

Section Preview

- ❑ **Introduction to eXtensible Stylesheet Language for Transformation (XSLT)**
 - ◆ **XML and Style**
 - ◆ **XML and Validation**
 - ◆ **XML and Transformation**

XML and Style

- ❑ **When you learn about XML, you see how we can use style sheets to direct the formatting of an XML document**
 - ◆ **Style information can be embedded in a document**
 - ◆ **External style information can be referenced in a document's header**
 - x Point to a style sheet (CSS type is what we looked at)
 - x Document uses style attributes defined in the style sheet as part of the document mark up

- ❑ **The XML processor (typically a browser, but could be any appropriate application) formats output based on**
 - ◆ **The content of the document**
 - ◆ **The details of the stylesheet**
 - ◆ **And the characteristics of the target output device (screen, printer, braille output, etc.)**

XML and Validation

- ❑ In addition, you can use DTD' s or Schemas to validate the contents of an XML document
 - ◆ DTD information can be embedded in a document
 - ◆ External validation information can be referenced in a document' s header
 - x Point to DTD file(s)
 - x Point to Schema file(s)
- ❑ The XML processor examines the input document and tests the content' s validity based on the implied tests of the DTD or Schema
 - ◆ How to handle errors is left to the application

XML and Transformation

- ❑ In this paper we examine a tool designed to transform an XML document into another XML document: **eXtensible Stylesheet Language for Transformation (XSLT)**

- ❑ In a very confusing arrangement, **XSLT is actually a subset of the eXtensible Stylesheet Language (XSL)**
 - ◆ **Sometimes you' ll see the notation XSL/T to emphasize you are doing transformations using the XSL language**

 - ◆ **An XSL document can perform two major classes of operations: Transformation and Formatting**

 - ◆ **You can do just Transformation, just Formatting, or both**
 - x If you do both, the Transformation work is always done first, so we focus on that in this section

 - x We ignore Formatting techniques in this paper

XML and Transformation, 2

- **In the same approach as style sheets and schema, to use transformation you construct a well-formed XML document that uses the tags and attributes and syntax defined for XSL/T**

- ◆ **Then, you can process this in one of two ways:**

- ✗ In the document to be transformed, point to the XSL/T document via a processing instruction
- ✗ Invoke an XSL processor that allows you to point to the base document and the XSL/T document

Notes

- ◆ **The standards have this cryptic comment:**

- ✗ "The mechanism chosen for this version of the specification is not a constraint on the additional mechanisms planned for future versions. There is no expectation that these will use processing instructions; indeed they may not include the linking information in the source document."

- This is why I mention the second alternative above

- ◆ **The actual document processing may be done server-side (before transmission) or client-side (by the browser or other user agent)**

XML and Transformation, 3

□ To understand the terminology used in the standard, we cover some basic terms

◆ An XML document is considered to be a tree (the original document is the "source tree", the output document is the "result tree") [actually, each is a sort of inverted tree]

◆ A document tree is composed of nodes

x The root node contains everything in the document

➤ It is not the document root element, it is an abstract point above the root element and contains all nodes in the document

x An element node is a document element; it may contain other nodes; the root node is a special case of element node: it is the starting point

x An attribute node comes from a document element' s attribute

x A text node is a string of characters with no intervening nodes

x A comment node comes from a document comment

x A processing instruction node comes from document PI' s

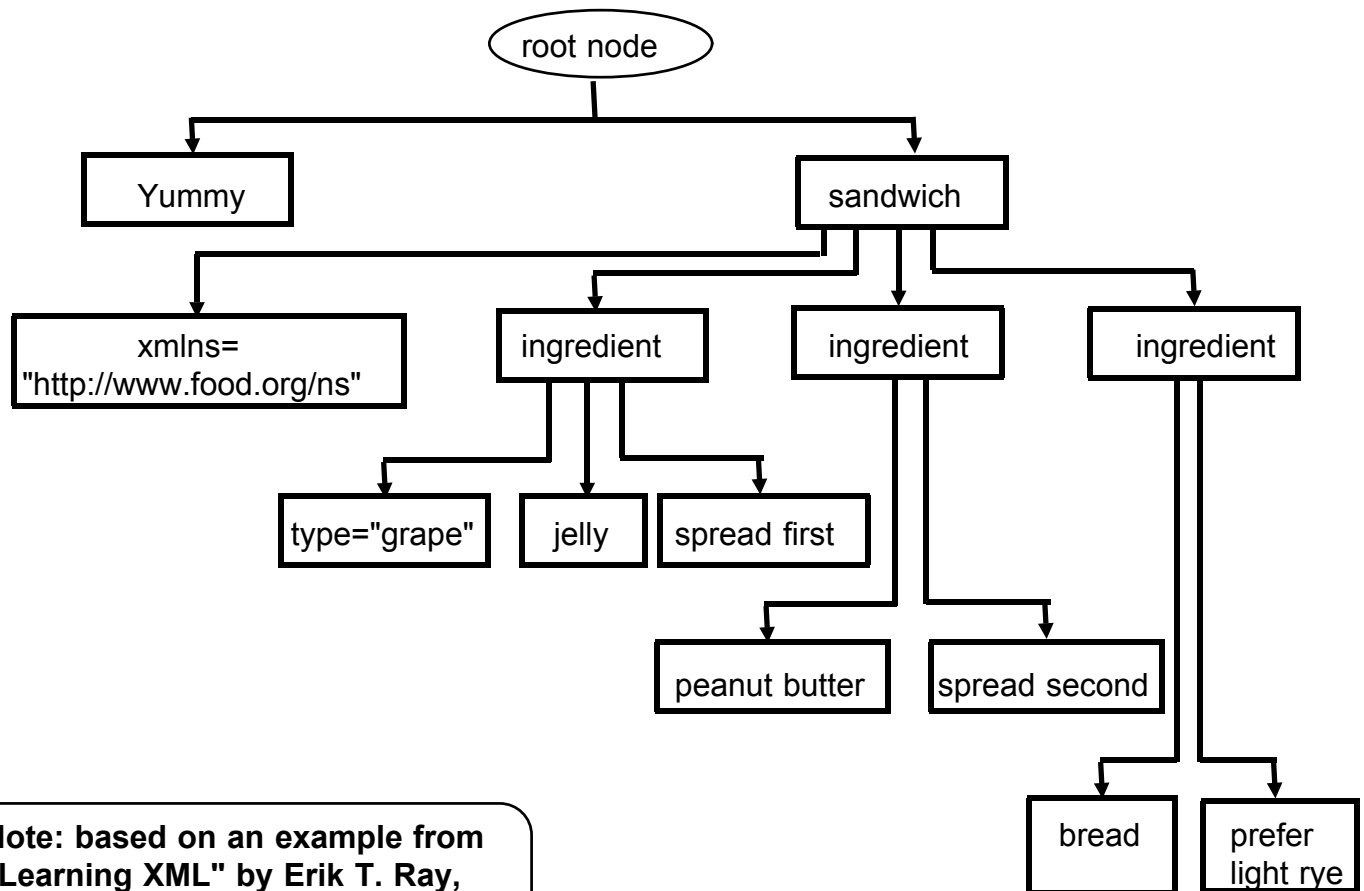
x A namespace node comes from xmlns declarations

XML and Transformation, 4

Here's a simple example that contains each kind of node:

<pre><?xml version=' 1.0' ?> <!-- Yummy! --> <sandwich xmlns=:http://www.food.org/ns"> <ingredient type="grape"><?knife spread first?> jelly</ingredient> <ingredient><?knife spread second?> peanut butter</ingredient> <ingredient>bread <!-- prefer light rye --></ingredient> </sandwich></pre>	<ul style="list-style-type: none">* PI* comment* namespace* element, attribute, PI, text* element, PI, text* element, text, comment
---	--

□ Here's a diagram of the nodes to demonstrate the "tree-ness" of this document in a visual fashion...



Note: based on an example from "Learning XML" by Erik T. Ray, O' Reilly Books, 2001

XML and Transformation, 5

- ❑ So you can cover the whole document by starting at the root and following the nodes, or branches, down and / or sideways
 - ◆ Get the idea of child / parent / sibling relationships
 - ◆ **Note that attribute, text, comment, processing instruction, and namespace nodes may not have child branches - these are called leaf branches or leaf nodes**
 - ◆ Element nodes with no child branches are also leaf branches

- ❑ The XSLT language allows you specify how you want to process the branches
 - ◆ Which nodes to keep, what data to keep from each node
 - ◆ Where to insert new nodes
 - ◆ How to order output nodes

XML and Transformation, 6

- **The big picture is: given a source tree and an XSLT document to run against it, you get a result tree**
 - ◆ **The objective of XSLT processing is to produce a result tree that is a well-formed XML document in itself**
 - ◆ **Although it's possible to produce other kinds of output**
 - x In fact it's common to produce HTML or XHTML output

- **The actual format of the output depends on several factors**
 - ◆ **What you request in any `xsl:output` processing instruction (discussed shortly)**
 - ◆ **The actual transformation instructions in your XSLT document**
 - ◆ **The name you assign the result tree (output file)**
 - ◆ **The specifics of your XSLT processor**
 - x The examples in this section were tested with various settings using the XMLSpy product

XML and Transformation, 7

- ❑ **An XSLT document is itself an XML document so it needs a document declaration, and a PI to point to the namespace that the XSLT tags refer to, so start with something like this:**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

<!-- body of XSLT document goes here -->

</xsl:stylesheet>
```

- ◆ **The body of the XSLT document contains the rules for the transformation you want to make**
-
- ❑ **Note: a synonym for "stylesheet" here is "transform"; these terms can be used interchangeably in the XSL element tag**

XML and Transformation, 8

- ❑ The `xsl:output` processing instruction is an optional statement

Syntax

```
<xsl:output method="{xml | html | text | other}" />
```

- ◆ Provides direction to the XSL processor regarding the method to be used for outputting the result tree
- ◆ There are other attributes available, but none of interest for this introductory session
- ◆ The default is either `html` or `xml`, depending on the output file name and the structure of the result tree
- ◆ The *other* option is provided for XSL processing products created to produce special outputs

XML and Transformation, 9

- ❑ The rules for making a transformation are called template rules or just templates

- ◆ A template rule must

- ✗ Identify which node(s) it applies to
- ✗ Specify what action to take

- ❑ The general syntax for a template rule is...

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

```
    <!-- actions to take when node matches the pattern -->
```

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

- ❑ There are a couple of other options but they don't concern us here in the beginning stages

XML and Transformation, 10

- There are many ways to specify match patterns, here are just a few:

/ - matches the root node

*** - matches any element node

text() - matches any text node

name - matches all elements named *name*

name₁ | name₂ [...] - matches each of the named elements
(in other words, a list of names separated by vertical bars)

ID(*id_value*) - match any element with an ID value of *id_value*

For example

```
<xsl:template match="/">
  <!-- action -->
</xsl:template>
```

```
<xsl:template match="*">
  <!-- action -->
</xsl:template>
```

```
<xsl:template match="ingredient">
  <!-- action -->
</xsl:template>
```

```
<xsl:template match="title | artist">
  <!-- action -->
</xsl:template>
```

XML and Transformation, 11

- The default action is to output the content of the selected node
 - ◆ You may also code literals and these will be output also

For example, if your template is:

```
<xsl:template match="/">
  <html>
    <body>
      <h1>Header</h1>
    </body>
  </html>
</xsl:template>
```

Then your output will contain just the literals:

```
<html>
  <body>
    <h1>Header</h1>
  </body>
</html>
```

- ◆ Note that if your output contains any markup, it should be well-formed markup

XML and Transformation, 12

- If you just want to copy a file, you can specify:

```
<xsl:template match="/">  
  <xsl:copy-of select="*" />  
</xsl:template>
```

- ◆ If that is the only template in your XSLT document, the output tree will be a copy of the source tree

- ✗ Omitting any **xml-stylesheet** processing instruction present in the source

- More interestingly, you can specify an action of apply-templates

- ◆ This tells the XSL processor to output the content of the current node and then recurse to lower level (child) nodes, examining all templates for matches
- ◆ Since this gets interesting, let's build a little example of an XML document and some XSLT files to transform it

XML and Transformation, 13

- ❑ In our example, we have the CD inventory from a music store captured in XML format
 - ◆ Each CD is an element
 - ◆ For each CD, the information carried is
 - ✗ Album title
 - ✗ Recording label
 - ✗ Artist
 - ✗ Recording label identifier for the album
 - ✗ Each song on the CD; each song has the following information included:
 - Song title
 - Song duration in
 - Minutes and
 - Seconds
- ❑ So each of these is an element or sub-element; for our example we just list a couple of CD' s and just a few songs from each
 - ◆ We call the document "Inpute" (this is for historical reasons)
 - ◆ So here' s the document...

XML and Transformation, 14

```
<?xml version="1.0" encoding="UTF-8"?>
<Inpute>
  <album>
    <atitle>Seeking</atitle>
    <label>Arcane</label>
    <artist>Adriane and the Outer Limits</artist>
    <label_id>Ar-1059</label_id>
    <song>
      <title>Matchsticks In The Ashtray</title>
      <duration>
        <min>03</min>
        <sec>19</sec>
      </duration>
    </song>
    <song>
      <title>There' s No Light In The Men' s Room</title>
      <duration>
        <min>03</min>
        <sec>52</sec>
      </duration>
    </album>
  <album>
    <atitle>Alexander' s Bag Time Gland</atitle>
    <label>Inane</label>
    <artist>Alexander</artist>
    <label_id>An-2029</label_id>
    <song>
      <title>Pot Time In The Old Town Tonight</title>
      <duration>
        <min>03</min>
        <sec>13</sec>
      </duration>
    </song>
    <song>
      <title>The Seventh Time Around</title>
      <duration>
        <min>02</min>
        <sec>25</sec>
      </duration>
    </song>
  </album>
</Inpute>
```

XML and Transformation, 15

- For starters, let's suppose we just want album titles and artist names

- ◆ We might start with an XSLT file like this:

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

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

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

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

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

</xsl:stylesheet>
```

- This seems to say:

- ◆ Start with the root ("/") and recurse down

- x When you find an **album** element, display its content and recurse down
- x When you find a **title** element, display its content and recurse down

XML and Transformation, 16

- ◆ But in fact, your output tree will be all the contents of all the elements strung together, that is:

```
<?xml version="1.0" encoding="UTF-8"?>SeekingArcaneAdriane and the Outer  
LimitsAr-1059Matchsticks In The Ashtray0319There' s No Light In The Men' s  
Room0352Alexander' s Bag Time GlandlnaneAlexanderAn-2029Pot Time In The  
Old Town Tonight0313The Seventh Time Around0225
```

- You need to tell the parser what elements you want to eliminate
 - ◆ If a node will never be reached (due to no matching or recursing), you will not see it in the output tree
 - ◆ If a node will be reached (explicitly due to matching or implicitly due to recursing), its content will be included in the output tree
 - ◆ To prevent the content of a node from going to the output tree, even if it is reached due to recursion, use an empty template
 - ◆ In our example, we could add:

```
<xsl:template match="song"/>  
<xsl:template match="title"/>  
<xsl:template match="duration"/>  
<xsl:template match="min"/>  
<xsl:template match="sec"/>
```

- ◆ And thus prevent the contents of song, title, duration, min, and sec elements from going to the output tree

XML and Transformation, 17

- The output now contains:

```
<?xml version="1.0" encoding="UTF-8"?>SeekingArcaneAdriane and the Outer  
LimitsAr-1059Alexander' s Bag Time GlandInaneAlexanderAn-2029
```

- ◆ Which is the XML header and the content of the album_title, label, artist, and label_id elements, but still all strung together

x Depending on your application, this may be OK

- ◆ Alternatively, you can specify the previous set of empty templates with a single template:

```
<xsl:template match="song | title | duration | min | sec"/>
```

- ◆ This produces the same result as above (note that the spaces around the vertical bars are optional)

- Along this line, if you force a node to be eliminated, it's child nodes will also be eliminated; for example,

```
<xsl:template match="song | title"/>
```

- ◆ Will work the same as the template above it, since eliminating song automatically eliminates duration, min, and sec

XML and Transformation, 18

□ But let' s suppose you want more structure for your result tree

◆ Suppose you want some markup, so the result is XML or HTML or XHTML

◆ Since it' s easy to imagine an application processing an XML document to create HTML output, let' s use that objective for our work

x Just be aware that the possible applications of XSLT processing can be much more flexible

XML and Transformation, 19

- ❑ So here's a stylesheet to produce an HTML version of this input tree:

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

  <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:output method="html"/>

    <xsl:template match="/">
      <html>
        <head>
          <title>Music Schmusic - CD Inventory</title>
        </head>
        <body>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>

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

    <xsl:template match="atitle">
      <h1>Album: <xsl:apply-templates/> </h1>
    </xsl:template>

    <xsl:template match="artist | song | duration | label | label_id"/>

  </xsl:stylesheet>
```

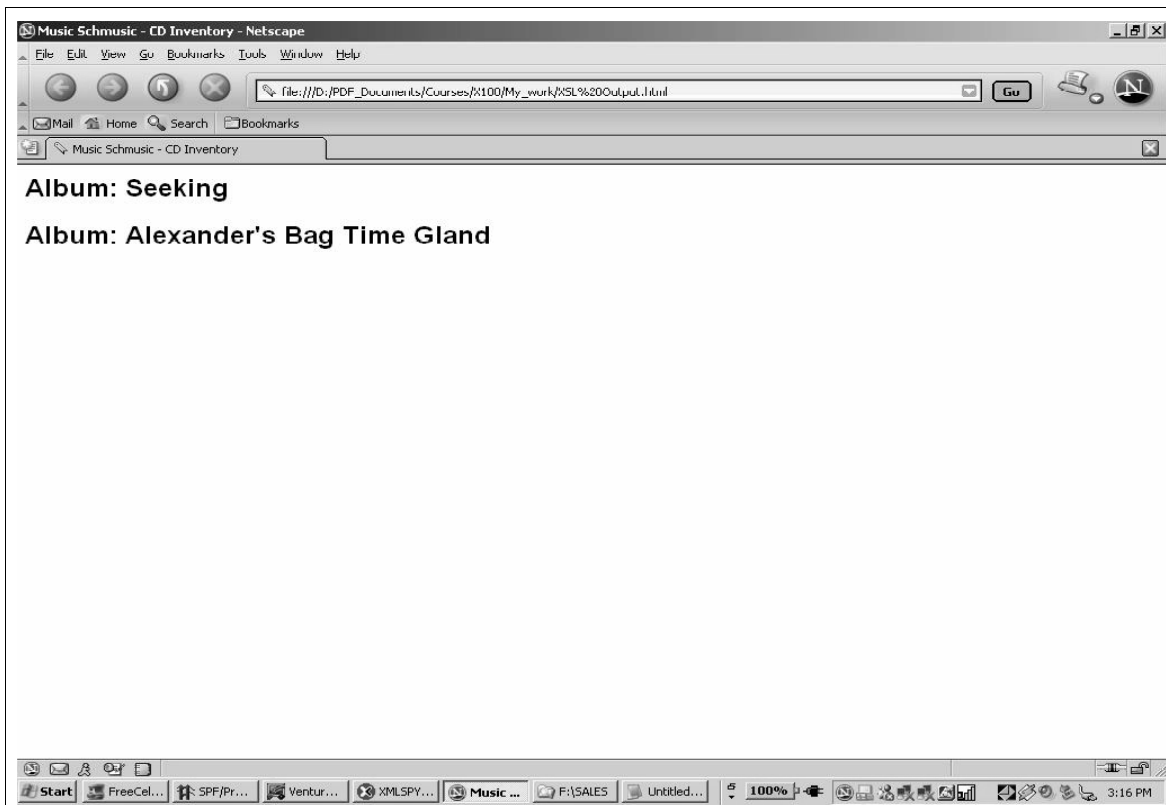
- ◆ Notice the literal HTML tags and the literal "Album: "; they get put out just as literals to the output tree

XML and Transformation, 20

- ❑ So the XSL processor might produce an output tree like this:

```
<html>
  <head>
    <META http-equiv="Content-Type"
      content="text/html; charset=UTF-16">
    <title>Music Schmusic - CD Inventory</title>
  </head>
  <body>
    <h1>Album: Seeking</h1>
    <h1>Album: Alexander' s Bag Time Gland</h1>
  </body>
</html>
```

- ❑ Which, when viewed through a browser might display like this:



XML and Transformation, 21

- Now, suppose you want to use the contents of several child elements for a single output element for the parent node
 - ◆ To be more precise, suppose we want the album title and artist name to be output together
 - ◆ We can accomplish this by referring to the value-of attributes of specific elements, for example:

```
<xsl:template match="album">
  <h2>Album Title: <xsl:value-of select="atitle"/> </h2>
  <h3>
  Artist: <xsl:value-of select="artist"/>
  </h3>
  <p>
  Songs .....
  </p>
  <xsl:apply-templates select="song"/>
</xsl:template>
```

- ◆ This says, for each album:
 - x Build a line with H2 attribute containing the words "Album Title: " followed by the content of the **atitle** tag, then
 - x Build a line with H3 attribute containing the word "Artist: " followed by the content of the "artist" element, then
 - x Build a text header paragraph for all songs to follow, then apply the template for all songs in this album

XML and Transformation, 22

- Here's how the whole XSLT document might look:

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:output method="html"/>

    <xsl:template match="/">
      <html>
        <head>
          <title>Music Schmusic - CD Inventory</title>
        </head>
        <body>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>

    <xsl:template match="album">
      <h2>Album Title: <xsl:value-of select="atitle"/></h2>
      <h3>
        Artist: <xsl:value-of select="artist"/>
      </h3>
      <p>Songs .....</p>
      <xsl:apply-templates select="song"/>
    </xsl:template>

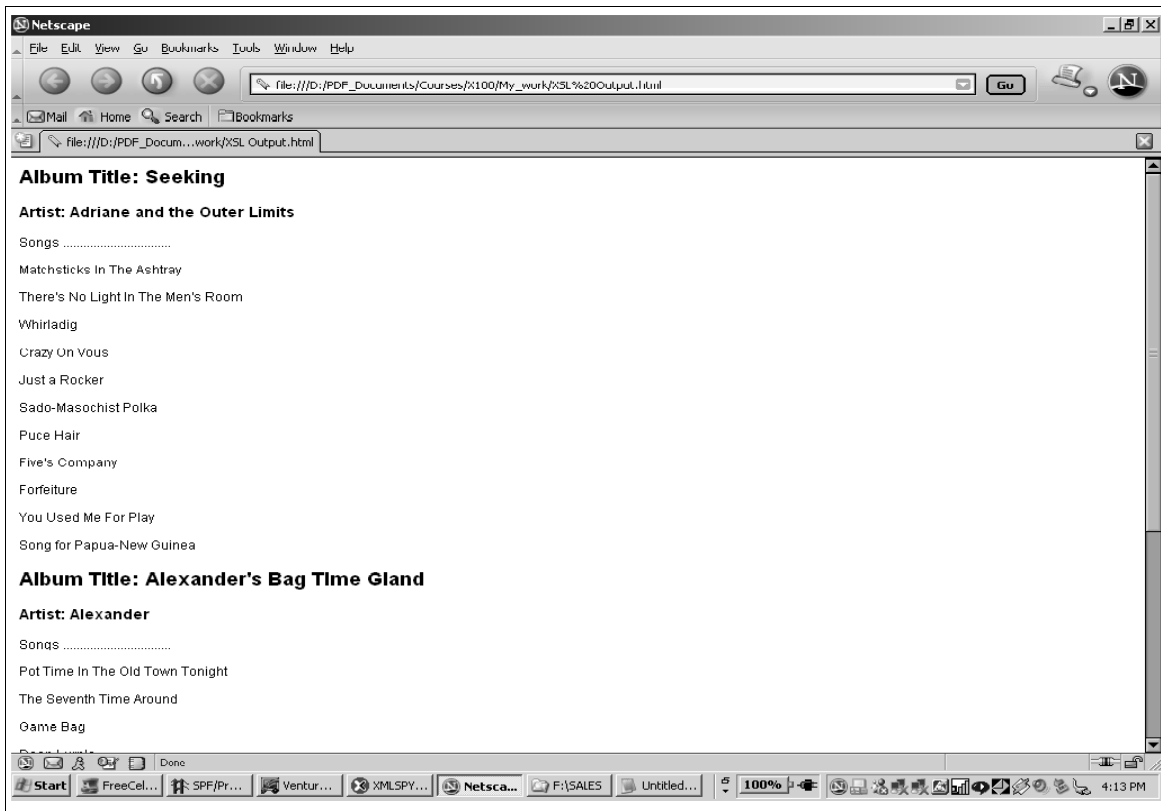
    <xsl:template match="song">
      <p><xsl:apply-templates/> </p>
    </xsl:template>

    <xsl:template match="duration | label | label_id"/>

  </xsl:stylesheet>
```

XML and Transformation, 23

- ❑ Now, this output might look like this (just to make it interesting we've added more albums and songs to the input tree):



- ❑ You can probably envision alternative ways to use HTML tags to format this just the way you like it

XML and Transformation, 24

- Finally, let's work with the `xsl:sort` transformation to sort the song titles in the albums
 - ◆ To do this, we open up the `apply-templates` tag, so instead of this:

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

- ◆ ... we code this:

```
<xsl:apply-templates select="song">  
  <xsl:sort select="title"/>  
</xsl:apply-templates>
```

- ◆ So the resulting XSLT document is on the next page...

XML and Transformation, 25

- So this XSLT document will sort the songs within each album:

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:output method="html"/>

    <xsl:template match="/">
      <html>
        <head>
          <title>Music Schmusic - CD Inventory</title>
        </head>
        <body>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>

    <xsl:template match="album">
      <h2>Album Title: <xsl:value-of select="atitle"/></h2>
      <h3>
        Artist: <xsl:value-of select="artist"/>
      </h3>
      <p>Songs .....</p>
      <xsl:apply-templates select="song">
        <xsl:sort select="title"/>
      </xsl:apply-templates>
    </xsl:template>

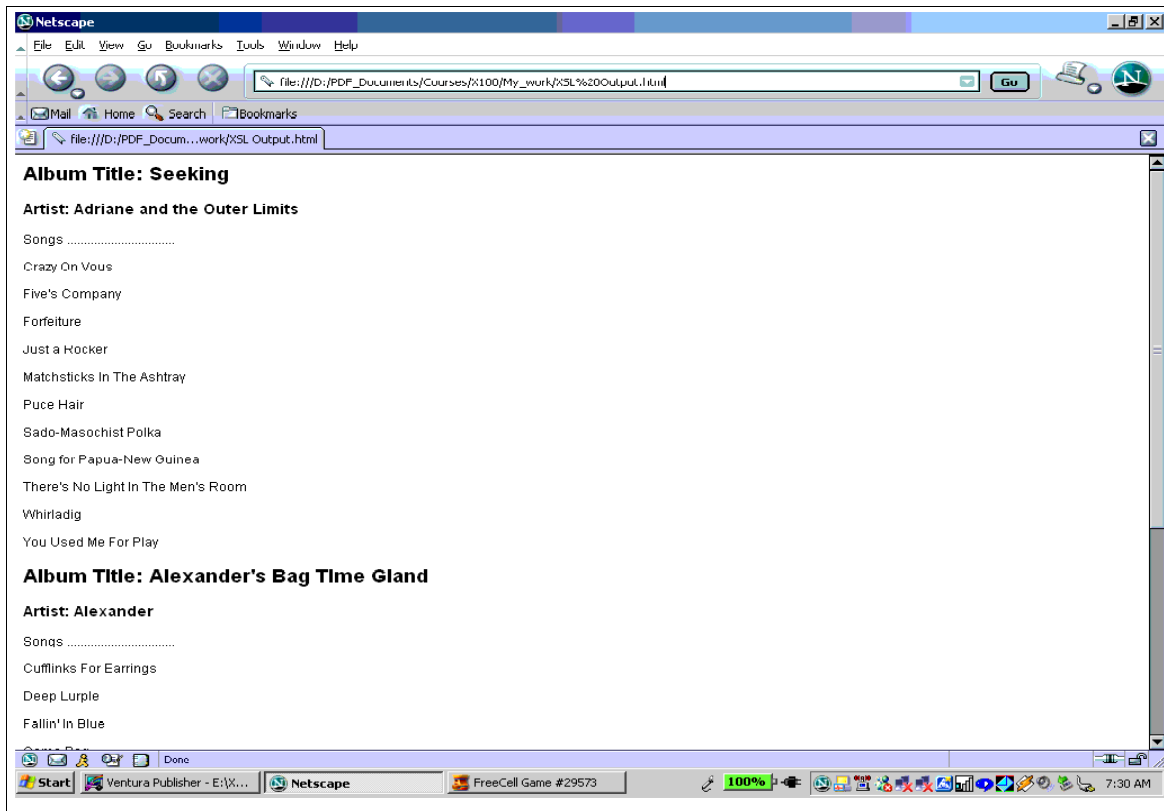
    <xsl:template match="song">
      <p><xsl:apply-templates/> </p>
    </xsl:template>

    <xsl:template match="duration | label | label_id"/>

  </xsl:stylesheet>
```

XML and Transformation, 26

- And the result might look like this:



- So there you have a brief introduction to XSLT

- ◆ It is a much richer standard than we've been able to indicate here, but this should get you started

Resources

- ❑ The official standards for XSL-related technology are found at these web sites

XSL

- ◆ <http://www.w3.org/TR/xsl/>

XSLT

- ◆ <http://www.w3.org/TR/xslt>

XPath

- ◆ <http://www.w3.org/TR/xpath>

The Whole XSL Family starting point

- ◆ <http://www.w3.org/Style/XSL/>