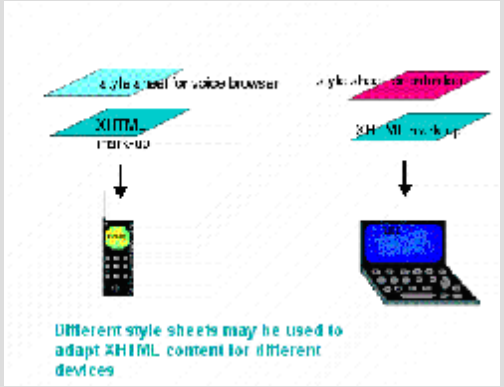
XML With Style

- **XML relies on additional technologies to provide a rendering mechanism**
  - two complementary technologies available
    - *Cascading Style Sheets*
      - introduced for HTML
        - *support is still patchy, however*
      - we saw CSS earlier
    - *XSL*
      - an XML-specific technology
        - *very new and in constant state of change*



From http://www.w3c.org/Style/CSS-vs-XSL.html:

*"**Why does W3C recommend two different style languages?***
...

Use CSS when you can, use XSL when you must.

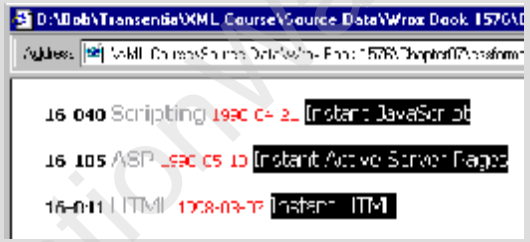
...
CSS is much easier to use, thus easier to maintain and cheaper. ... Some things you cannot do with CSS, or with CSS alone. Then you need XSL, or at least the transformation part of XSL"

The default IE5 stylesheet: *res://msxml.dll/DEFAULTSS.XSL*

A no-brainer…

- xml:stylesheet tag

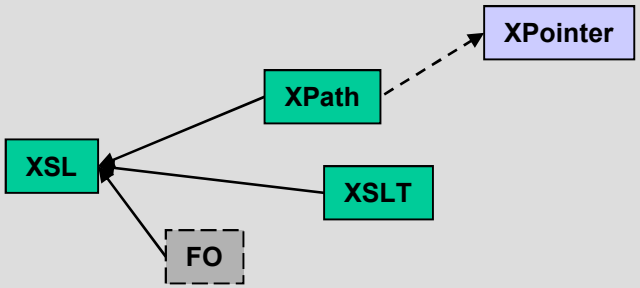
```
ITEM         { display:block; margin:15px }

CODE         { display:inline;
               font-family:Tahoma,Arial,sans-serif;
               font-size:10pt;
               font-weight:bold }

CATEGORY     { display:inline;
               font-family:Tahoma,Arial,sans-serif;
               color:darkgray;
               font-size:12pt;
               font-weight:bold }

RELEASE_DATE { display:inline;
               font-family:Tahoma,Arial,sans-serif;
               color:red;
               font-size:10pt }

TITLE        { display:inline;
               font-family:Tahoma,Arial,sans-serif;
               font-size:12pt;
               color:white;
               background-color:black }

SALES        { display:none }
```




```
<?xml version="1.0"?>
<?xml:stylesheet type="text/css" href="booklist.css"?>
<BOOKLIST>
 <ITEM>
  <CODE>16-048</CODE>
  <CATEGORY>Scripting</CATEGORY>
  <RELEASE_DATE>1998-04-21</RELEASE_DATE>
  <TITLE>Instant JavaScript</TITLE>
  <SALES>375298</SALES>
 </ITEM>
 <ITEM>
  <CODE>16-105</CODE>
  <CATEGORY>ASP</CATEGORY>
  <RELEASE_DATE>1998-05-10</RELEASE_DATE>
  <TITLE>Instant Active Server Pages</TITLE>
  <SALES>297311</SALES>
 </ITEM>
 <ITEM>
  <CODE>16-041</CODE>
  <CATEGORY>HTML</CATEGORY>
  <RELEASE_DATE>1998-03-07</RELEASE_DATE>
  <TITLE>Instant HTML</TITLE>
  <SALES>127853</SALES>
 </ITEM>
</BOOKLIST>
```

Example from "XML IE5"

- **XSL is not a simple thing**
  - a vocabulary for expressing stylesheets
    - *stylesheet language for XML*
  - various components and relationships
    - *XSL Transformations (XSLT)*
      - *transforms* one XML document *tree* into another XML document *tree*
        - *specifies patterns and applies templates to the matches*
    - *Formatting objects*
      - concerned with rendering/formatting an XML tree
    - *XPath*
      - addressing/selecting parts of a tree
      - a common specification with XPointer
  - forever changing
    - all *books are out of date…*



XSL & Multiple Types of Client

Client

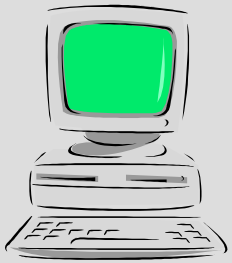
Client Application

XSLT Processor

Client

Client Application

XML Parser



Client Applicatio n

XML Parser

Client

Multiple views of the data created from the delivered XML data

XML Document

Middleware

Business Logic
Data integration

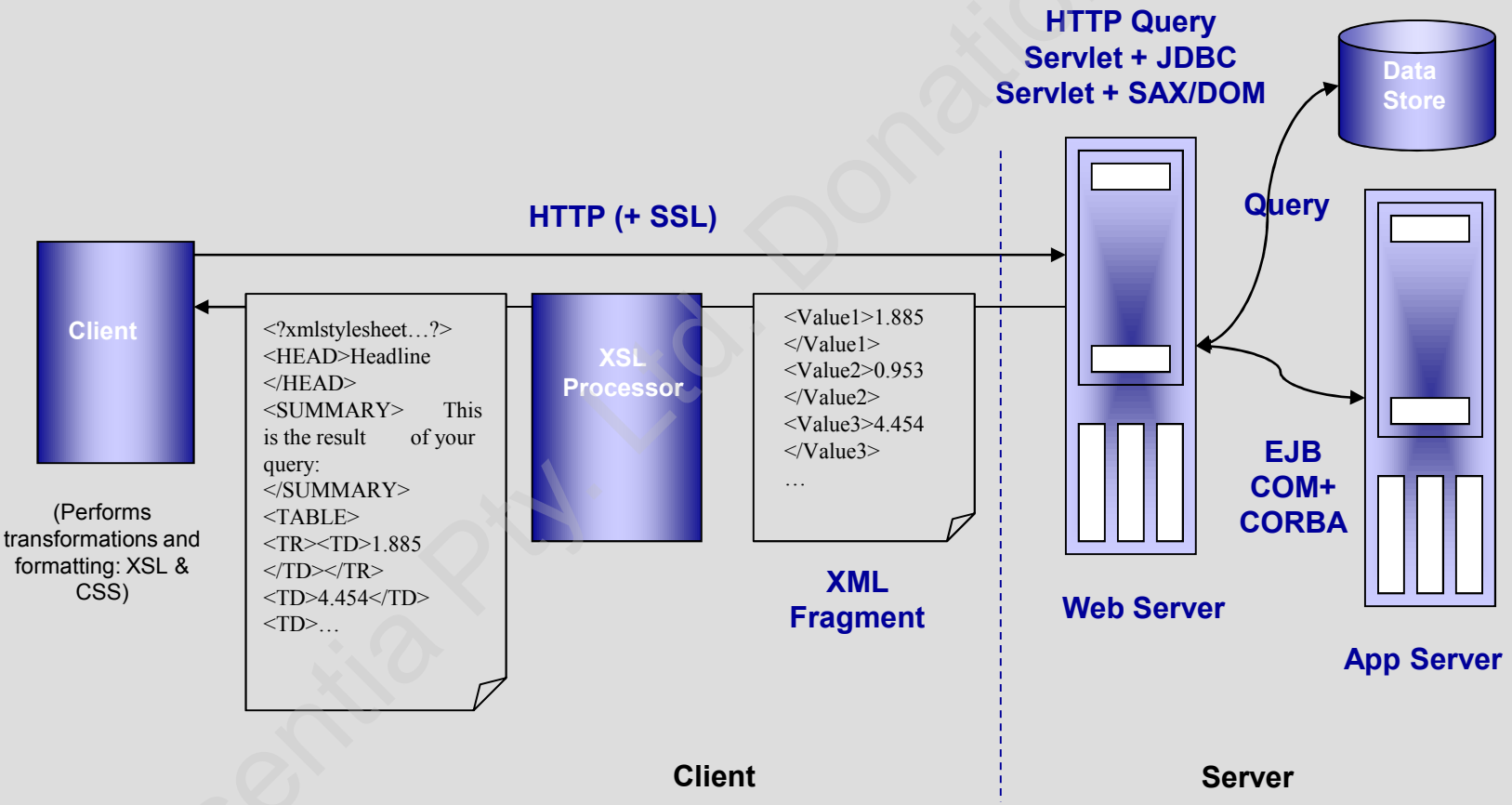
XML Parser

Database

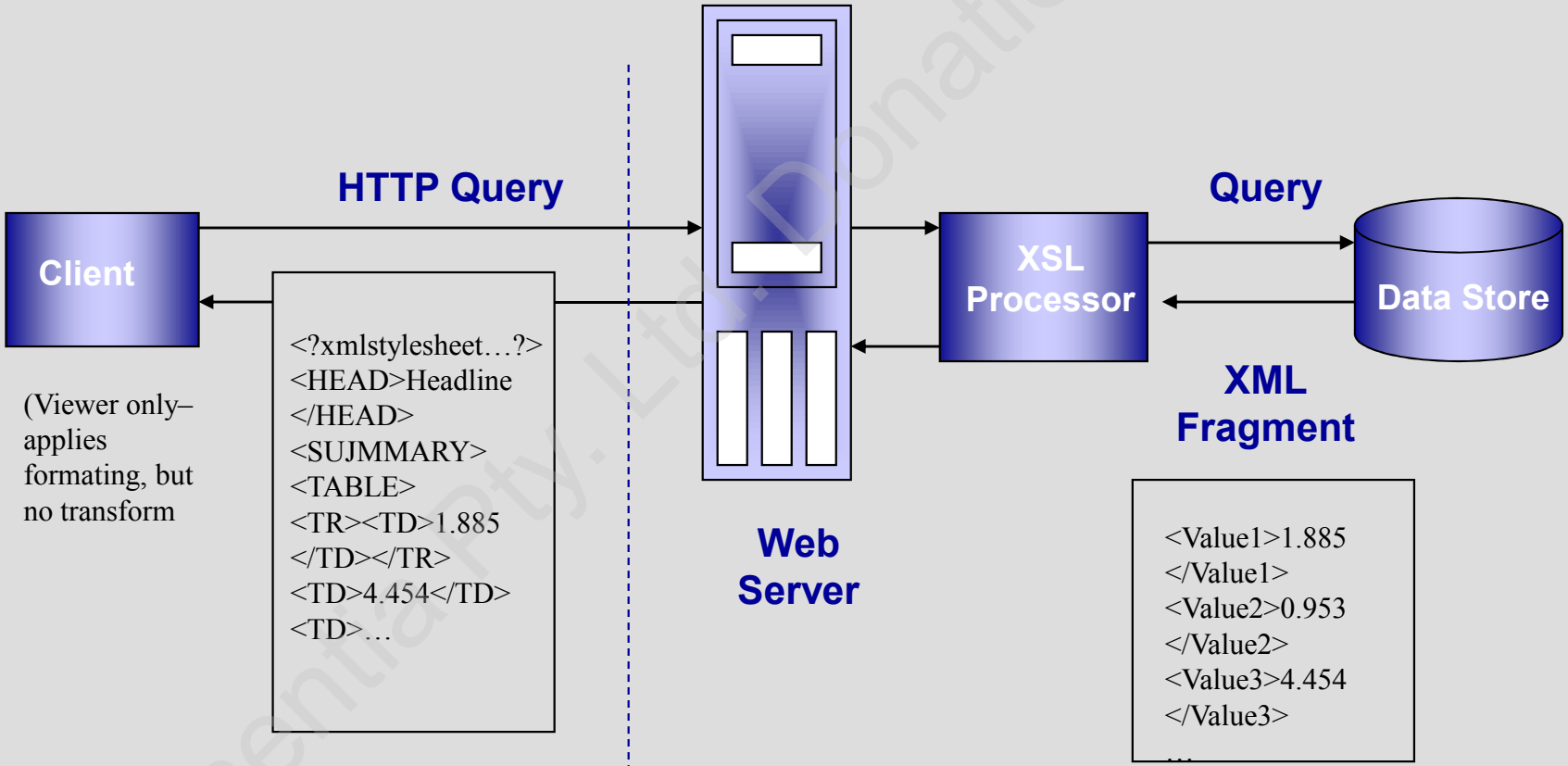
Server

Data can be served up as XML regardless of 'native' format

Client-Side XSL



Server-Side XSL



- **Objects that actually produce formatted output when used in XSL**
  - currently have to 'fall-back' to HTML for serious display purposes
    - *not pure!*
  - not currently supported by anything 'real'…
    - *will eventually provide ways to specify fonts, colours, paragraph styles, multiple columns, floating content, etc.*
      - as HTML does now
      - but in a more general way
        - *also more powerful, for more than just the web*

```
<xsl:template match = "emph">
  <fo:inline-sequence font-weight = "bold">
    <xsl:apply-templates />
  </fo:inline-sequence>
</xsl:template>
```

```
This is an <emph>important</emph> point!
```

XPath

- **A vocabulary for addressing/selecting parts of an XML document**
  - a declarative scheme
    - *states* what *data is wanted, not* how *to get to it*
  - used by both XSL and XPointer
  - note: it is possible to program with the DOM to do the same job but fiddly tree-walking programming is always something to avoid
    - *XPath is cleaner and (will be) standard*

XPath Goals

- **XPath should be:**
  - easy to express and parse
    - *should be easy to fit into a URL*
      - for automated processing

```
<THING href="http://www.thingworld.com/things/lookup.pl?name=fred+id=99" />
```

  - able to specify any path that can occur in an XML document and any set of conditions for the nodes in the path
  - able to uniquely identify any node in a document
  - a declarative scheme
    - *say what should be found, not* how *to find*
    - *allows optimisation and lets XPath fit into many situations*
  - language independent
    - *usable with Java, C++, JavaScript, VBScript, etc.*

More XPath

- **Two basic components**
  - a declarative path selection language and operators
    - *node position and hierarchy*
      - /
        - *child path operator. Selects direct children of a node.*
      - //
        - *recursive descent path operator. Selects matching nodes at any depth below the current node.*
      - .
        - *current context operator*
      - ..
        - *parent context operator*
      - @
        - *attribute path operator*
      - *
        - *wildcard operator*
    - *node index position*
      - path operators always return all things that match. The node index can be used to select a single thing
    - *filters and filter patterns*
      - selects one or more things based on a range of criteria
    - *operators and a function library*
      - used when building matching criteria
  - a method library
    - *IXTLRuntime in IE*

```
<xsl:for-each select="chapter">
  <H2>
    Chapter
      <xsl:eval>formatIndex(childNumber(this), "I")</xsl:eval>:
  <xsl:value-of select="title"/>
  </H2>
  <xsl:apply-templates />
</xsl:for-each>
```

XPath Patterns

- **Examples:**
  - find all author elements within the current context:

```
./author       author
```

  - find all first.name elements:

```
first.name
```

  - find the “bookstore” element at the root of this document:

```
/bookstore
```

  - find the root element of this document:

```
/*
```

  - find all author elements anywhere within the current document:

```
//author
```

  - find all bookstores where the value of the specialty attribute is equal to "textbooks":

```
/bookstore[@specialty = "textbooks"]
```

  - find all books where the value of the style attribute on the book is equal to the value of the specialty attribute of the bookstore element at the root of the document:

```
book[/bookstore/@specialty = @style]
```

Note: all examples are for IE5 and may not match the current W3C proposal…

More Patterns

- **More examples:**
  - find all author elements that contain at least one degree or award and at least one publication

```
author[(degree $or$ award) $and$ publication]
```

  - return book elements where any author element is “Bob”:

```
book[$any$ author="Bob"]
```

  - find all author elements containing a first-name child whose text is “Bob”:

```
author[first-name!text() = "Bob"]
```

  - find the first 3 degrees:

```
degree[index() $lt$ 3]
```

  - find the last category:

```
book/category[end()]
```

- **The usual operators are available:**
  - ( ), =, !=, lt, le, gt, ge, eq, ne, not, and, or
    - *note: can’t use <, <=*
      - without escaping…
  - any, all
  - case-insensitive matching
    - *ieq, ine, ilt, etc.*
  - the way that operators work is determined from the way that the operands are defined in the DTD/Schema
    - *if no DTD exists, some ‘appropriate’ action is taken*

- **Various:**
  - string
    - *string, concat, starts-with, substring, string-length, etc.*
      - all the usual suspects…
  - node set
    - *last, position, count, id, name, namespace-uri, local-name*
  - boolean
    - *boolean, not, true, false, lang*
  - number
    - *number, sum, floor, ceiling, round*

Alternative Syntax

- **Less convienient (?)**
  - gives a more 'procedural' feel

```
child::para[attribute::type="warning"][position()=5]
```

```
/descendant::olist/child::item
```

```
<A>
  <B>
    <C test="sampletest">sample</C>
    <C>sample2</C>
  </B>
  <B>
    <C>sample</C>
    <C>sample2</C>
    <D>sample3</D>
  </B>
</A>
```

```
A/B[2]/preceding-sibling::*
```

```
<B>
  <C test="sampletest">sample</C>
  <C>sample2</C>
</B>
```

XPath 2.0

- **Curently being worked on**
- **Has the following goals:**
  - simplify manipulation of XML Schema-typed content
  - simplify manipulation of string content
  - support related XML standards
  - improve ease of use
  - improve interoperability
  - improve i18n support
  - maintain backward compatibility
  - enable improved processor efficiency
  - explicit for-any and for-all processing
  - add intersection/difference operators

```
field[ . =~ "\d\d\d-\d\d\d\-\d\d\d\d ]
```

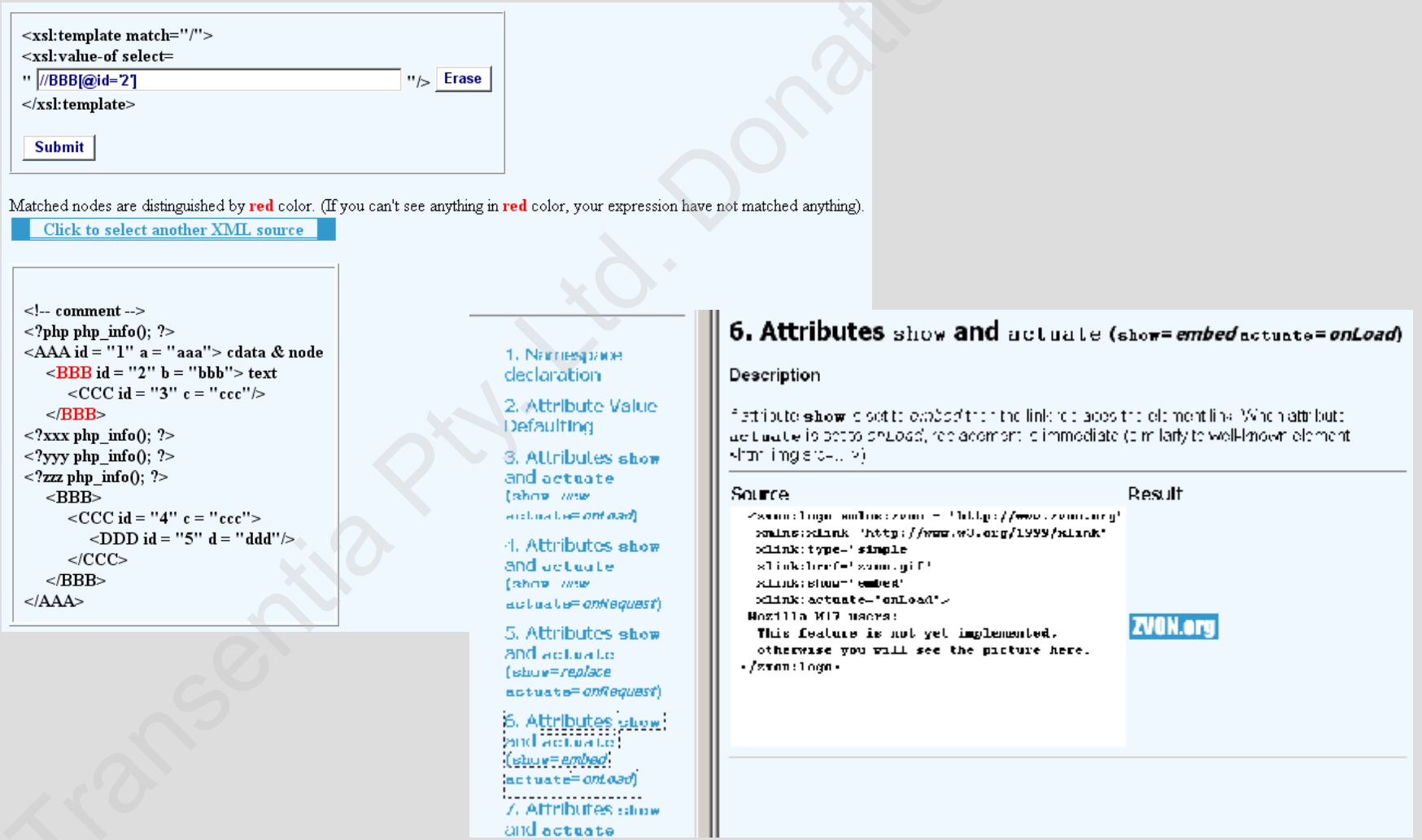
```
Department[Code =~ concat("[0-9][0-9]",$v,".*")]
```

```
document("otherdoc.xml")/id("foo")/a/b
```

```
/foo/(xxx|yyy)/li/(p|eg)
```

Useful Site

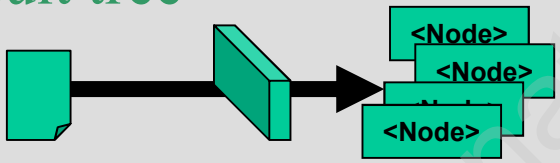
- **ZVON.org "The guide to the XML galaxy"**
  - interactive demos for various XML technologies
  - http://www.zvon.org/index.php?nav_id=tutorials

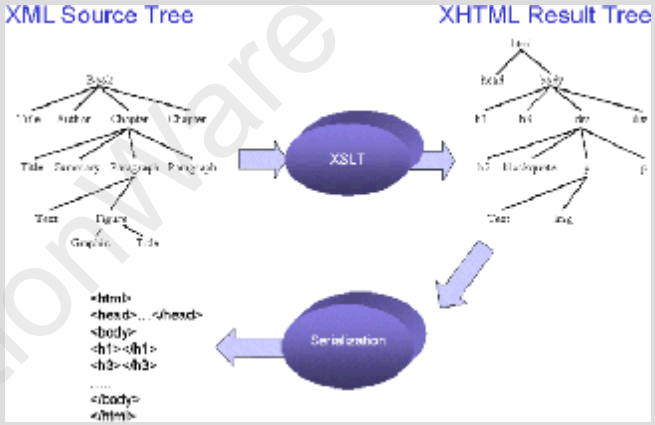


XSLT

XSL Transformations

- XSLT describes rules for transforming a source tree into a result tree





  - *achieved by associating XPath patterns with templates*
- in constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure added
  - *a major difference from CSS*
    - which just changes the 'look'
- XSLT is intended to be a general-purpose tool
  - *not just for XML to XML*
    - can also be used to transform XML to PDF, RTF, etc. etc.

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template>
    <html>
      <body>
       Hello, World!
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Simple XSLT Example

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>
<EMAILS>
   <EMAIL>
      <TO>sally@harem.com</TO>
      <FROM>bill@lovelorn.com</FROM>
      <SUBJECT>I need company tonight...</SUBJECT>
      <BODY>How about it, babe?</BODY>
   </EMAIL>
   <EMAIL>
      <TO>mary@lovenest.org</TO>
      <FROM>bill@lovelorn.com</FROM>
      <SUBJECT>I'm lonely...</SUBJECT>
      <BODY>Meet me tonight?</BODY>
   </EMAIL>
</EMAILS>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
      <xsl:for-each select="EMAILS/EMAIL/TO">
        <p><xsl:value-of /></p>
      </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Note the way in which matches/selections operate from the *current context*…

Templates

- **Fundamental to XSLT**
  - allow actions to be initiated and (eventually) output to be generated
  - have:
    - *a name*
      - used with <xsl:call-template>
        - *permits parameters*
    - *a match patttern*
      - alternative mechanism to the application of a named template
      - allows the XSL processor to decide when to apply the template based on features of the data
    - *a priority*
      - allowing pattern matching to be tuned
    - *a mode (may specify multiple modes)*
      - allows a template to behave differently depending on context

XSLT Modes Example

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE>
          <xsl:apply-templates select="customers/customer">
            <xsl:sort select="state"/>
            <xsl:sort select="name"/>
          </xsl:apply-templates>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match="customer">
     <TR>
        <xsl:apply-templates select="name" />
        <xsl:apply-templates select="address" />
        <xsl:apply-templates select="phone" />
        <xsl:apply-templates select="phone"
                             mode="accountNumber"/>
     </TR>
  </xsl:template>
  <xsl:template match="name">
    <TD STYLE="font-size:14pt font-family:serif">
      <xsl:apply-templates />
    </TD>
  </xsl:template>
  <xsl:template match="address">
    <TD> <xsl:apply-templates /> </TD>
  </xsl:template>
  <xsl:template match="phone">
    <TD> <xsl:apply-templates /> </TD>
  </xsl:template>
  <xsl:template match="phone" mode="accountNumber">
    <TD STYLE="font-style:italic">
      1-<xsl:value-of select="."/>-001
    </TD>
  </xsl:template>
</xsl:stylesheet>
```

from Microsoft's MSXML3 help

<xsl:call-template> Example

```
<xsl:param name="lang" select="en"/>
<xsl:variable name="messages"
  select="document(concat('resources/', $lang, '.xml'))/messages"/>

<xsl:template name="localized-message">
  <xsl:param name="name"/>
  <xsl:message>
    <xsl:value-of select="$messages/message[@name=$name]"/>
  </xsl:message>
</xsl:template>

<xsl:template name="problem">
  <xsl:call-template name="localized-message"/>
    <xsl:with-param name="name">problem</xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

resources/en.xml

```
<messages>
  <message name="problem">A problem was detected.</message>
  <message name="error">An error was detected.</message>
</messages>
```

from Microsoft's MSXML3 help

July 3, 2009

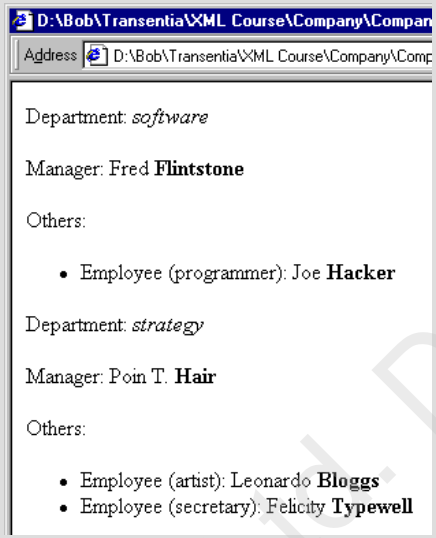
Basic XSLT Constructs

xsl:apply-imports	Invokes an overridden template rule.
xsl:apply-templates	Directs the XSLT processor to find the appropriate template to apply, based on the type and context of each selected node.
xsl:attribute	Creates an attribute node and attaches it to an output element.
xsl:attribute-set	Defines a named set of attributes.
xsl:call-template	Invokes a template by name.
xsl:choose	Provides multiple conditional testing in conjunction with the <xsl:otherwise element> and <xsl:when element>.
xsl:comment	Generates a comment in the output.
xsl:copy	Copies the current node from the source to the output.
xsl:copy-of	Inserts subtrees and result-tree fragments into the result tree.
xsl:decimal-format	Declares a decimal-format, which controls the interpretation of a format pattern used by the format-number function.
xsl:element	Creates an element with the specified name in the output.
xsl:fallback	Calls template content that can provide a reasonable substitute to the behavior of the new element when encountered.
xsl:for-each	Applies a template repeatedly, applying it in turn to each node in a set.
xsl:if	Allows simple conditional template fragments.
xsl:import	Specifies an XSLT style sheet to include.
xsl:include	Specifies another XSLT style sheet to include.
xsl:key	Declares a named key for use with the **key ()** function in XML Path Language (XPath) expressions.
xsl:message	Sends a text message to either the message buffer or a message dialog box.
xsl:namespace-alias	Replaces the prefix associated with a given namespace with another prefix.
xsl:number	Inserts a formatted number into the result tree.
xsl:otherwise	Provides multiple conditional testing in conjunction with the <xsl:choose element> and <xsl:when element>.
xsl:output	Specifies options for use in serializing the result tree.
xsl:param	Declares a named parameter for use within an xsl:stylesheet or xsl:template. Allows specification of a default value.
xsl:preserve-space	Preserves white space in a document.
xsl:processing-instruction	Generates a processing instruction in the output.
xsl:sort	Specifies sort criteria for node lists selected by xsl:for-each or xsl:apply-templates.
xsl:strip-space	Strips white space from a document.
xsl:stylesheet	The document element of a style sheet, containing all other style sheet elements.
xsl:template	Defines a reusable template for generating the desired output for nodes of a particular type and context.
xsl:text	Generates text in the output.
xsl:transform	Synonym for xsl:stylesheet.
xsl:value-of	Inserts the value of the selected node as text.
xsl:variable	Specifies a value bound in an expression.
xsl:when	Provides multiple conditional testing in conjunction with the <xsl:choose element> and <xsl:otherwise element>.
xsl:with-param	Passes a parameter to a template.

from Microsoft's MSXML3 help

Another Example

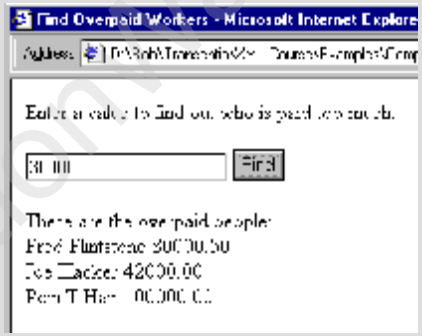
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Company.xsl"?>
<company>
  <dept code="software">
    <emp role="manager">
      <name>
        <first>Fred</first>
        <last>Flintstone</last>
      </name>
      <salary>80000.50</salary>
    </emp>
    <emp role="programmer">
      <name>
        <first>Joe</first>
        <last>Hacker</last>
      </name>
      <salary>42000.00</salary>
    </emp>
  </dept>
  <dept code="strategy">
    <emp role="manager">
      <name>
        <first>Poin</first>
        <middle>T</middle>
        <last>Hair</last>
      </name>
      <salary>100000.00</salary>
    </emp>
    <emp role="artist">
      <name>
        <first>Leonardo</first>
        <last>Bloggs</last>
      </name>
      <salary>13000.99</salary>
    </emp>
    <emp role="secretary">
      <name>
        <first>Felicity</first>
        <last>Typewell</last>
      </name>
      <salary>22000.10</salary>
    </emp>
  </dept>
</company>
```




```
<?xml version = "1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
      <xsl:for-each select="company/dept">
        <p>
          Department:
            <xsl:apply-templates select="@code" />
        </p>
        <p>
          Manager:
            <xsl:apply-templates
             select="emp[@role='manager']/name" />
        </p>
        <p>Others:</p>
        <ul>
        <xsl:for-each select="emp[@role!='manager']">
          <li>Employee (<xsl:value-of select="@role"/>):
              <xsl:apply-templates select="name" /></li>
        </xsl:for-each>
        </ul>
      </xsl:for-each>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="name">
    <xsl:apply-templates select="first" />
    <xsl:apply-templates select="middle" />
    <strong><xsl:apply-templates select="last" /></strong>
    <xsl:define-template-set>
      <xsl:template match="first">
        <xsl:value-of />
      </xsl:template>
      <xsl:template match="middle">
        <xsl:value-of />.
      </xsl:template>
      <xsl:template match="last">
        <xsl:value-of />
      </xsl:template>
    </xsl:define-template-set>
  </xsl:template>
  <xsl:template match="@code">
    <em><xsl:value-of /></em>
  </xsl:template>
</xsl:stylesheet>
```

Example (Cont'd)

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Find Overpaid Workers</title>
    <xml id="Company" src="Company.xml"></xml>
    <script>
      <!--
      function find()
        {
        overPaid.innerHTML = "";
        var xmlDoc = document.all ("Company").XMLDocument;
        var root = xmlDoc.documentElement
        var kids = root.getElementsByTagName ("emp");
        var value = inputText.value * 1.0;
        for (var i = 0; i < kids.length; i ++)
          {
          var node = kids.item(i);
          var salaries = node.getElementsByTagName ("salary");
          var salaryNode = salaries.item(0);
          var salaryValue = salaryNode.childNodes.item(0).nodeValue * 1.0;
          if (salaryValue >= value)
            overPaid.innerHTML += node.text + "<br>";
          }
        }
      -->
    </script>
  </head>
  <body>
    <p>
      Enter a value to find out who is paid too much:
    </p>
    <p>
      <input type="text" name="inputText" size="20">
      <input type="button" name="findButton" value="Find" onClick="find()">
    </p>
    <p>
      These are the overpaid people: <DIV ID="overPaid"></DIV>
    </p>
  </body>
</html>
```

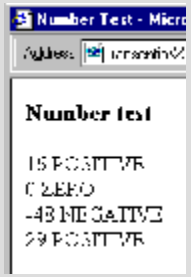


Loops & Decisions

- **Five major constructs:**
  - xsl:for-each
  - xsl:if
  - xsl:choose
  - xsl:when
  - xsl:otherwise

```
<xsl:template match="RELEASE_DATE">
   <xsl:if match="*[. $lt$ '1998-04-01']" >
      Note that this is an old title, and is now out of print.
   </xsl:if>
</xsl:template>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
   <HTML>
   <HEAD><TITLE>Number Test</TITLE></HEAD>
   <BODY>
   <H3>Number test</H3>
   <xsl:for-each select="//NUMBER">
      <xsl:value-of />
      <xsl:choose>
         <xsl:when match="*[. $gt$ 0]"> POSITIVE </xsl:when>
         <xsl:when match="*[. $lt$ 0]"> NEGATIVE </xsl:when>
         <xsl:otherwise> ZERO </xsl:otherwise>
      </xsl:choose>
      <BR />
   </xsl:for-each>
   </BODY>
   </HTML>
   </xsl:template>
</xsl:stylesheet>
```




```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="numbertest.xsl"?>
<NUMBERTEST>
   <NUMBER>16</NUMBER>
   <NUMBER>0</NUMBER>
   <NUMBER>-48</NUMBER>
   <NUMBER>29</NUMBER>
</NUMBERTEST>
```

Other Constructs

- **Create new nodes in the output:**
  - xsl:attribute
  - xsl:cdata
  - xsl:comment
  - xsl:element
  - xsl:entity-ref
  - xsl:pi
  - xsl:node-name

```
<xsl:template match="/">
  <xsl:for-each select="//">
    Node name: <xsl:node-name /> </br>
  </xsl:for-each>
</xsl:template>
```

- **Various others:**
  - xsl:script
  - xsl:eval

```
<xsl:script langauge="VBScript">
  Function getDateTime()
    getDateTime = Now()
  End Function
</xsl:script>
…
<xsl:eval>getDateTime()</xsl:eval>
```

- **There are others defined in the standard**
  - import files, tailoring error messages, passing parameters and using values, tailoring the output format, etc.

More Examples

- The default template invoked when nothing else matches:

```
<xsl:template>
  <xsl:value-of />
</xsl:template>
```

- Defining the order in which matching elements will be displayed:

```
<xsl:for-each select="book/author" order-by="+last_name; +first_name; -age">
  …
</xsl:for-each>
```

- Creating a comma-separated list:

```
<xsl:for-each select="products/product">
  <xsl:sort select="product" order="descending"/>
    <xsl:value-of select="."/><xsl:if test="position()!=last()">, </xsl:if>
</xsl:for-each>
```

- **Resolving conflicts**
  - which template is matched?

```
<xsl:template match="NAME">
  …
</xsl:template>

<xsl:template match="PLANT/NAME">
  …
</xsl:template>
```

  - *the one with the highest priority is chosen*
  - *if several templates have the same priority, the last (textually) in the style sheet is chosen*

Table Example

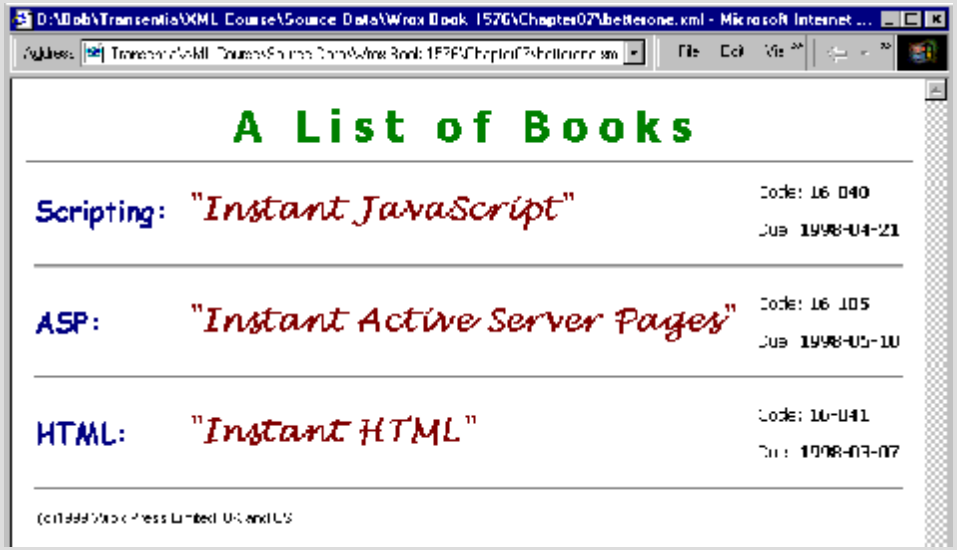
- From "XML IE5":
  - Alex Homer, Wrox Press

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="betterone.xsl"?>
<BOOKLIST>
 <ITEM>
  <CODE>16-048</CODE>
  <CATEGORY>Scripting</CATEGORY>
  <RELEASE_DATE>1998-04-21</RELEASE_DATE>
  <TITLE>Instant JavaScript</TITLE>
  <SALES>375298</SALES>
 </ITEM>
 <ITEM>
  <CODE>16-105</CODE>
  <CATEGORY>ASP</CATEGORY>
  <RELEASE_DATE>1998-05-10</RELEASE_DATE>
  <TITLE>Instant Active Server Pages</TITLE>
  <SALES>297311</SALES>
 </ITEM>
 <ITEM>
  <CODE>16-041</CODE>
  <CATEGORY>HTML</CATEGORY>
  <RELEASE_DATE>1998-03-07</RELEASE_DATE>
  <TITLE>Instant HTML</TITLE>
  <SALES>127853</SALES>
 </ITEM>
</BOOKLIST>
```

```
<xsl:template match="/">
  <HTML>
    <BODY>
      <DIV STYLE="font-family:Tahoma,Arial,sans-serif;
                  font-size:24pt; color:green;
                  text-align:center; letter-spacing:8px;
                  font-weight:bold">
        A List of Books
      </DIV>
      <HR />
      <TABLE WIDTH="100%" CELLPADDING="5">
        <xsl:apply-templates select="//ITEM" />
      </TABLE>
      <DIV STYLE="font-family:Arial,sans-serif;
                  font-size:8pt; margin-left:10px">
        (c)1999 Wrox Press Limited, UK and US
      </DIV>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match="ITEM">
  <TR>
    <xsl:apply-templates select="CATEGORY" />
    <xsl:apply-templates select="TITLE" />
    <xsl:apply-templates select="CODE" />
  </TR>
  <TR>
    <xsl:apply-templates select="RELEASE_DATE" />
  </TR>
  <TR><TD COLSPAN="3"><HR /></TD></TR>
</xsl:template>

<xsl:template match="CODE">
  <TD STYLE="font-family:Tahoma,Arial,sans-serif;
             font-size:10pt">
    Code: <B><xsl:value-of /></B>
  </TD>
</xsl:template>
…
```

Table Example (cont'd)

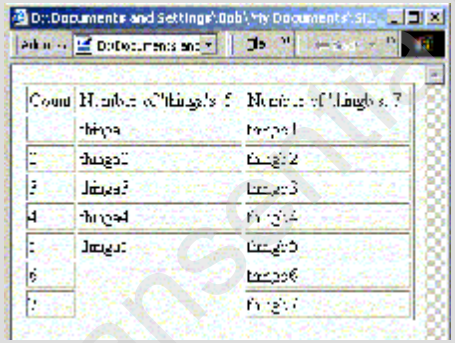
```
...
<xsl:template match="CATEGORY">
  <TD ROWSPAN="2"
      STYLE="font-family:Comic Sans MS, Arial,sans-serif;
             color:darkblue; font-size:16pt;
             font-weight:bold">
    <xsl:value-of />:
  </TD>
</xsl:template>

<xsl:template match="RELEASE_DATE">
  <TD STYLE="font-family:Tahoma,Arial,sans-serif;
             font-size:10pt">
    Due: <B><xsl:value-of /></B>
  </TD>
</xsl:template>

<xsl:template match="TITLE">
  <TD ROWSPAN="2"
      STYLE="font-family:Lucida Handwriting Italic,Arial,sans-serif;
             font-size:18pt; font-weight:bold; color:darkred">
    "<xsl:value-of />"
  </TD>
</xsl:template>

</xsl:stylesheet>
```



A Tricky Problem...

• Side-by-side display

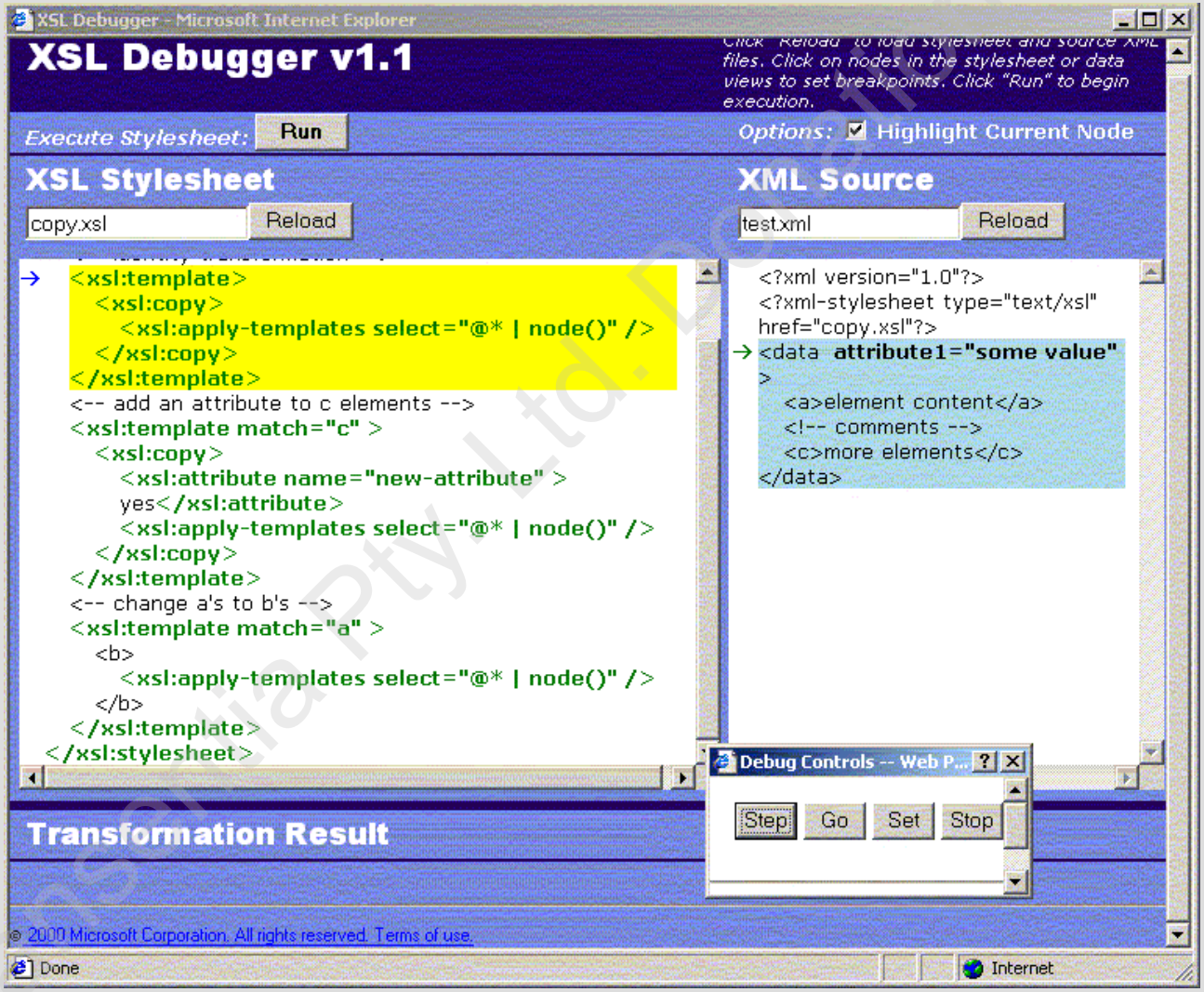
```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"
                 href="sbstest.xsl" ?>
<test>
  <thinga>thinga1</thinga>
  <thinga>thinga2</thinga>
  <thinga>thinga3</thinga>
  <thinga>thinga4</thinga>
  <thinga>thinga5</thinga>
  <thingb>thingb1</thingb>
  <thingb>thingb2</thingb>
  <thingb>thingb3</thingb>
  <thingb>thingb4</thingb>
  <thingb>thingb5</thingb>
  <thingb>thingb6</thingb>
  <thingb>thingb7</thingb>
</test>
```



```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <xsl:variable name="a" select="test/thinga"/>
        <xsl:variable name="aCount" select="count($a)"/>
        <xsl:variable name="b" select="test/thingb"/>
        <xsl:variable name="bCount" select="count($b)"/>
        <table width="100%" border="1">
          <tr>
            <td>Count</td>
            <td>Number of 'thinga's: <xsl:value-of select="$aCount" /></td>
            <td>Number of 'thingb's: <xsl:value-of select="$bCount" /></td>
          </tr>
          <xsl:choose>
            <xsl:when test="$aCount >= $bCount">
              <xsl:for-each select="$a">
                <xsl:variable name="p" select="position()"/>
                  <tr>
                    <td><xsl:value-of select="$p" /></td>
                    <td><xsl:value-of select="$a[$p]" /></td>
                    <td><xsl:value-of select="$b[$p]"/></td>
                  </tr>
                </xsl:for-each>
              </xsl:when>
              <xsl:otherwise>
                <xsl:for-each select="$b">
                  <xsl:variable name="p" select="position()"/>
                    <tr>
                      <td><xsl:value-of select="$p" /></td>
                      <td><xsl:value-of select="$a[$p]" /></td>
                      <td><xsl:value-of select="$b[$p]"/></td>
                    </tr>
                  </xsl:for-each>
                </xsl:otherwise>
          </xsl:choose>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

XSL Debugger

- From Microsoft
  - http://msdn.microsoft.com/downloads/samples/internet/default.asp?url=/Downloads/samples/Internet/xml/xsl_debugger/Default.asp



Server-Side XSL

- **XSL's transforming capabilities can also be used at the server side**
  - filtering, customizing, or changing the schema, etc.
  - makes it possible to deploy content in XML and transform it to HTML on demand for "down-level" clients
  - will probably be one of the major uses
  - simple Active Server Pages example:

```
<%@ LANGUAGE = JScript %>
<%
  // Set the source and style sheet locations here
  var sourceFile = Server.MapPath("simple.xml");
  var styleFile = Server.MapPath("simple.xsl");

  // Load the XML
  var source = Server.CreateObject("Microsoft.XMLDOM");
  source.async = false;
  source.load(sourceFile);

  // Load the XSL
  var style = Server.CreateObject("Microsoft.XMLDOM");
  style.async = false;
  style.load(styleFile);

  Response.Write(source.transformNode(style));
%>
```

  - another example on next page

XML/ASP Example

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <BODY STYLE="font-family:Arial, helvetica, sans-serif; font-size:12pt;
        background-color:#EEEEEE">
    <xsl:for-each select="breakfast-menu/food">
      <DIV STYLE="background-color:teal; color:white; padding:4px">
        <SPAN STYLE="font-weight:bold; color:white"><xsl:value-of select="name"/></SPAN>
        - <xsl:value-of select="price"/>
      </DIV>
      <DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt">
        <xsl:value-of select="description"/>
        <SPAN STYLE="font-style:italic">
          (<xsl:value-of select="calories"/> calories per serving)
        </SPAN>
      </DIV>
    </xsl:for-each>
  </BODY>
</HTML>
```

```
<?xml version='1.0'?>
<?xml:stylesheet
  type="text/xsl"
  href="simple.xsl" ?>
<breakfast-menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>two of our famous
                 Belgian Waffles with
                 plenty of real maple
                 syrup</description>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>light Belgian waffles
                 coverred with strawberrys
                 and whipped cream</description>
    <calories>900</calories>
  </food>
  <food>
    <name>Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    ...
```

```
<%@ LANGUAGE = JScript %>
<%
  function reportError(where, error)
  {
    Response.Write("<font face=Arial><B>Error loading '" + where +
                   "'</B></font> <BLOCKQUOTE><XMP>" + error.reason +
                   "</XMP></BLOCKQUOTE>");
  }

  // Load the XML
  var doc = Server.CreateObject("Microsoft.XMLDOM");
  doc.async = false;
  doc.load(Server.MapPath("simple.xml"));
  if (doc.parseError.errorCode != 0)
    reportError("simple.xml", doc.parseError);
  else
  {
    // Load the stylesheet
    var style = Server.CreateObject("Microsoft.XMLDOM");
    style.async = false;
    style.load(Server.MapPath("simple.xsl"));
    if (style.parseError.errorCode != 0)
      reportError("simple.xsl", doc.parseError);
    var result = doc.transformNode(style);
    Response.Write(result);
  }
%>
```

SAXON

- **A collection of tools for processing XML documents:**
  - an XSLT processor with a number of extensions
  - a Java library, which supports a similar processing model to XSL, but allows full programming capability
    - *needed if you want to perform complex processing or to access external services such as a relational database*
  - an improved version of the Ælfred parser
    - *can use SAXON with any SAX-compliant XML parser*
- **Can use SAXON by writing XSL stylesheets, by writing Java applications, or by any combination of the two**
- **Also provides a library of extensions**
  - permit things that are difficult to achieve, or inefficient, using standard XSLT facilities alone
- **Instant SAXON**
  - the XSLT interpreter for Windows