ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στην Επιστήμη και Τεχνολογία των Υπηρεσιών

Ενότητα 7: XSLT

Χρήστος Νικολάου
Τμήμα Επιστήμης Υπολογιστών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

XML
XSLT
605.444 / 635.444

David Silberberg

Lecture 14

Guts of XSLT

- XSLT operates on programmer-defined templates
- An XSLT document can have 0, 1, or many templates defined
- Templates match against patterns in the incoming XML document
- If a portion (or all) of the XML document is matched by a template
  - The internals of the template are executed
  - Their results are output
- If multiple templates match, there are rules that determine which one to apply.

Example

```
<?xml version=“1.0”?>
<xsl:stylesheet version=“1.0”
    xmlns:xsl=“http://www.w3.org/1999/XSL/Transform”>
     <xsl:template match=“/”>
          <HTML>
          <TITLE>Name document</TITLE>
          <BODY>
          <xsl:for-each select=“//name”>
               <P>Name: <xsl:value-of select=“.”/></P>
          </xsl:for-each>
          </BODY>
          </HTML>
     </xsl:template>
</xsl:stylesheet>
```

Template Characteristics

```
<xsl:template match="/" >
   <HTML>

   <TITLE>Name document</TITLE>
   <BODY>
   <xsl:for-each select="//name">
         <P>Name: <xsl:value-of select="."/></P>
   </xsl:for-each>
   </BODY>
   </HTML>
</xsl:template>
```

Match criteria

Output specification

Output Specifications

- The inside of the <xsl:template …> … </xsl:template> tags describe the output
- All XML tags are output
- All text is output
- Tags that start with **<xsl:** are special operations
  - XSLT process performs operations on the matching template
  - The result of the operations is output
  - <xsl:for-each select="//name"> matches any tag <name> in the document
  - <xsl:value-of select="."/> does something with the matched tag

Example

- Remember *customer.xml*?

```
<?xml version="1.0"?>
<customers>
        <good>
                <name>Jones, Fred</name>
                <name>Li, Sue</name>
                <name>Carnot, John</name>
        </good>
        <bad>
                <name>Tell, William</name>
                <name>Carr, Sam</name>
        </bad>
</customers>
```

Example Output

```
<HTML>
<TITLE>Name document</TITLE>
<BODY>
      <P>Name:Jones, Fred</P>
      <P>Name:Li, Sue</P>
      <P>Name:Carnot, John</P>
      <P>Name:Tell, William</P>
      <P>Name:Carr, Sam</P>
</BODY>
</HTML>
```

XSLT Order of Operations

- XSL processors do not:
  - Read your document in order and apply templates
  - Read the templates in order and apply them to the document
- XSL processors:
  - Look for the template(s) that match the root
  - Apply it (them) to the document
- Common match to root:
  - <xsl:template match="/">
- Another matches to root:
  - <xsl:template match="/[transaction]" >
  - Matches a root with the name <transaction>
- The most specific match is checked first and applied, if possible.

Default Templates

- Highest level match

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

  - Applied if there is no other document root match defined

- Lowest level match

```
<xsl:template match="text()|@*">
        <xsl:value-of select="."/>
</xsl:template>
```

  - Applied to tag if nothing else matches it

Default Templates (cont.)

- Both are active if your XSL document has no templates.
- By default, the XSL processor will output the values of the elements and the attributes.
- When the root is matched, the entire document is "read in" to the processor.
- <xsl:apply-templates/> informs the processor to apply other templates to what was "read in."
- Any match, such as <xsl:template match="/">, overrides the default.

Recall sales.xml

```
<?xml version=“1.0”?>
<transaction>
	<salesman>
		<lastname>Smith</lastname>
		<firstname>Fred</firstname>
		<mi>P</mi>
	</salesman>
	<customer>
		<name>Frank Thomas</name>
		<address>10 Maple Street</address>
		<city>Columbia</city>
		<state>MD</state>
		<zip>22222</zip>
	</customer>
```

Recall sales.xml - (cont.)

```
        <date>
                <year>2001</year>
                <month>12</month>
                <day>22</day>
        </date>
        <item>
                <name>Cat Chow</name>
                <size>30</size>
                <qty>2</qty>
                <unitprice>9.95</unitprice>
        </item>
</transaction>
```

Context Nodes from Templates

- Whatever is matched becomes the context
- Children of <salesman>:

```
<xsl:template match="/transaction/salesman">
        <xsl:value-of select="*">
</xsl:template>
```

- All parent tags that contain <name>:

```
<xsl:template match="//name">
        <xsl:value-of select="parent::*">
</xsl:template>
```

  – Yields both <customer> and <item>

Circumventing the Context Node

- You can still have access to nodes outside the context

```
<xsl:template match=“/transaction/customer”>
    <xsl:value-of select=“/transaction/salesman/lastname”/>
</xsl:template>
```

- You can also use axis specifications

```
<xsl:template match=“/transaction/customer”>
    <xsl:value-of select=“preceding-
                        sibling::salesman/lastname”/>
</xsl:template>
```

The Basics - Stylesheets

- <xsl:stylesheet>
- This defines the stylesheet for XSL processors
- Must follow <?xml …?> statement
- In <xsl:stylesheet …>
  - Specify version
  - Specify namespace
- Follow this by template list

Stylesheet Example

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

Simplifying Stylesheets for HTML

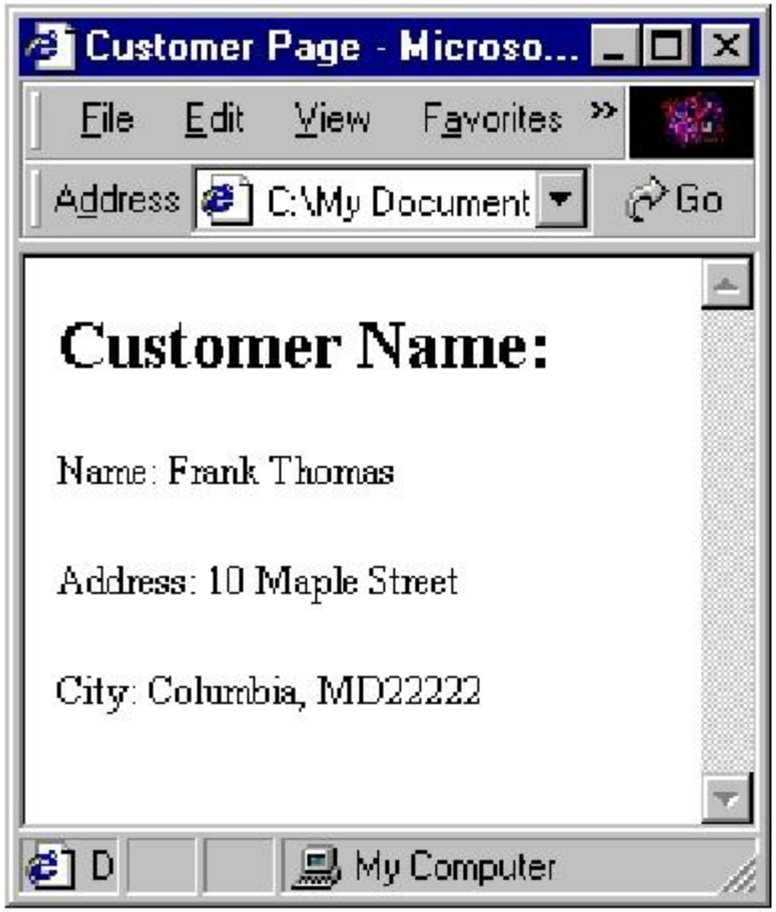
- Regular stylesheet that prints out customer information:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
     <HTML><TITLE>Customer Page</TITLE>
        <BODY>
        <H1>Customer Name:</H1>
        <P>Name: <xsl:value-of select="/transaction/customer/name"/></P>
        <P>Address: <xsl:value-of select="/transaction/customer/address"/></P>
        <P>City: <xsl:value-of select="/transaction/customer/city"/>,
           <xsl:value-of select="/transaction/customer/state"/>
           <xsl:value-of select="/transaction/customer/zip"/> </P>
        </BODY>
     </HTML>
   </xsl:template>
</xsl:stylesheet>
```

Simplifying Version of Stylesheet

```
<HTML xsl:version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<TITLE>Customer Page</TITLE>
<BODY>
  <H1>Customer Name:</H1>
  <P>Name: <xsl:value-of select="/transaction/customer/name"/></P>
  <P>Address: <xsl:value-of select="/transaction/customer/address"/></P>
  <P>City: <xsl:value-of select="/transaction/customer/city"/>,
    <xsl:value-of select="/transaction/customer/state"/>
    <xsl:value-of select="/transaction/customer/zip"/> </P>
</BODY>
</HTML>
```

Simplifying Version Display



Customer Name:

Name: Frank Thomas

Address: 10 Maple Street

City: Columbia, MD22222

Template Attributes

- <xsl:template match=“XPath Expression”
  name=“template name”
  priority=“number”
  mode=“mode name”>
- match
  - Matches XPath pattern
  - Not actual path!
  - Example: //name - matches all <name> nodes
- name
  - Lets you name the template
  - Can specifically call the template by name

Template Attributes (cont.)

- priority
  - There is normally a set of rules by which the XSLT processor uses to select template
  - On occasion, two or more might conflict
  - This provides a way to prioritize the templates
- mode
  - Allows multiple templates to be applied to same nodes
  - Will process the same templates in different “modes”
  - Will describe in more detail later

Apply Templates

- <xsl:apply-templates select="XPath Expression" mode="mode name">
- Used to call other templates
- Sets the context to the path defined in the XPath Expression
  - If not specified, the current context node is used
- The mode determines which templates to apply (more later)
- Remember the default

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

  - XPath or mode must be different if you want different results

Value Of

- <xsl:value-of select="XPath Expression"
  disable-output-escaping="yes or no" />
- Searches the context node for the path specified in XPath
- Then, inserts data from the context node into the output
  – <xsl:value-of select="name"/>
    - Inserts text value of name element
  – <xsl:value-of select="name/@id"/>
    - Inserts text value of id attribute of name element

Example Apply Templates to customer.xsl

ApplyTemplate.xsl:

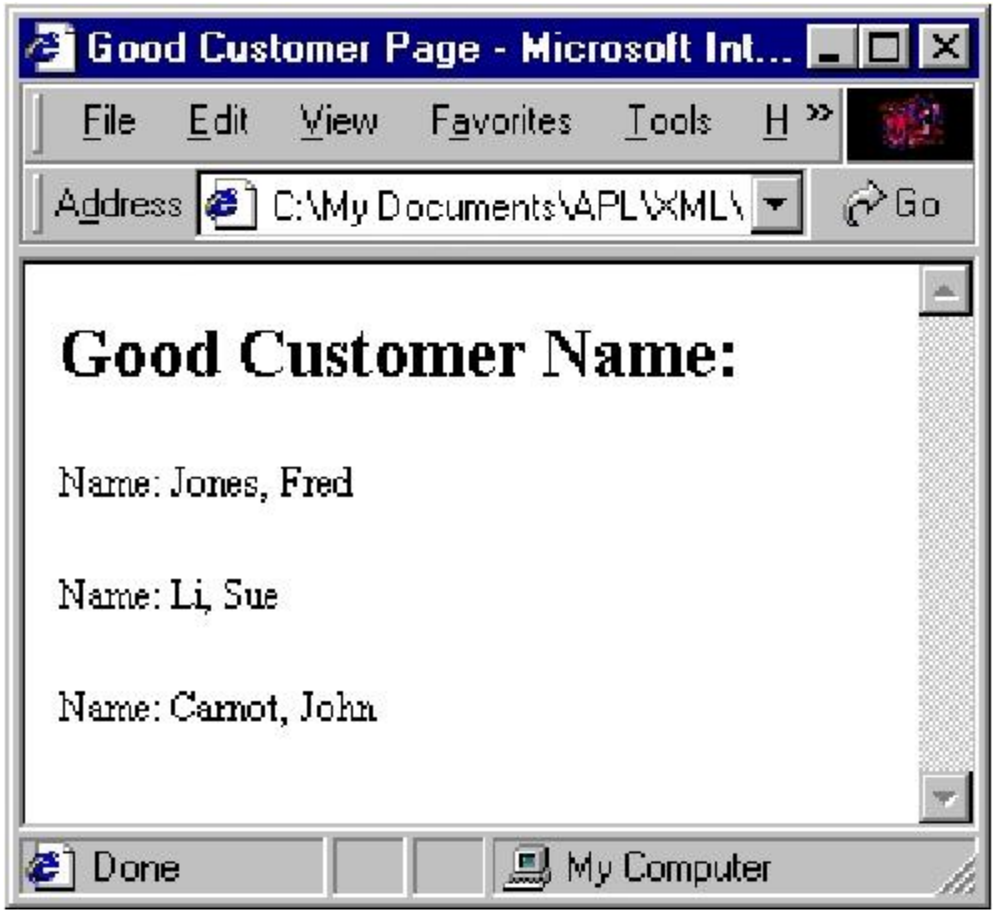
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
     <HTML><TITLE>Good Customer Page</TITLE>
        <BODY>
        <H1>Good Customer Name:</H1>
        <xsl:apply-templates select="/customers/good" />
        </BODY>
     </HTML>
   </xsl:template>

   <xsl:template match=”name">
        <P>Name: <xsl:value-of select=“."/></P>
   </xsl:template>
</xsl:stylesheet>
```

Apply Templates Output

Running XSLT processor yields:

```
<HTML>
<TITLE>Good Customer Page</TITLE>
<BODY>
<H1>Good Customer Name:</H1>
        <P>Name: Jones, Fred</P>
        <P>Name: Li, Sue</P>
        <P>Name: Carnot, John</P>
</BODY>
</HTML>
```

Apply Templates Display


Good Customer Page - Microsoft Int...
File Edit View Favorites Tools H
Address C:\My Documents\APL\XML\
Go
Done
My Computer

Good Customer Name:

Name: Jones, Fred

Name: Li, Sue

Name: Carnot, John

More on Value Of Command

- disable-output-escaping
  - Enables XSLT to output “**&**” , “**<**“ , and “ “ instead of **&amp;** , **&lt;** , and **&nbsp;** with the **“yes”** option
  - With the **“no”** option, it will just produce **&amp;** , **&lt;** , and **&nbsp;**
- This is useful for many reasons
- Will discuss this later with the <xsl:text …> command

Output Command

- <xsl:output method="xml or html or text"

  version="version"

  encoding="encoding"

  omit-xml-declaration="yes or no"

  standalone="yes or no"

  cdata-section-elements="CDATA sections"

  indent="yes or no"/>

- Must be top-level element below <xsl:stylesheet …>
- Tells XSL processor how to format output

Rules for Output

- **method**
  - Specifies whether XML, HTML, or Text is produced
  - XSL processor can support other types of output if desired
- If <xml:output …> not specified, method is:
  - HTML if the root element of the result tree is <HTML>
  - XML otherwise
- XSL processor can do special things based on the method
  - Change <BR/> for XML-compliant documents to <BR> for HTML documents
  - etc.

More on Output

- **version**, **encoding** and **standalone**
  - Used for XML processing
  - Used to create the result tree of XML document
- **version** is the XML version
- **encoding** is the *suggested* encoding value
  - XSL processor may use it
  - However, it can ignore it
  - For example, XSL processor may change a “windows-1252” suggestion to a “utf-8” coding
    - Maybe XSL processor does not handle “windows-1252”
    - Maybe XSL processor may detect object platform
- **standalone=“yes”** indicates that there are no external references

Even More on Output

- **omit-xml-declaration**
  - As it states: omits XML declaration statement from the output
  - Might be for a number of reasons
  - Output document could be:
    - Included in another document
    - An XML-stylesheet
  - Default is “no” if the method is XML
    - In this case, XML declaration will appear in output document
- **indent**
  - “no” specifies that no indenting will occur - tags will be run on in some XSL processors
  - “yes” specifies that that pretty printing will occur

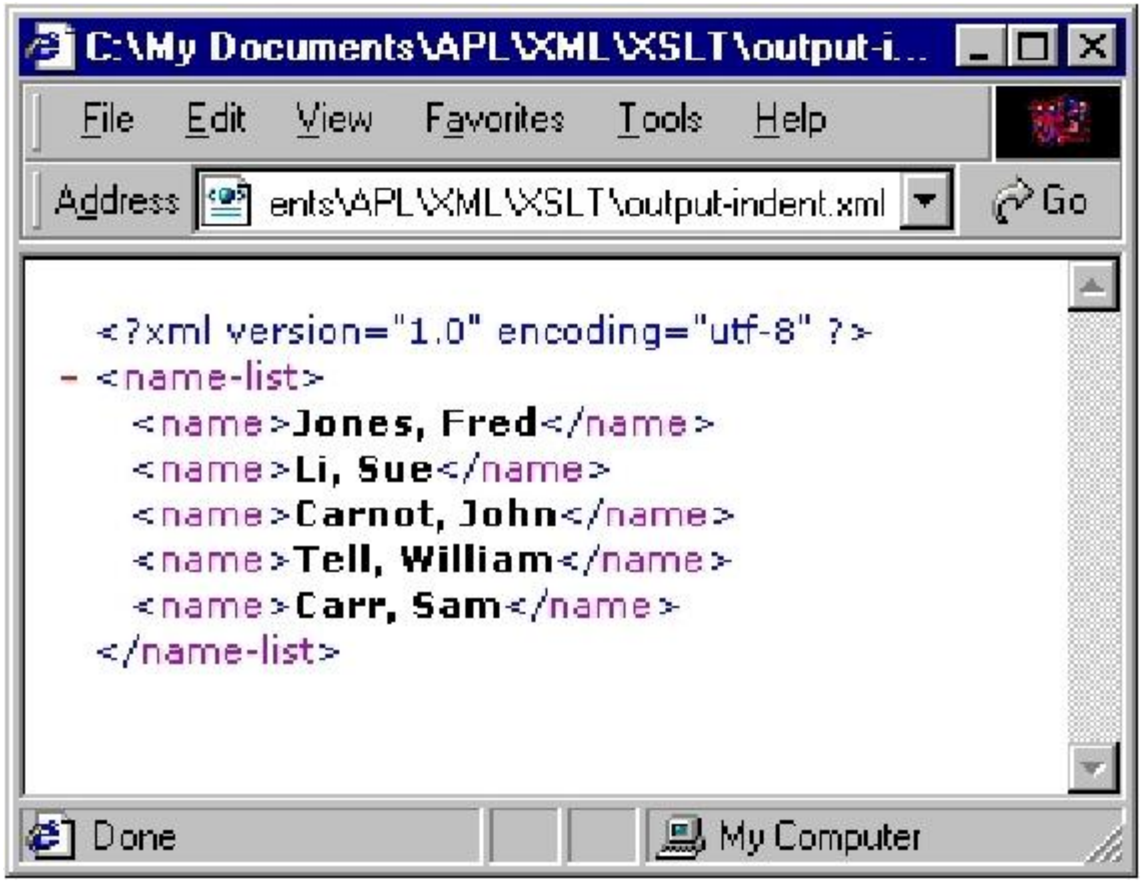
Indent Example

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

   <xsl:template match="/">
     <name-list>
     <xsl:apply-templates/>
     </name-list>
   </xsl:template>

   <xsl:template match="//name">
     <name><xsl:value-of select="."/></name>
   </xsl:template>
</xsl:stylesheet>
```

Indent Output

```
<?xml version="1.0" encoding="utf-8"?>
<name-list>

    <name>Jones, Fred</name>
    <name>Li, Sue</name>
    <name>Carnot, John</name>



    <name>Tell, William</name>
    <name>Carr, Sam</name>

</name-list>
```

Indent Display


C:\My Documents\APL\XML\XSLT\output-i...
File Edit View Favorites Tools Help
Address ents\APL\XML\XSLT\output-indent.xml
Go
Done
My Computer

```
  <?xml version="1.0" encoding="utf-8" ?>
- <name-list>
    <name>Jones, Fred</name>
    <name>Li, Sue</name>
    <name>Carnot, John</name>
    <name>Tell, William</name>
    <name>Carr, Sam</name>
  </name-list>
```

Element

- <xsl:element name=“element name”
  use-attribute-sets=“attribute set names”>
- This enables elements to be created dynamically.
- This is in case the element name in not known à priori
- Using the previous example, instead of:

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

the following may be specified:

```
<xsl:template match="//name">
   <xsl:element name=“name”><xsl:value-of select="."/></xsl:element>
</xsl:template>
```

Element (cont.)

- This is not so interesting.
- However, the following may be done:

```
<xsl:template match="//name">
   <xsl:element name=“{.}”><xsl:value-of select="."/></xsl:element>
</xsl:template>
```

- This builds the element name dynamically.
- Any XPath Expression can be placed in the parentheses.

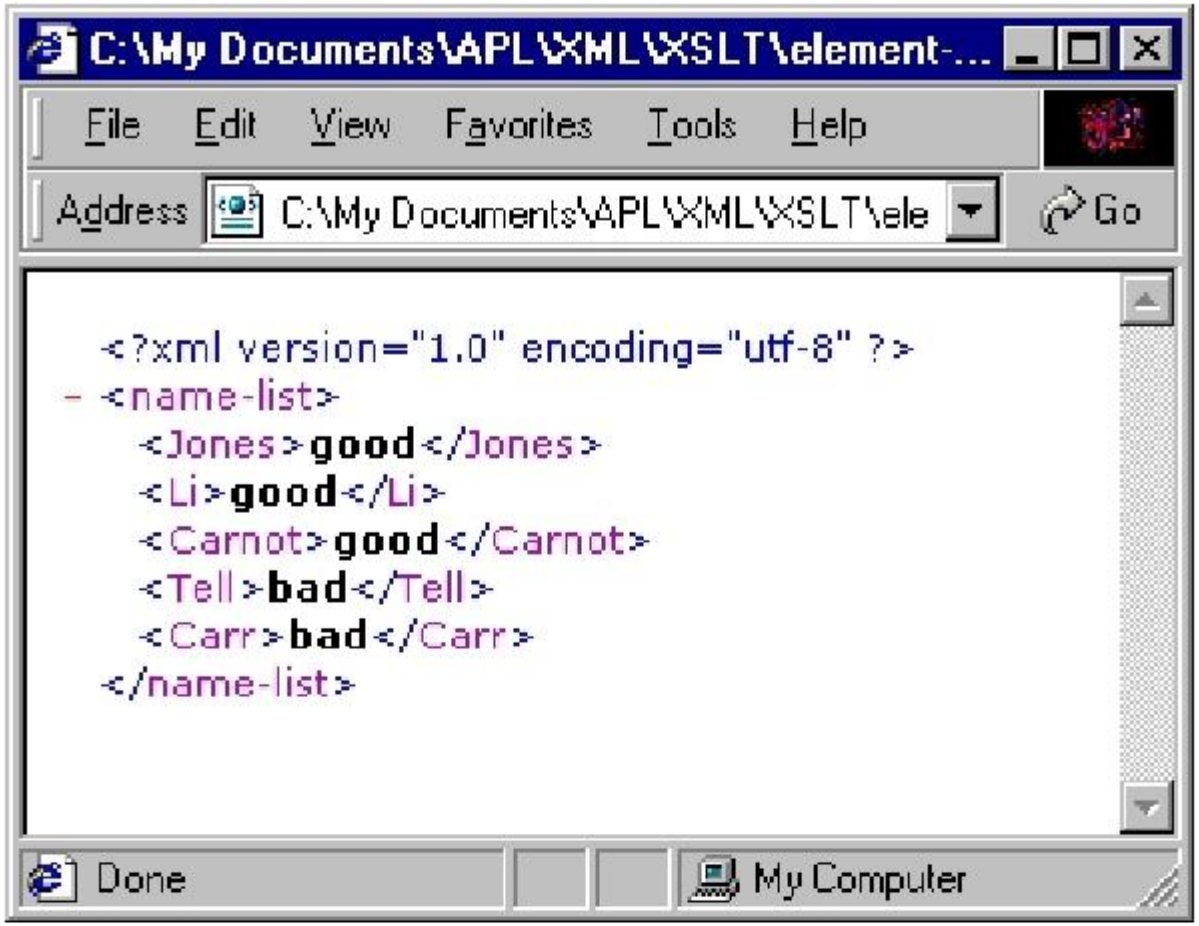
Element Example

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <name-list>
      <xsl:apply-templates/>
      </name-list>
    </xsl:template>
    <xsl:template match="//name">
      <xsl:element name="{substring-before(., ',')}">
                <xsl:value-of select="name(..)"/>
      </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

Element Output

```
<?xml version="1.0" encoding="utf-8"?>
<name-list>

    <Jones>good</Jones>
    <Li>good</Li>
    <Carnot>good</Carnot>


    <Tell>bad</Tell>
    <Carr>bad</Carr>

</name-list>
```

Element Display


C:\My Documents\APL\XML\XSLT\element-...
File Edit View Favorites Tools Help
Address C:\My Documents\APL\XML\XSLT\ele
Go
Done
My Computer

```
  <?xml version="1.0" encoding="utf-8" ?>
- <name-list>
    <Jones>good</Jones>
    <Li>good</Li>
    <Carnot>good</Carnot>
    <Tell>bad</Tell>
    <Carr>bad</Carr>
  </name-list>
```

Attributes

- <xsl:attribute name=“attribute name”>
- Creates attributes and their values

```
<xsl:template match=“//name”>
    <name>
        <xsl:attribute name=“last”>
            <xsl:value-of  select=“substring-before(., ',')”/>
        </xsl:attribute>
        <xsl:attribute name=“first”>
            <xsl:value-of  select=“substring-after(., ',')”/>
        </xsl:attribute>
    </name>
</xsl:template>
```

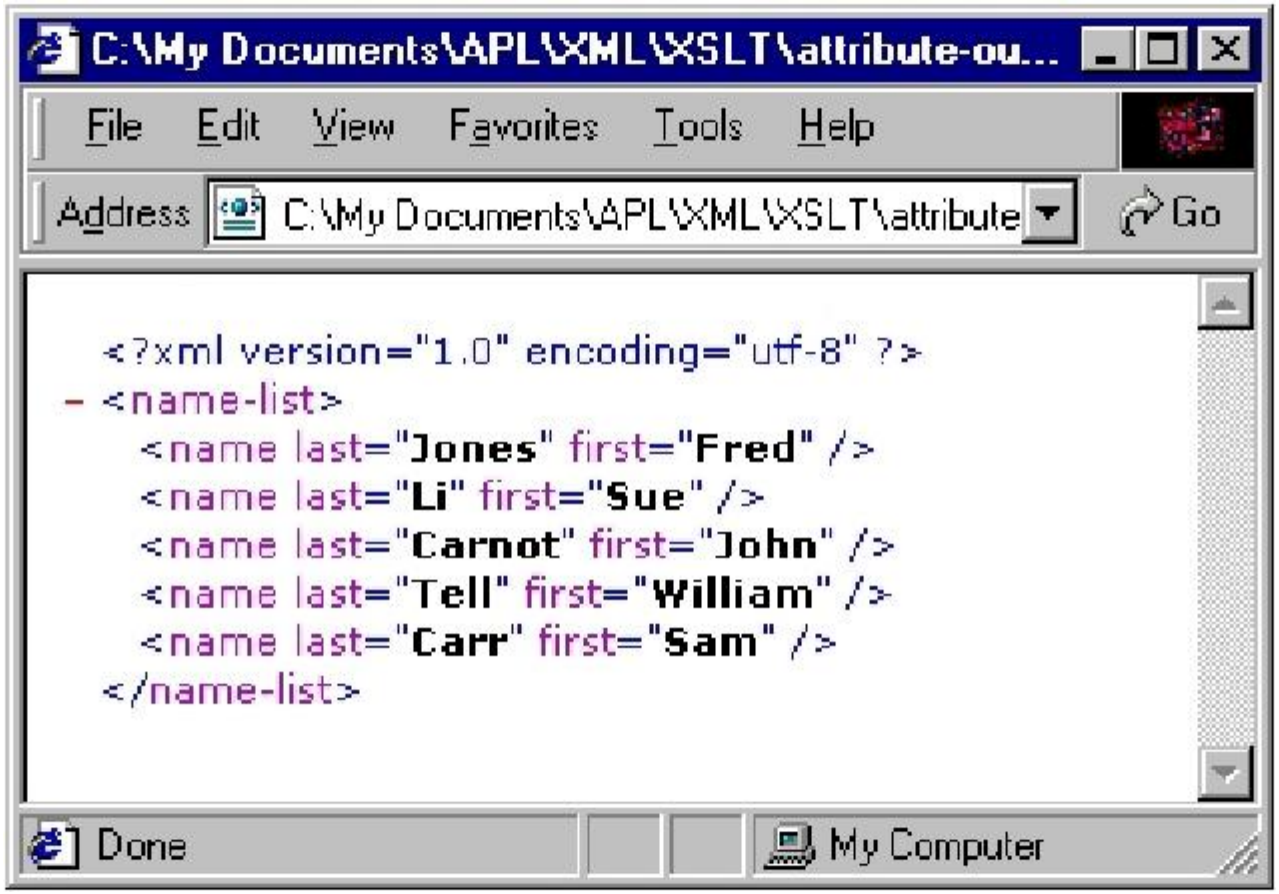
Attributes Example

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <name-list>
      <xsl:apply-templates/>
      </name-list>
   </xsl:template>
   <xsl:template match="//name">
      <name>
            <xsl:attribute name="last"> <xsl:value-of  select="substring-before(., ',')"/></xsl:attribute>
            <xsl:attribute name="first"><xsl:value-of  select="substring-after(., ',')"/></xsl:attribute>
      </name>
   </xsl:template>
</xsl:stylesheet>
```

Attributes Output

```
<?xml version="1.0" encoding="utf-8"?>
<name-list>

    <name last="Jones" first=" Fred"/>
    <name last="Li" first=" Sue"/>
    <name last="Carnot" first=" John"/>


    <name last="Tell" first=" William"/>
    <name last="Carr" first=" Sam"/>

</name-list>
```

Attribute Output Display


C:\My Documents\APL\XML\XSLT\attribute-ou...
File Edit View Favorites Tools Help
Address C:\My Documents\APL\XML\XSLT\attribute
Go
Done
My Computer

```
  <?xml version="1.0" encoding="utf-8" ?>
- <name-list>
    <name last="Jones" first="Fred" />
    <name last="Li" first="Sue" />
    <name last="Carnot" first="John" />
    <name last="Tell" first="William" />
    <name last="Carr" first="Sam" />
  </name-list>
```

Attribute Sets

- <xsl:attribute-set   name=“name of this attribute set”

  use-attribute-sets=“attr set names”>
- Sets of attributes can be related in an attribute set
- We can define a set of attributes that always give first and last names

```
<xsl:attribute-set name=“FirstAndLast”>
    <xsl:attribute name="last"> <xsl:value-of  select="substring-before(.,',')"/>
    </xsl:attribute>
    <xsl:attribute name="first"><xsl:value-of  select="substring-after(.,',')"/>
    </xsl:attribute>
</xsl:attribute-set>
```

- Attribute sets can also use other attribute sets!

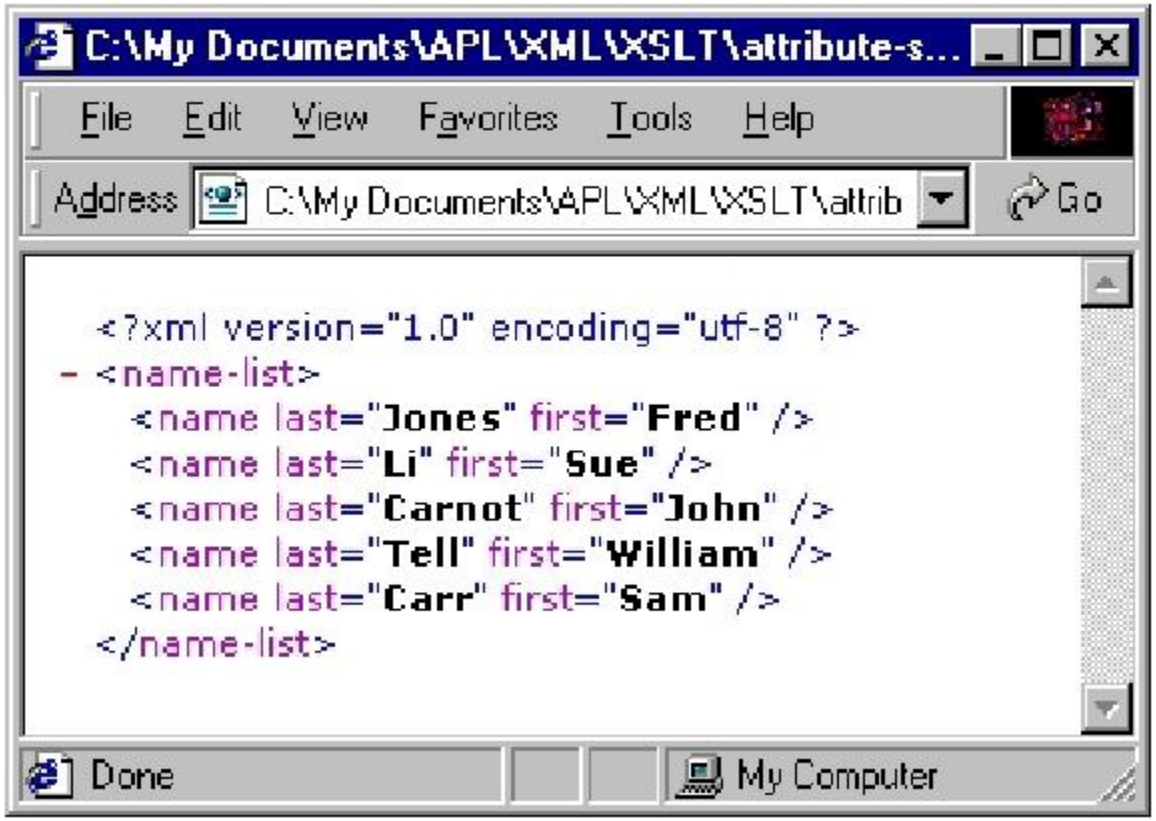
Attribute Sets Example

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
      <name-list>
      <xsl:apply-templates/>
      </name-list>
   </xsl:template>
   <xsl:template match="//name">
      <xsl:element name="{name()}" use-attribute-sets="FirstAndLast"/>
   </xsl:template>
   <xsl:attribute-set name="FirstAndLast">
      <xsl:attribute name="last"><xsl:value-of select="substring-before(.,',')"/></xsl:attribute>
      <xsl:attribute name="first"><xsl:value-of select="substring-after(.,',')"/></xsl:attribute>
   </xsl:attribute-set>
</xsl:stylesheet>
```

Attribute Sets Output

```
<?xml version="1.0" encoding="utf-8"?>
<name-list>

    <name last="Jones" first=" Fred"/>
    <name last="Li" first=" Sue"/>
    <name last="Carnot" first=" John"/>


    <name last="Tell" first=" William"/>
    <name last="Carr" first=" Sam"/>

</name-list>
```

Attribute Sets Display


C:\My Documents\APL\XML\XSLT\attribute-s...
File Edit View Favorites Tools Help
Address C:\My Documents\APL\XML\XSLT\attrib
Go
Done
My Computer

```
  <?xml version="1.0" encoding="utf-8" ?>
- <name-list>
    <name last="Jones" first="Fred" />
    <name last="Li" first="Sue" />
    <name last="Carnot" first="John" />
    <name last="Tell" first="William" />
    <name last="Carr" first="Sam" />
  </name-list>
```

Text

- <xsl:text>
- Inserts PCDATA text into the output
- Most of the time, XSL processors output text anyway
- However, it is useful in two cases
  - If white spaces are to be inserted into output
  - If output escaping is to be disabled
- Suppose the names are to be transposed

```
<xsl:template match=“name”>
        <P><xsl:value-of select=“substring-after(., ‘,’)”/>
                <xsl value-of select=“substring-before(., ‘,’)”/> </P>
</xsl:template>
```

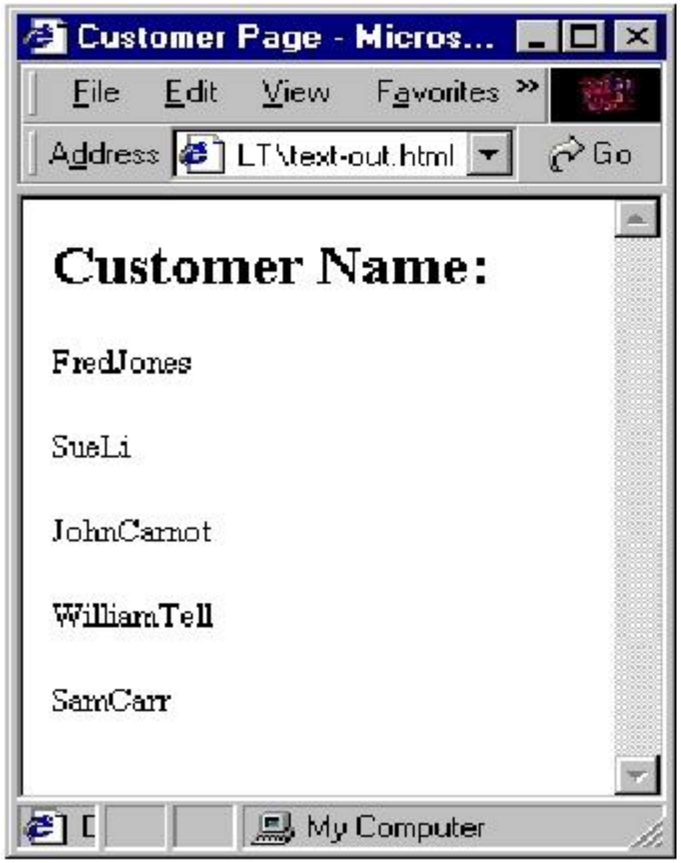
Text Example

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <HTML><TITLE>Customer Page</TITLE>
         <BODY>
               <H1>Customer Name:</H1>
               <xsl:apply-templates/>
         </BODY>
      </HTML>
   </xsl:template>
   <xsl:template match="//name">
      <P><xsl:value-of select="substring-after(., ',')"/>
        <xsl:value-of select="substring-before(., ',')"/> </P>
   </xsl:template>
</xsl:stylesheet>
```

Text Output

```
<HTML>
<TITLE>Customer Page</TITLE>
<BODY>
<H1>Customer Name:</H1>


      <P> FredJones</P>
      <P> SueLi</P>
      <P> JohnCarnot</P>



      <P> WilliamTell</P>
      <P> SamCarr</P>

</BODY>
</HTML>
```

Text Display



Customer Name:

FredJones

SueLi

JohnCarnot

WilliamTell

SamCarr

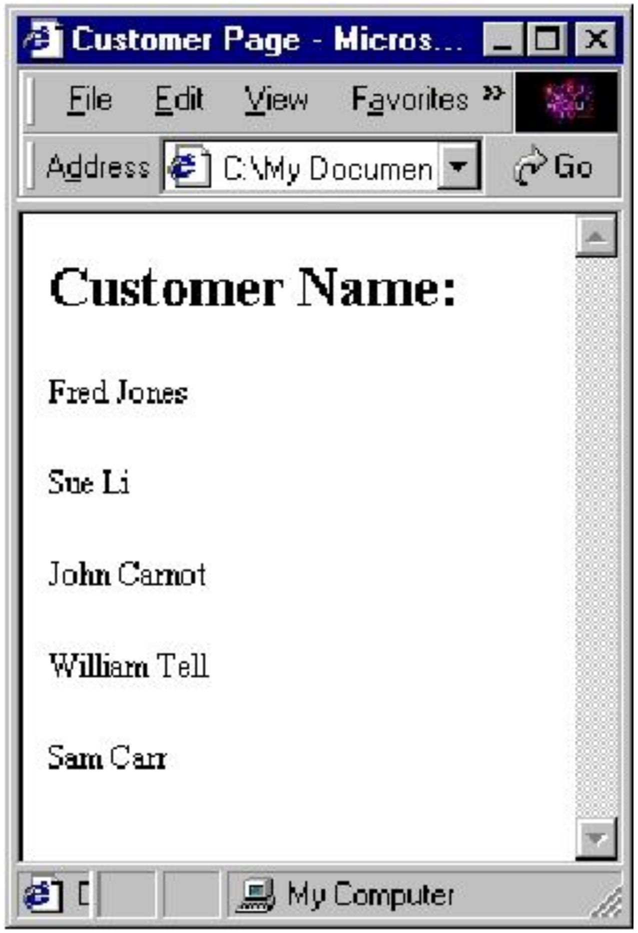
Corrected Text Example

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <HTML><TITLE>Customer Page</TITLE>
         <BODY>
                <H1>Customer Name:</H1>
                <xsl:apply-templates/>
         </BODY>
      </HTML>
   </xsl:template>
   <xsl:template match="//name">
      <P><xsl:value-of select="substring-after(., ',')"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="substring-before(., ',')"/> </P>
   </xsl:template>
</xsl:stylesheet>
```

Corrected Text Output

```
<HTML>
<TITLE>Customer Page</TITLE>
<BODY>
<H1>Customer Name:</H1>

    <P> Fred Jones</P>
    <P> Sue Li</P>
    <P> John Carnot</P>



    <P> William Tell</P>
    <P> Sam Carr</P>


</BODY>
</HTML>
```

Corrected Text Display



Customer Name:

Fred Jones

Sue Li

John Carnot

William Tell

Sam Carr

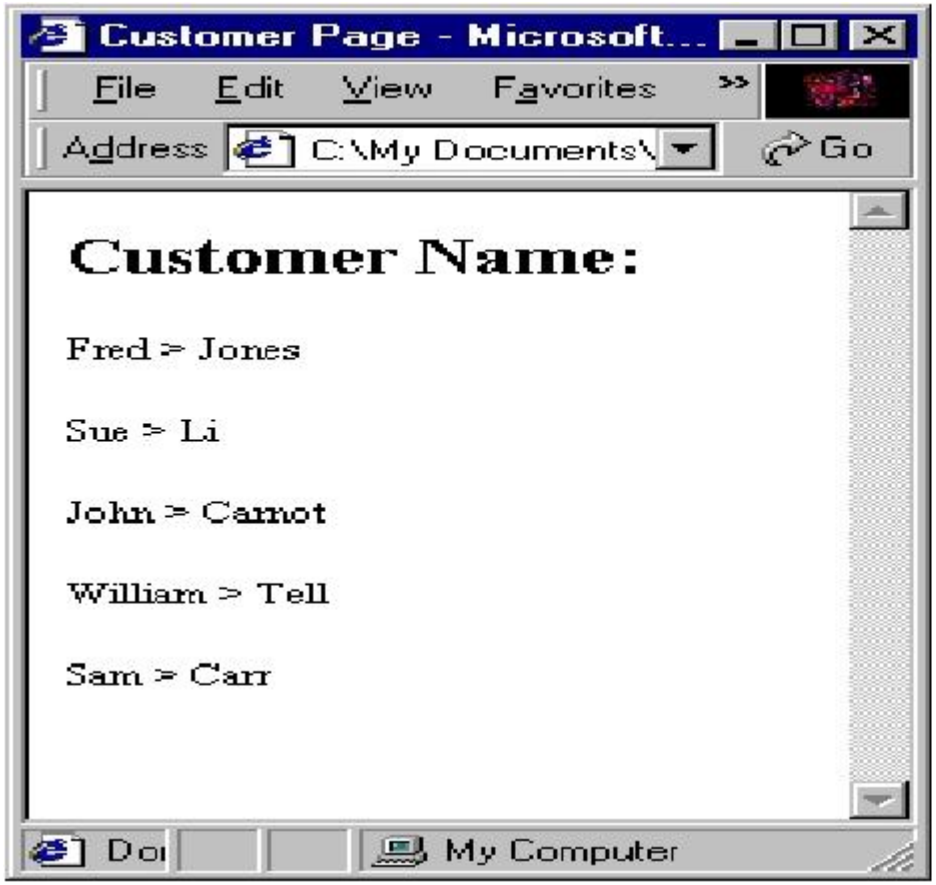
Text Example for Special Characters

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <HTML><TITLE>Customer Page</TITLE>
         <BODY>
                <H1>Customer Name:</H1>
                <xsl:apply-templates/>
         </BODY>
      </HTML>
   </xsl:template>
   <xsl:template match="//name">
      <P><xsl:value-of select="substring-after(., ',')"/>
        <xsl:text disable-output-escaping="yes"> &gt; </xsl:text>
        <xsl:value-of select="substring-before(., ',')"/> </P>
   </xsl:template>
</xsl:stylesheet>
```

Text Output for Special Characters

```
<HTML>
<TITLE>Customer Page</TITLE>
<BODY>
<H1>Customer Name:</H1>

    <P> Fred > Jones</P>
    <P> Sue > Li</P>
    <P> John > Carnot</P>



    <P> William > Tell</P>
    <P> Sam > Carr</P>

</BODY>
</HTML>
```

Special Character Display



Customer Name:

Fred > Jones

Sue > Li

John > Carnot

William > Tell

Sam > Carr

Τέλος Ενότητας