The Limitations of CSS

- CSS displays element content as it appears in the XML document, you cannot change the format of the content itself.
- CSS cannot change the order in which elements appear in a document.
- CSS displays only element content, not element attributes.
- CSS cannot do computations.
- CSS does not allow you to add additional text to element content.
- An element can be formatted only one way in a document.
The History of XSL

- In 1998, the W3C began developing the Extensible Stylesheet Language, or XSL.
- XSL is composed of two parts:
  - XSL-FO (Extensible Stylesheet Language – Formatting Objects) is used for the layout of paginated documents.
  - XSLT (Extensible Stylesheet Language Transformations) is used to transform the contents of an XML document into another document format.

\[
XSL = XSLT + XSL-FO
\]
An XSLT style sheet contains instructions for transforming the contents of an XML document into new XML, Hypertext Markup Language (HTML), Extensible Hypertext Markup Language (XHTML), or plain text documents.

XML Path Language, or XPath, is a companion technology to XSLT that helps identify and find nodes in XML documents—elements, attributes, and other structures.

An XSLT style sheet document is itself an XML document.

An XSLT style sheet document has an extension .xsl
An XSLT style sheet converts a source document of XML content into a result document by using the XSLT processor.
<?xml version="1.0" ?>
<!-- hello.xml -->
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<myMessage>
  <message>HelloWorld XSLT!</message>
</myMessage>

<!-- hello.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="myMessage">
    <html><body>
      <h1><xsl:value-of select="message"/></h1>
    </body></html>
  </xsl:template>
</xsl:stylesheet>

<!-- result.html -->
<html>
  <body>
    <h1>HelloWorld XSLT!</h1>
  </body>
</html>
Running Example

HelloWorld XSLT!
XSLT Processor

- An XSLT processor is a piece of software that reads an XSLT stylesheet, reads an XML document, and builds an output document by applying the instructions in the stylesheet to the information in the input document.

- An XSLT processor can be
  - built into a web browser, just as MSXML in Internet Explorer, TransforMiiX in Mozilla browser engine and employed by Netscape and Firefox.
  - built into a web server, as in the Apache XML Project’s Cocoon
  - a standalone program run form the command line as in SAXON or Xalan
XSLT Transformation

The transformation can be performed by a server or a client.

In a server-side transformation, the server receives a request from a client, applies the style sheet to the source document, and returns the result document to the client.

In a client-side transformation, a client requests retrieval of both the source document and the style sheet from the server, then performs the transformation, and generates the result document.
There are two ways to view a result document

1. use a browser that contains an XSLT processor to view the source document.
   - The browser transforms the source document and presents the result document in the browser window

2. use a third-party XSLT processor to create the result document as a separate file on their computer, and then view that file in a browser that does not contain an XSLT processor
   - You have to regenerate the result file every time you make a change to the source document, or the style sheet
Creating an XSLT Style Sheet

• To create an XSLT style sheet, the general structure:

```xml
<?xml version =“1.0”?>
<xsl:stylesheet version = “1.0”
    xmlns:xsl = “http://www.w3.org/1999/XSL/Transform”>
    Content of the style sheet
</xsl:stylesheet>
```

• The `<xsl:stylesheet>` tag can be substituted for the `<xsl:transform>`
Specifying the Output Method

- The default output method for XSLT is XML unless the document element in the result is `<html>`. In such a case, the default output method is HTML.
- To control how the processor formats the source document, you can specify the output method using the XSLT element

\[
<\text{xsl:output attributes} />\]
## Attributes of the `<xsl:output>` Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>Defines the output format using the value xml, html, or text</td>
</tr>
<tr>
<td>version</td>
<td>Specifies the version of the output</td>
</tr>
<tr>
<td>encoding</td>
<td>Specifies the character encoding</td>
</tr>
<tr>
<td>omit-xml-declaration</td>
<td>Specifies whether to omit an XML declaration in the first line of the result document (yes) or to include it (no)</td>
</tr>
<tr>
<td>standalone</td>
<td>Specifies whether a standalone attribute should be included in the output and sets its value (yes or no)</td>
</tr>
<tr>
<td>doctype-public</td>
<td>Sets the URI for the public identifier in the <code>&lt;!DOCTYPE&gt;</code> declaration</td>
</tr>
<tr>
<td>doctype-system</td>
<td>Sets the system identifier in the <code>&lt;!DOCTYPE&gt;</code> declaration</td>
</tr>
<tr>
<td>cdata-section-elements</td>
<td>Specifies a list of element names whose content should be output in CDATA sections</td>
</tr>
<tr>
<td>indent</td>
<td>Specifies whether the output should be indented to better display its structure (indentations are automatically added to HTML files without use of this attribute)</td>
</tr>
<tr>
<td>media-type</td>
<td>Sets the MIME type of the output</td>
</tr>
</tbody>
</table>
<xsl:output
method="xml|html|text"
version="string"
encoding="string"
omit-xml-declaration="yes|no"
standalone="yes|no"
doctype-public="string"
doctype-system="string"
cdata-section-elements="namelist"
indent="yes|no"
media-type="string"/>
xsl:output Examples

- To instruct XSLT processors to create HTML 4.0 files, you insert the following tag directly after the opening <xsl:stylesheet> tag:

  <xsl:output method="html" version="4.0" />

- To transform one XML document into another

  <xsl:output method="xml" version="1.0" />
**xsl:output Examples**

- To remove XML declaration from the result document, you use the following open method:

  `<xsl:output method="xml" version="1.0" omit-xml-declaration="yes" />`

- To create a plain text file

  `<xsl:output method="text" />`
xsl:output Examples

- `<xsl:output method="xml" indent="yes" encoding="UTF-8"/>

- `<xsl:output doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"/>

- `<xsl:output doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>

- `<xsl:output cdata-section-elements="notes"/>

- `<xsl:output standalone="no"/>

- `<xsl:output output="xml" media-type="text/xml"/>
how you can use XSLT to write a program in the C# programming language
Templates

- XSLT stylesheets are collections of templates.
- A template is a mixture of output data and instructions that define how a particular section of the source document should be transformed in the result document.
- An XSLT processor runs through all the templates in the stylesheet and tests whether the node matches the template’s pattern. All the templates that match this node are candidates for processing, and the XSLT processor must select one with the highest precedence to process.
Templates

- Generally, a template with more specific information has higher priority than one that is more generic.
  - A pattern that contains specific hierarchical information has higher priority than a pattern that contains general information. For example, `chapter/section/para` is more specific than `para`, so it takes precedence.
  - The wildcard is more general than a specific element or attribute name and therefore has lower priority, for example, the pattern `chapter` takes priority over the wildcard pattern `*`.
  - A pattern with a predicate overrides a pattern with no predicate, for example, `chapter[@id="ch01"]` has higher priority than `chapter`
Creating Template

° To create a template, the syntax is:

```
<xsl:template match="node set">
  styles
</xsl:template>
```

° Where the match attribute of template elements are a subset of XPath expressions which may be used only descending axes: child and attribute. The shorthand "//" can be used but it’s in not expanded. Paths in a match attribute are evaluated right to left.

styles contains the XSLT instructions or data to be output in the result tree.
XPath in XSLT

- The root template sets up the initial code for the entire result document
  
  ```xml
  <xsl:template match=""/>
  ...
  </xsl:template>
  ```

- Match any element name (or attribute name if using the attribute axis).
  
  ```xml
  <xsl:template match=""*">
  ...
  </xsl:template>
  ```
XPath in XSLT (cont.)

- Match an element

```xml
<xsl:template match="entry/*/street">
    <!-- Match all street elements that are grandchildren of entry. -->
</xsl:template>

<xsl:template match="order//item">
    <!-- Match all item elements that are descendants of order. -->
</xsl:template>
```
XPath in XSLT (cont.)

- Match a specific element
  - Use [...] as a predicate filter to select a particular element

```xml
<xsl:template match="author/name[middle]">
  <!-- Match all name elements that have an author parent and a middle child. -->
</xsl:template>

<xsl:template match="entry/phone[1]">
  <!-- Match the first phone element that is a child of entry. -->
</xsl:template>
```
XPath in XSLT (cont.)

- Match a specific attribute
  - Use @attribute to select a particular attribute

```xml
<xsl:template match="phone[@location]">
  <!-- Match all phone elements that have a location attribute. -->
</xsl:template>

<xsl:template match="catalog/item[@id='123']">
  <!-- Match only item elements with id attribute of 123. -->
</xsl:template>
```
A template contains two types of content: *XSLT elements* and *literal result elements*

1. XSLT elements are those elements that are part of the XSLT namespace, usually within the namespace prefix “xsl” and are used to send commands to the XSLT processor.

2. A literal result element is text sent to the result document, but not acted upon by the XSLT processor. It is treated as raw text.
Creating the Root Template Example

```xml
<xsl:template match="/">
  <html>
    <head>
      <title>Stock Information</title>
      <link href="stock.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
      <h1>Hardin Financial</h1>
      <h2>Stock Information</h2>
    </body>
  </html>
</xsl:template>
```
Extracting Element Values

- To insert a node’s value into the result document, the syntax is:
  - `<xsl:value-of select="expression" />`
  - where `expression` is an XPath expression that identifies the node from the source document’s node tree

- If the node contains child elements in addition to text content, the text in those descendant nodes appears as well
Inserting a Node Value Example

Figure 6-16  Displaying the date and time element values

```xml
<xsl:template match="/">
  <html>
    <head>
      <title>Stock Information</title>
      <link href="stock.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
      <div id="datetime">
        Last updated: at
        <xsl:value-of select="portfolio/date" />
        at
        <xsl:value-of select="portfolio/time" />
      </div>
      <h1>Hardin Financial</h1>
      <h2>Stock Information</h2>
    </body>
  </html>
</xsl:template>
```
Processing Several Elements

- The for-each element creates a template-within-a-template to process a batch of nodes, the syntax is:

```xml
<xsl:for-each select="expression">
  styles
</xsl:for-each>
```

Where `expression` is an XPath expression that defines the group of nodes to which the XSLT and literal result elements are applied
Processing Several Elements

**Figure 6-20** Setting a style for each occurrence of the name element

```
<dl id="datetime"><b>Last Updated: </b>
  <s1:value-of select="portfolio/date" /> at
  <s1:value-of select="portfolio/time" />
</dl>
<h1>Hardin Financial</h1>
<h2>Stock Information</h2>
<s1:for-each select="portfolio/stock">
  <h3>
    <s1:value-of select="sName" />
  </h3>
</s1:for-each>
```

replace the three lines to generate the h3 heading with a for-each statement
Working with Templates

- The element `apply-templates` interrupts the current processing in the template and forces the XSLT engine to move on to the templates matching nodes of `select` attribute.

- If no `select` attribute, XSLT engine moves on to the template matching children of the current node.

- `<xsl:apply-templates select="expression" />`

- where `expression` is XPath expression for a node set in the source document
Creating Template Example

```
<xsl:template match="/">
  <html>
    <head>
      <title>Stock Information</title>
      <link href="stock.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
      <div id="datetime">Last Updated: </div>
      <xsl:value-of select="portfolio/date" /> at <xsl:value-of select="portfolio/time" />
    </div>
    <h1>Hardin Financial</h1>
    <h2>Stock Information</h2>
    <xsl:apply-templates select="portfolio/stock/sName" />
  </body>
</html>
</xsl:template>
```
Creating the Stock Template Example

Figure 6-23  Creating the stock template

```xml
<h1>Hardin Financial</h1>
<h2>Stock Information</h2>

```xml
<xsl:apply-templates select="portfolio/stock" />
</html>
</xsl:template>

```xml
<xsl:template match="stock">
  <div>
    <xsl:apply-templates select="sName" />
    <p><xsl:value-of select="description" /></p>
  </div>
</xsl:template>

<xsl:template match="sName">
  <h3>
    <xsl:value-of select="." />
  </h3>
</xsl:template>
```
Working with Attribute Nodes

- `<xsl:template match="sName">
  <h3>
    <xsl:value-of select="."/>
    (<xsl:value-of select="@symbol" />)
  </h3>
</xsl:template>`
Working with Attribute Nodes

```xml
<xsl:template match="today">
    <table>
        <tr>
            <th>Current</th>
            <th>Open</th>
            <th>High</th>
            <th>Low</th>
            <th>Volume</th>
        </tr>
        <tr>
            <xsl:apply-templates select="@current"/>
            <xsl:apply-templates select="@open"/>
            <xsl:apply-templates select="@high"/>
            <xsl:apply-templates select="@low"/>
            <xsl:apply-templates select="@vol"/>
        </tr>
    </table>
</xsl:template>
```
Using the Built-in Templates

- Each node has its own built-in template.
- The built-in template for root node and element nodes matches the document root and all elements in the node tree.
- The built-in template for all text nodes and attributes and causes their values to appear in the result document.
- The built-in template for comment and processing instructions in a source document do nothing.
Built-in Template for Root Node

- Processing starts at the root node. To force processing of the entire tree, the default behavior is to apply templates to all the children.

```xml
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
```
Built-in Template for Element Node

- The following template matches any element nodes in the source document’s node tree.
  
  ```xsl
template match="*">
    <apply-templates/>
  </template>
```

- Note that no `select` attribute is given for the `apply-templates` element.

- If no `select` attribute is specified, the XSLT processor locates all of the children of the context node and applies templates to them.
Built-in Text Templates

- This template matches all text nodes and attributes and causes their values to appear in the result document

```xml
<xsl:template match="text()|@*">
    <xsl:value-of select="."/>
</xsl:template>
```
Replace built-in Template for Element Node

- Replace the built-in template for element node with your own template to matches both all attributes and children of the context node in the source document’s node tree.

```xml
<xsl:template match="*">
  <xsl:apply-templates select="@*"/>
  <xsl:apply-templates/>
</xsl:template>
```
Built-in Comment, Processing Instruction, and Namespace Templates

- By default, the comments, processing instructions, and namespace in a source document do not appear in the result document.
- The built-in templates for these nodes are

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

- Templates do nothing, no values are sent to the result document.
Inserting Attribute Values of Literal Result Element

- To insert an attribute value, use the syntax
  
  \[
  \text{<element attribute=\"\{expression\}\"/>}
  
- Where element is the name of the element in the result document, attribute is the name of an attribute associated with the element, and expression is an XPath or XSLT expression that defines the value of the attribute.
Inserting Attribute Values of Literal Result Element

- `<h3>
  <a href="http://www.alcoa.com">Aluminum Company of America</a>
</h3>`

- `<xsl:template match="sName">
    <h3>
      <a href="{../link}">
        <xsl:value-of select="." />
        (<xsl:value-of select="@symbol" />)
      </a>
    </h3>
  </xsl:template>`
Sorting Node Sets

- By default, nodes are processed in document order, by their appearance in the document.
- To specify a different order, XSLT provides the `<xsl:sort>` element.
- This element can be used with either the `<xsl:apply-templates>` or the `<xsl:for-each>` element.
Sorting Node Sets

<xsl:apply-templates select="expression">
  <xsl:sort attributes />
</xsl:apply-templates>

Or

<xsl:for-each select="expression">
  <xsl:sort attributes />
</xsl:for-each>
The syntax of the sort element is

```xml
<xsl:sort select="expression" data-type="type"
    order="type" case-order="type" />
```

Where `select` attribute determines the criteria under which the context node is sorted.

- `data-type` attribute indicates the type of data (text, number, or qname).
- `order` attribute indicates the direction of the sorting (ascending or descending).
- `case-order` attribute indicates how to handle the sorting of uppercase and lowercase letters (upper-first or lower-first).
Sorting Examples

- Sort the stocks by stock name
  
  `<xsl:apply-templates select="portfolio/stock">`
  
  `<xsl:sort select="sName" />`
  
  `</xsl:apply-templates>`

  OR equivalently with the for-each element
  
  `<xsl:for-each select="portfolio/stock">`
  
  `<xsl:sort select="sName" />`
  
  `</xsl:for-each>`
Sorting Order

- If you don’t include the `select` attribute, XSLT processors assume that you want to sort the values of the context node
  
  ```xml
  <xsl:for-each select="portfolio/stock/sName">
    <xsl:sort />
  </xsl:for-each>
  ```

- To sort in descending order, add the `order` attribute to the `sort` element
  
  ```xml
  <xsl:sort select="sName" order="descending" />
  ```
Sorting Number

- If you try to sort the numbers 1 through 100, you end up with the sort order of 1, 10, 100, 11, 12, and so forth, as by default element content is treated as text.
- To sort numerically, you must include the data-type attribute.

  `<xsl:sort select="expression" data-type="number" />`
Sorting more than One Factor

- If you need to sort by more than one factor, you must place one sort element after another.
- For example, to sort the stocks first by category and then by the stock name within each category, you enter the following code into the root templates:

```xml
<xsl:apply-templates select="portfolio/stock">
  <xsl:sort select="category" />
  <xsl:sort select="sName" />
</xsl:apply-templates>
```
Creating Conditional Nodes

- XSLT supports two kinds of conditional elements:
  - `<xsl:if>`
  - `<xsl:choose>`

- To apply a format only if a particular condition is met, use the `<xsl:if>` element

- To test for multiple conditions and display different outcomes, use the `<xsl:choose>` element
Using the if Element

- The syntax for the if element is
  
  ```xml
  <xsl:if test="expression">
    styles
  </xsl:if>
  ```

- Where expression is an XPath expression that is either true or false
  
  ```xml
  <xsl:if test="@symbol = 'AA'">
    <h3><xsl:value-of select="." /></h3>
  </xsl:if>
  ```
Using the if Element

- Be careful when comparing node sets and single values.
- When multiple values are involved, the expression is true if any of the values in the node set satisfy the test condition.
- This means that the if condition
  
  ```xml
  <xsl:if test="/portfolio/stock/sName/@symbol= 'AA' ">
    <xsl:value-of select="/portfolio/stock/sName"/>
  </xsl:if>
  ```

- Displays a stock name even for those stock names whose attribute values are not equal to “AA”, just as long as one stock in the node set has a symbol equal to “AA”.
Using the choose Element

- The syntax for the choose element is

  `<xsl:choose>
      <xsl:when test="expression1">
        styles
      </xsl:when>
      <xsl:when test="expression2">
        styles
      </xsl:when>
      ...
      <xsl:otherwise>
        styles
      </xsl:otherwise>
  </xsl:choose>`
Choose Example

<xsl:choose>
  <xsl:when test="@current &lt; @open">
    <img src="down.gif" />
  </xsl:when>
  <xsl:when test="@current &gt; @open">
    <img src="up.gif" />
  </xsl:when>
  <xsl:otherwise>
    <img src="same.gif" />
  </xsl:otherwise>
</xsl:choose>
## Using Comparison Operators and Functions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=</code></td>
<td>Tests whether two values are equal to each other</td>
<td><code>@symbol = &quot;AA&quot;</code></td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Tests whether two values are unequal</td>
<td><code>@symbol != &quot;AA&quot;</code></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Tests whether one value is less than another</td>
<td><code>day &lt; 5</code></td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Tests whether one value is less than or equal to another</td>
<td><code>day &lt;= 5</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Tests whether one value is greater than another</td>
<td><code>day &gt; 1</code></td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Tests whether one value is greater than or equal to another</td>
<td><code>day &gt;= 1</code></td>
</tr>
<tr>
<td><code>and</code></td>
<td>Combines two expressions, returning a value of true only if both expressions are true</td>
<td><code>@symbol = &quot;AA&quot;</code> and <code>day &gt; 1</code></td>
</tr>
<tr>
<td><code>or</code></td>
<td>Combines two expressions, returning a value of true if either expression is true</td>
<td><code>@symbol = &quot;AA&quot;</code> or <code>@symbol = &quot;UCL&quot;</code></td>
</tr>
<tr>
<td><code>not</code></td>
<td>Negates the value of the expression, changing true to false or false to true</td>
<td><code>not(day &gt;= 1)</code></td>
</tr>
</tbody>
</table>
Using Comparison Operators

- day < 5

- 5 > day

- day > 2 and day < 5

- @symbol="AA" or @symbol="UCL"

- not(@symbol="AA")
Working with Predicates

- Predicates are XPath expressions that test for a condition and create subsets of nodes that fulfill that condition.
- The predicate can also indicate the position of the node in the node tree.
- To select a specific position in the source document, use the `position()` function combined with any XPath expression.
Adding Predicates to the Root Template

Example

```
<html>
<head>
  <title>Stock Information</title>
  <link href="stock.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <div id="datetime">Last updated: </div>
  <xsl:value-of select="portfolio/date" />
  at
  <xsl:value-of select="portfolio/time" />
</body>
</html>
```

Figure 6-39 Display stock information by categories

- industrial stocks
- utility stocks
- transportation stocks
Creating Elements and Attributes

- To create an element, XSLT uses the `<xsl:element>` tag

  `<xsl:element name="name" namespace="uri"
               use-attribute-sets="namelist">
    styles
  </xsl:element>`

- The `name` attribute assigns a name to the element
- The `namespace` attribute provides a namespace
- The `use-attribute-sets` provides a list of attribute-sets
Creating an Element Example

- If you want a result document in which the `sName` element and `symbol` attribute are both enter as elements
  ```xml
  <xsl:template match="sName">
    <xsl:element name="stockName">
      <xsl:value-of select="."/>
    </xsl:element>
    <xsl:element name="stockSymbol">
      <xsl:value-of select="@symbol"/>
    </xsl:element>
  </xsl:template>
  ```

- When this template is applied, the source document code
  ```xml
  <sName symbol="AA">Aluminum Company of America</sName>
  ```

  - is transformed in the result document to
    ```xml
    <stockName>Aluminum Company of America</stockName>
    <stockSymbol>AA</stockSymbol>
    ```
Creating an Element

• To create the <a> element in the result document, use the <xsl:element> tag

```xml
<xsl:template match="name">
  <xsl:element name="a">
    <h3 class="name">
      <xsl:value-of select="." />
      (<xsl:value-of select="@symbol" />)
    </h3>
  </xsl:element>
</xsl:template>
```
Creating an Attribute

- Attributes are created in XSLT by using the `<xsl:attribute>` element

```xml
<xsl:attribute name="name" namespace="uri">
  styles
</xsl:attribute>
```

- The `name` attribute specifies the name of the attribute
- The `namespace` attribute indicates the namespace
Creating an Attribute

- To add the `href` attribute to the `<a>` tag, use the `<xsl:attribute>` element

```xml
<xsl:template match="name">
  <xsl:element name="a">
    <xsl:attribute name="href">
      <xsl:value-of select="../link" />
    </xsl:attribute>
    <h3 class="name">
      <xsl:value-of select="." />
      (\<xsl:value-of select="@symbol" />)
    </h3>
  </xsl:element>
</xsl:template>
```
Creating an Attribute Sets

- Related to XSLT’s attribute element is the attribute-set element, which is used to create sets of attributes to be applied to different element within a style sheet.

- The syntax of the attribute-set element is
  
  `<xsl:attribute-set name="name" use-attribute-sets="name-list">
      <xsl:attribute name="name1">styles</xsl:attribute>
      <xsl:attribute name="name2">styles</xsl:attribute>
      ...
    </xsl:attribute-set>

  - `name` attribute contains the name of the set
  - `name1, name2`, and so on are the names of the individual attributes created within that set
  - You can also refer to other attribute sets by specifying their names in the `name-list` parameter
Attribute Sets Example

```xml
<xsl:attribute-set name="formats">
  <xsl:attribute name="bgcolor">red</xsl:attribute>
  <xsl:attribute name="fgcolor">white</xsl:attribute>
  <xsl:attribute name="align">right</xsl:attribute>
</xsl:attribute-set>

To apply this attribute set to an h1 element, you use the code

```xml
<xsl:element name="h1" use-attribute-sets="formats">
  Hardin Financial
</xsl:element>
```

Resulting in the following HTML code being sent to the result document:

```html
<h1 bgcolor="red" fgcolor="white" align="right">
  Hardin Financial
</h1>
```
Creating Comments and Processing Instructions

- The `<xsl:comment>` element creates the comment
  
  `<xsl:comment>
    Comment text
  </xsl:comment>`

- You can create a processing instruction by using the `<xsl:processing-instruction>` element
  
  `<xsl:processing-instruction name="name"> attributes
  </xsl:processing-instruction>`
Creating Comments and Processing Instructions

- `<xsl:comment>`
  Kevin Summers Stock Portfolio
  `</xsl:comment>`

  Creates the following comment in the result document:
  `<!—Kevin Summers Stock Portfolio-->`

- `<xsl:processing-instruction name="xml-stylesheet">`
  `href="styles.css" type="text/css"`
  `</xsl:processing-instruction>`

  Which generates the following tag in the result document:
  `<?xml-stylesheet href="styles.css" type="text/css"?>`