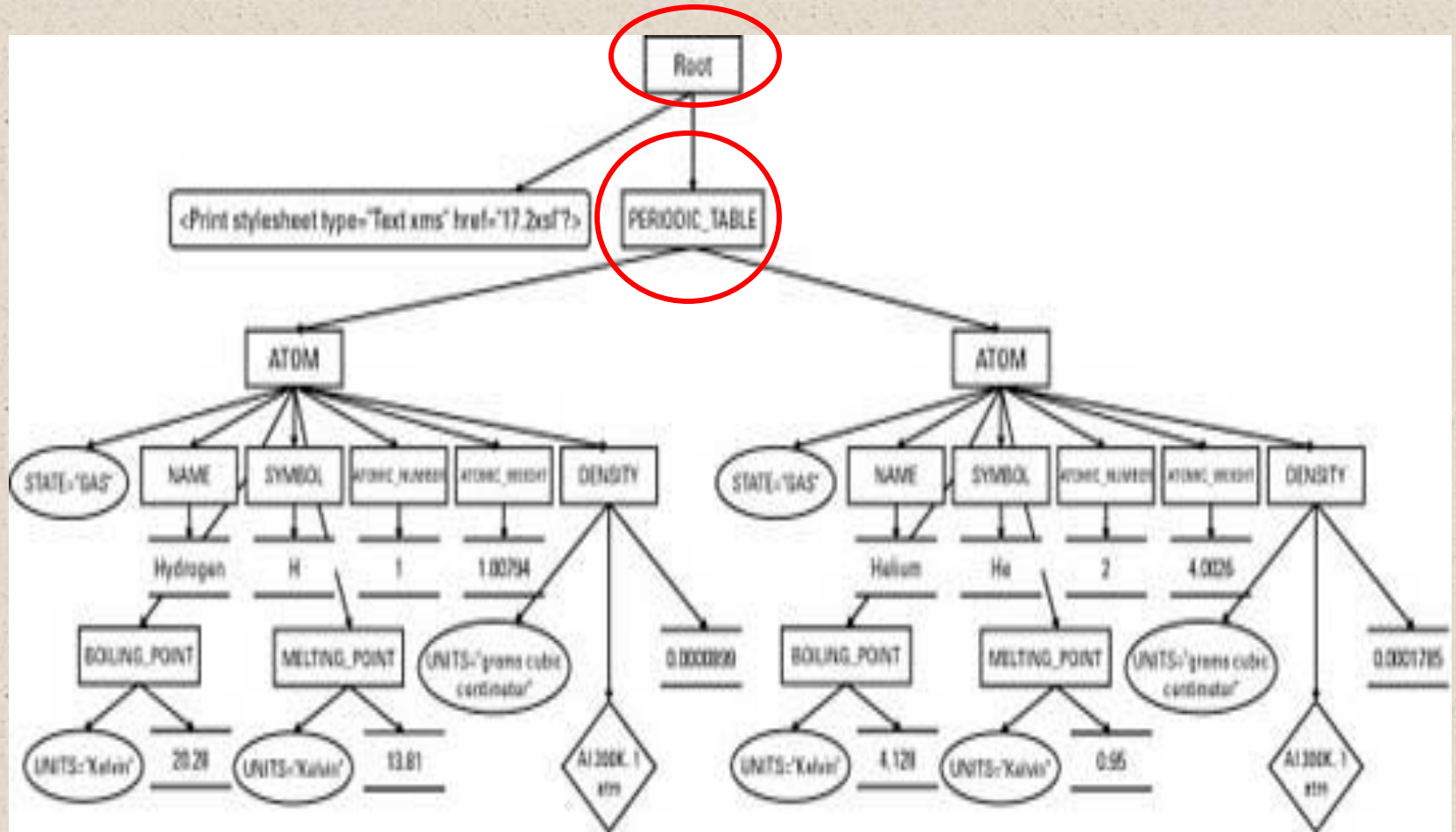
In constructing the result tree, the source tree can be filtered and reordered, and arbitrary structure and generated content can be added.

# Example: Periodic Table with two Atoms

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xml" href="eg.xml"?>
<PERIODIC_TABLE>
  <ATOM STATE="GAS">
    <NAME>Hydrogen</NAME>
    <SYMBOL>H</SYMBOL>
    <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
    <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
    <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter">
      <!-- At 300K, 1 atm -->
      0.0000899
    </DENSITY>
  </ATOM>
  <ATOM STATE="GAS">
    <NAME>Helium</NAME>
    <SYMBOL>He</SYMBOL>
    <ATOMIC_NUMBER>2</ATOMIC_NUMBER>
    <ATOMIC_WEIGHT>4.0026</ATOMIC_WEIGHT>
    <BOILING_POINT UNITS="Kelvin">4.216</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">0.95</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter">
      <!-- At 300K -->
      0.0001785
    </DENSITY>
  </ATOM>
</PERIODIC_TABLE>
```



# Tree Representation of Periodic Table



# XSL Templates

- Template rules identify the nodes to which they apply by using a **pattern**.
- A template rule is specified with the **xsl:template** element.
- The content of the **xsl:template** element is the actual template to be instantiated
- The **match** attribute is a **Pattern** that identifies the source node or nodes to which the rule applies.

*<xsl:template match = pattern >*

- The syntax for **patterns** is a subset of the syntax for Xpath expressions.

# The **xsl:apply-templates** element

- **xsl:apply-templates** in the output template, tells the formatter to compare each child element of the matched source element against the templates in the style sheet, and, if a match is found, output the template for the matched node.
- The template for the matched node may itself contain **xsl:apply-templates** elements to search for matches for its children.

# An XSLT style sheet (eg.xsl) for the periodic table with two template rules

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
  <xsl:template match="PERIODIC_TABLE">
    <body>
      <xsl:apply-templates/>
    </body>
  </xsl:template>
  <xsl:template match="ATOM">
    An Atom
  </xsl:template>
</xsl:stylesheet>
```

The result is:

```
<html>
  <body>
    An Atom
    An Atom
  </body>
</html>
```

# The *select* attribute

```
<xsl:template match="ATOM">  
    <xsl:apply-templates select="NAME"/>  
</xsl:template>
```

The result is:

```
<html>  
  <body>  
    Hydrogen  
    Helium  
  </body>  
</html>
```



# Computing the value of a Node with xsl:value-of

```
<xsl:template match="ATOM">  
  <xsl:value-of select="NAME"/>  
</xsl:template>
```

The result is:

```
<html>  
  <body>  
    Hydrogen  
    Helium  
  </body>  
</html>
```

# Values of Nodes in xsl:value-of

Node Type:	Value:
Root	The value of the root element
Element	The concatenation of all parsed character data contained in the element, including character data in any of the descendants of the element
Text	The text of the node; essentially the node itself
Attribute	The attribute value after entities are resolved and leading and trailing white space is stripped; does not include the name of the attribute, the equals sign, or the quotation marks
Namespace	The URI of the namespace
Processing instruction	The data in the processing instruction; does not include the processing instruction , <? or ?>
Comment	The text of the comment, <!-- and --> not included

# Processing Multiple Elements with xsl:for-each

```
<xsl:template match="PERIODIC_TABLE">  
  <xsl:for-each select="ATOM">  
    <xsl:value-of select="."/>  
  </xsl:for-each>  
</xsl:template>
```

The `select="."` in this template tells the formatter to take the value of the matched element, `ATOM` in this example.

# Patterns for Matching Nodes

The **match** attribute of the **xsl:template** element supports a complex syntax that allows you to express exactly which nodes you do and do not want to match

## Matching the root node

To specify the root node in a rule, you give its **match** attribute the value `"/`.

Eg. `<xsl:template match="/">`

## **Matching element names**

Eg. `<xsl:template match="ATOM">`

## **Matching children with /**

The / symbol is used to match specified hierarchies of elements.

Eg. 1. `<xsl:template match="ATOM/SYMBOL">`

Eg. 2. `<xsl:template match="PERIODIC_TABLE/*/SYMBOL">`



## Matching descendants with //

The double slash, //, refers to a descendant element at an arbitrary level

Eg. 1. `<xsl:template match="PERIODIC_TABLE//NAME">`

Eg. 2. `<xsl:template match="//ATOMIC_NUMBER">`

## Matching by ID

This is used to apply a particular style to a particular single element without changing all other elements of that type. This is done with the `id()` selector, which contains the ID value in single quotes.

Eg. `<xsl:template match="id('e47')">`

`<b><xsl:value-of select="."/;></b>`

`</xsl:template>`

## Matching attributes with @

The @ sign matches against attributes and selects nodes according to attribute names.

Ex:

```
<xsl:template match="MELTING_POINT">
    <xsl:apply-templates select="@UNITS"/>
</xsl:template>
<xsl:template match="@UNITS">
    <I><xsl:value-of select="."/></I>
</xsl:template>
```

The output is:

```
<I>kelvin</I>
<I>kelvin</I>
```

## Using the or operator |

The vertical bar (|) allows a template rule to match multiple patterns.

Ex: `<xsl:template match="ATOMIC_NUMBER|ATOMIC_WEIGHT">`

## Testing with [ ]

You can test for more details about the nodes that match a pattern using []. You can perform many different tests including:

- Whether an element contains a given child, attribute, or other node
- Whether the value of an attribute is a certain string
- Whether the value of an element matches a string
- What position a given node occupies in the hierarchy

Ex: 1. `<xsl:template match="ATOM[MELTING_POINT]">`

Ex: 2. `<xsl:template match="ATOM[DENSITY/@UNITS]">`

Ex: 3. `<xsl:template match="ATOM[ATOMIC_NUMBER='10']">`

# The Default Template Rules

## The default rule for elements:

This default rule applies to element nodes and the root node:

```
<xsl:template match="*/">  
  <xsl:apply-templates/>  
</xsl:template>
```

`*/` is XPath shorthand for "any element node or the root node."

The purpose of this rule is to ensure that all elements are recursively processed even if they aren't reached by following the explicit rules.



## **The default rule for text nodes and attributes:**

This rule is:

```
<xsl:template match="text()|@"*>
```

```
  <xsl:value-of select="."/>
```

```
</xsl:template>
```

This rule matches all text and attribute nodes (`match="text()|@"*`) and outputs the value of the node (`<xsl:value-of select="."/>`).

This rule ensures that at the very least an element's text is output, even if no rule specifically matches it.

This rule also copies attribute values (but not names).

## **The default rule for processing instructions and comments:**

It simply says to do nothing; that is, drop the processing instructions and comments from the output as if they didn't exist. It looks like this:

```
<xsl:template match="processing-instruction()|comment()"/>
```



# Attribute value templates

- Attribute value templates copy data from the input document to attribute values in the output.
- Inside attribute values, data enclosed in curly braces { } takes the place of the **xsl:value-of** element.

Ex: To convert the periodic table into empty ATOM elements, the template can be written as:

```
<xsl:template match="ATOM">
  <ATOM NAME="{NAME}"
    ATOMIC_WEIGHT="{ATOMIC_WEIGHT}"
    ATOMIC_NUMBER="{ATOMIC_NUMBER}" />
</xsl:template>
```

The **output** is of the form:

```
<ATOM NAME="Vanadium"  
    ATOMIC_WEIGHT="50.9415"  
    ATOMIC_NUMBER="23" />
```

Ex 2:

```
<xsl:template match="ATOM">  
    <A HREF="{SYMBOL}.html">  
        <xsl:value-of select="NAME"/>  
    </A>  
</xsl:template>
```

The **output** is of the form:

```
<A HREF=H.html>Hydrogen</A>  
<A HREF=He.html>Helium</A>
```

# Inserting elements into the output with `xsl:element`

- The name of the element is given by an attribute value template in the `name` attribute of `xsl:element`.
- The content of the element derives from the content of the `xsl:element` element, which may include `xsl:attribute`, `xsl:processing-instruction`, and `xsl:comment` instructions to insert these items.

Ex: To replace the `ATOM` elements with `GAS`, `LIQUID`, and `SOLID` elements

```
<xsl:template match="ATOM">
  <xsl:element name="{@STATE}">
    <NAME><xsl:value-of select="NAME"/></NAME>
    <!-- rules for other children -->
  </xsl:element>
</xsl:template>
```

# Inserting attributes into the output with `xsl:attribute`

- The name of the attribute is specified by the `name` attribute of the `xsl:attribute` element.
- The value of the attribute is given by the contents of the `xsl:attribute` element.
- Each `xsl:attribute` element is a child of either an `xsl:element` element or a literal element.

Ex:

```
<xsl:template match="ATOM">
  <LI><A>
    <xsl:attribute name="HREF">
      <xsl:value-of select="SYMBOL"/>.html
    </xsl:attribute> <xsl:value-of select="NAME"/>
  </A></LI>
</xsl:template>
```

The output is of the form:

```
<LI><A HREF="H.html">Hydrogen</A></LI>
<LI><A HREF="He.html">Helium</A></LI>
```



# Generating processing instructions with `xsl:processing-instruction`

- The target of the processing instruction is specified by a required `name` attribute
- The contents of the `xsl:processing-instruction` element become the contents of the processing instruction.

Ex: This rule replaces `PROGRAM` elements with a `gcc` processing instruction:

```
<xsl:template match="PROGRAM">
  <xsl:processing-instruction name="gcc"> -O4
</xsl:processing-instruction>
</xsl:template>
```

Output:

```
<?gcc -O4 ?>
```



# Generating comments with `xsl:comment`

- The `xsl:comment` element inserts a comment in the output document.
- It has no attributes.
- Its contents are the text of the comment.

Ex:

```
<xsl:template match="ATOM">  
    <xsl:comment>There was an atom here once.</xsl:comment>  
</xsl:template>
```

Output: This rule replaces `ATOM` nodes with this comment:

```
<!--There was an atom here once.-->
```

# Generating text with xsl:text

The `xsl:text` element inserts its contents into the output document as literal text

Ex:

```
<xsl:template match="ATOM">  
    <xsl:text>There was an atom here once.</xsl:text>  
</xsl:template>
```

**Output:** This rule replaces each ATOM element with the string:

There was an atom here once.

# Copying the Context Node with `xsl:copy`

- The `xsl:copy` element copies the source node into the output tree.
- Child elements, attributes, and other content are not automatically copied.

Ex: An XSLT style sheet that strips comments from a document

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="*|@*|processing-instruction()|text()">
    <xsl:copy>
      <xsl:apply-templates select="*|@*|processing-instruction()|text()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

# Sorting Output Elements

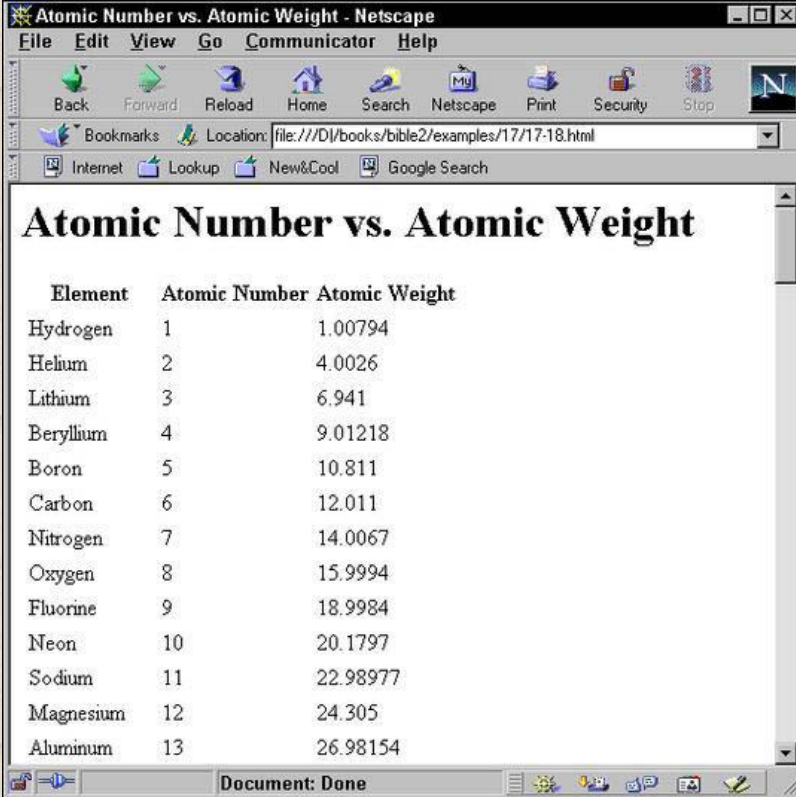
- The **xsl:sort** element sorts the output elements into a different order than they appear in the input.
- An **xsl:sort** element appears as a child of an **xsl:apply-templates** element or **xsl:for-each** element.
- The **select** attribute of the **xsl:sort** element defines the key used to sort the element's output by **xsl:apply-templates** or **xsl:for-each**

## Ex: An XSLT style sheet that sorts by atomic number

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="PERIODIC_TABLE">
    <html>
      <head>
        <title>Atomic Number vs. Atomic Weight</title>
      </head>
      <body>
        <h1>Atomic Number vs. Atomic Weight</h1>
        <table>
          <th>Element</th>
          <th>Atomic Number</th>
          <th>Atomic Weight</th>
          <xsl:apply-templates>
            <xsl:sort data-type="number" select="ATOMIC_NUMBER"/>
          </xsl:apply-templates>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="ATOM">
    <tr>
      <td><xsl:apply-templates select="NAME"/></td>
      <td><xsl:apply-templates select="ATOMIC_NUMBER"/></td>
      <td><xsl:apply-templates select="ATOMIC_WEIGHT"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```



## Output: Atoms numerically sorted by atomic number



The image shows a Netscape browser window titled "Atomic Number vs. Atomic Weight - Netscape". The address bar shows the file path "file:///D:/books/bible2/examples/17/17-18.html". The main content area displays a table titled "Atomic Number vs. Atomic Weight" with three columns: "Element", "Atomic Number", and "Atomic Weight". The table lists elements from Hydrogen to Aluminum, sorted by increasing atomic number. The status bar at the bottom indicates "Document: Done".

Element	Atomic Number	Atomic Weight
Hydrogen	1	1.00794
Helium	2	4.0026
Lithium	3	6.941
Beryllium	4	9.01218
Boron	5	10.811
Carbon	6	12.011
Nitrogen	7	14.0067
Oxygen	8	15.9994
Fluorine	9	18.9984
Neon	10	20.1797
Sodium	11	22.98977
Magnesium	12	24.305
Aluminum	13	26.98154

# Modes

- Modes allow an element to be processed multiple times, each time producing a different result.
- Both `xsl:template` and `xsl:apply-templates` have an optional **mode** attribute

## Ex: An XSLT style sheet that uses modes to format the same data differently in two different places

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/PERIODIC_TABLE">
    <HTML>
      <HEAD>
        <TITLE>The Elements</TITLE>
      </HEAD>
      <BODY>
        <H2>Table of Contents</H2>
        <UL>
          <xsl:apply-templates select="ATOM" mode="toc"/>
        </UL>
        <H2>The Elements</H2>
        <xsl:apply-templates select="ATOM" mode="full"/>
      </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match="ATOM" mode="toc">
    <LI><A>
      <xsl:attribute name="HREF">#<xsl:value-of select="SYMBOL"/></xsl:attribute>
      <xsl:value-of select="NAME"/>
    </A></LI>
  </xsl:template>
  <xsl:template match="ATOM" mode="full">
    <H3><A>
      <xsl:attribute name="NAME"> <xsl:value-of select="SYMBOL"/> </xsl:attribute>
      <xsl:value-of select="NAME"/>
    </A></H3>
    <P> <xsl:value-of select="."/> </P>
  </xsl:template>
```

# Defining Constants with xsl:variable

- The `xsl:variable` element defines a named string for use elsewhere in the style sheet via an attribute value template.
- It has a single attribute, **name**, which provides a name by which the variable can be referred to.
- The contents of the `xsl:variable` element provide the replacement text

Ex:

```
<xsl:variable name="copy01">  
    Copyright 2001 Elliotte Rusty Harold  
</xsl:variable>
```

Ex: Accessing the variable “copy01”

1. `<BLOCK COPYRIGHT="{ $copy01 }"> </BLOCK>`
2. `<xsl:value-of select="$copy01"/>`



# Named Templates

- Named templates enable you to include data from the place where the template is applied, rather than merely inserting fixed text.
- The **xsl:template** element has a **name** attribute by which it can be explicitly invoked
- The **xsl:call-template** element has a required **name** argument that names the template it will call.
- When processed, the **xsl:call-template** element is replaced by the contents of the **xsl:template** element it names.



Ex:

```
<xsl:template name="ATOM_CELL">
```

```
  <td>
    <font face="Times, serif" color="blue" size="2">
      <b>
        <xsl:value-of select="."/>
      </b>
    </font>
  </td>
```

```
</xsl:template>
```

Ex: Calling the named template ATOM\_CELL

```
<xsl:template match="ATOMIC_NUMBER">
```

```
  <xsl:call-template name="ATOM_CELL"/>
```

```
</xsl:template>
```

# Passing Parameters to Templates

- Parameters are passed to templates using the `xsl:with-param` element.
- The value of the parameter is specified in the same way as for `xsl:variable` and `xsl:param`.

Ex:

```
<xsl:template name="ATOM_CELL">
  <xsl:param name="file">index.html</xsl:param>
  <td>
    <font face="Times, serif" color="blue" size="2">
      <b> <a href="{ $file }"><xsl:value-of select="."/></a> </b>
    </font>
  </td>
</xsl:template>
```

## Ex: Passing the parameter “file”

```
<xsl:template match="ATOMIC_NUMBER">  
  <xsl:call-template name="ATOM_CELL">  
    <xsl:with-param name="file">atomic_number.html</xsl:with-param>  
  </xsl:call-template>  
</xsl:template>
```

The **Output** is of the form:

```
<td>  
  <font face="Times, serif" color="blue" size="2">  
    <b>  
      <a href="atomic_number.html">52</a>  
    </b>  
  </font>  
</td>
```

# Making Choices

## xsl:if

- The `xsl:if` element provides a simple facility for changing the output based on a pattern.
- The `test` attribute of `xsl:if` contains an expression that evaluates to a boolean.

**Ex:** A comma and a space is added after all names except the last name

```
<xsl:template match="ATOM">
  <xsl:value-of select="NAME"/>
  <xsl:if test="position() != last()">, </xsl:if>
</xsl:template>
```

## xsl:choose

- It selects one of several possible outputs depending on several possible conditions
- Each condition and its associated output template is provided by an **xsl:when** child element
- If none of the **xsl:when** elements are true, the **xsl:otherwise** child element is instantiated.

**Ex:** Changes the color of the output based on whether the **STATE** attribute of the **ATOM** element is **SOLID** or **LIQUID**

```
<xsl:template match="ATOM">
```

```
  <xsl:choose>
```

```
    <xsl:when test="@STATE='SOLID'"> ←—————  
      <P style="color: black"> <xsl:value-of select="."/> </P>
```

```
    </xsl:when>
```

```
    <xsl:when test="@STATE='LIQUID'"> ←—————  
      <P style="color: blue"> <xsl:value-of select="."/> </P>
```

```
    </xsl:when>
```

```
    <xsl:otherwise> ←—————  
      <P style="color: green"> <xsl:value-of select="."/> </P>
```

```
    </xsl:otherwise>
```

```
  </xsl:choose>
```



# Merging Multiple Style Sheets

## Importing with **xsl:import** :

- An XSLT stylesheet may import another XSLT stylesheet using an **xsl:import** element.
- The **xsl:import** element is a top-level element whose **href** attribute provides the URI of a style sheet to import.
- All **xsl:import** elements must appear before any other top-level element in the **xsl:stylesheet** root element

```
<xsl:import  
    href = uri-reference />
```

## Inclusion with **xsl:include** :

- An **xsl:include** element can occur anywhere at the top level after the last **xsl:import** element.

# Output Methods

- The **xsl:output** element allows stylesheet authors to specify how they wish the result tree to be output
- The **method** attribute of the **xsl:output** element specifies which output method to use and normally has one of these three values:
  - XML
  - HTML
  - Text

```
<!-- Category: top-level-element -->  
<xsl:output  
    method = "xml" | "html" | "text" />
```

# Conclusion

- An XSL transformation applies rules to a tree read from an XML document to transform it into an output tree written out as an XML document
- XPath expressions are a superset of match patterns used by the `select` attribute of `xsl:apply-templates`, `xsl:value-of`, `xsl:for-each`, `xsl:copy-of`, `xsl:variable`, `xsl:param`, `xsl:with-param`, and `xsl:sort` elements
- Modes can apply different templates to the same element from different locations in the style sheet.
- Named templates help you reuse common template code
- The `xsl:import` and `xsl:include` elements merge rules from different style sheets.

# Reference

- **XSL Transformations (XSLT) V1.0**

*<http://www.w3.org/TR/xslt>*

- **XML BIBLE**

*<http://www.ibiblio.org/xml/books/bible2/chapters/ch17.html>*

- **Apache Xalan – XSLT processor**

*<http://xml.apache.org/xalan/>*