Updates and companion files available:
ibm.com/developerWorks/speakers/colan

XSL by Example

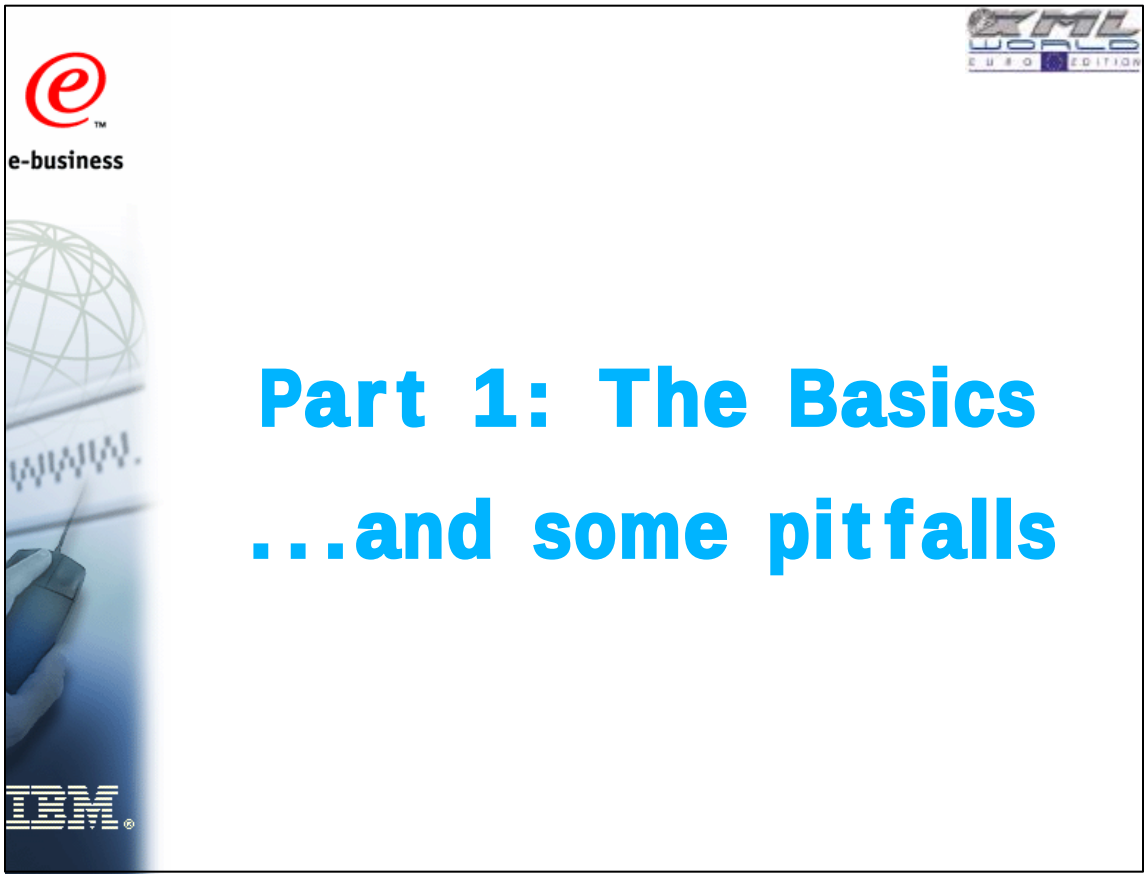
an introduction to writing
XSL stylesheets

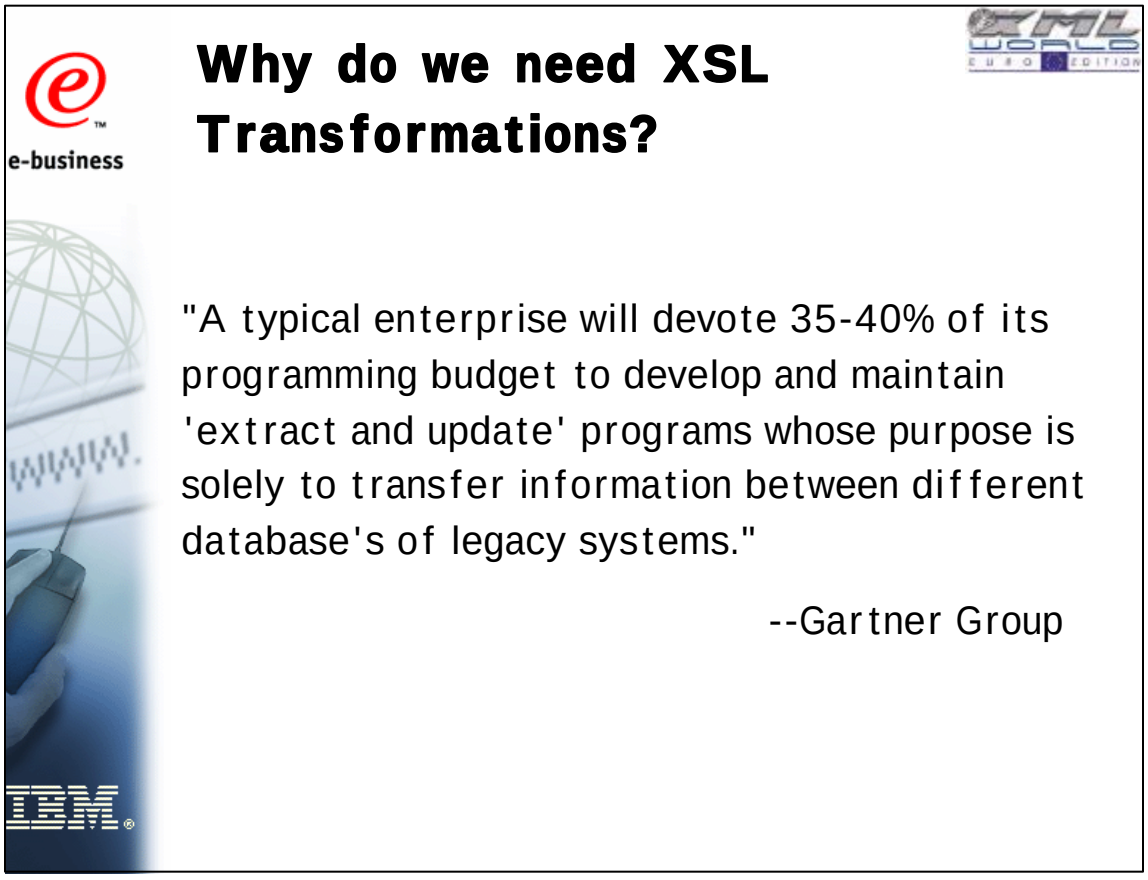
version 1.4

Mark Colan
XML Technologist, IBM Corporation
mcolan@us.ibm.com

Agenda

- The Basics
- Generating a complete web page
- Sorting and numbering
- Fun with XPath
- XML-to-XML vocabulary translation
- Formatting Objects and PDFs
- XSL in e-business Solutions
- XSL Development
- Q & A


e-business
Part 1: The Basics
...and some pitfalls
IBM


e-business
Why do we need XSL Transformations?
"A typical enterprise will devote 35-40% of its programming budget to develop and maintain 'extract and update' programs whose purpose is solely to transfer information between different database's of legacy systems."
--Gartner Group
IBM

XML is about REPURPOSING data

- Data is expensive to enter, validate, update
- Data is a valueable asset!
- XML data is easy to repurpose and reuse
- ...using standard tools for transformation

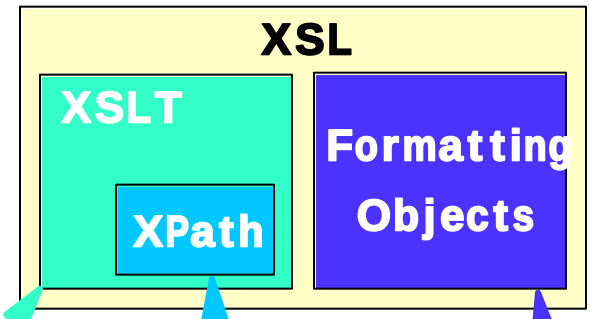
XSL allows XML data to be restructured, restyled, or converted to another form

- XML to HTML
- XML to XML (vocabulary translation)
- XML to Formatting Objects
- XML to non-XML (e.g. comma-separated files)

Alternatives to XSL

- XML Parser + Java DOM code
  - ►infinitely flexible
  - ►LOTS of code to develop and maintain
  - ►does not address problem cited by Gartner Group statement
- PERL 5 + XML4P
  - ►DOM access in a Perl environment
  - ►suitable for server deployment
  - ►XML4P available at xml.apache.org
  - ►less code than Java, but more than XSL

XSL: Three parts

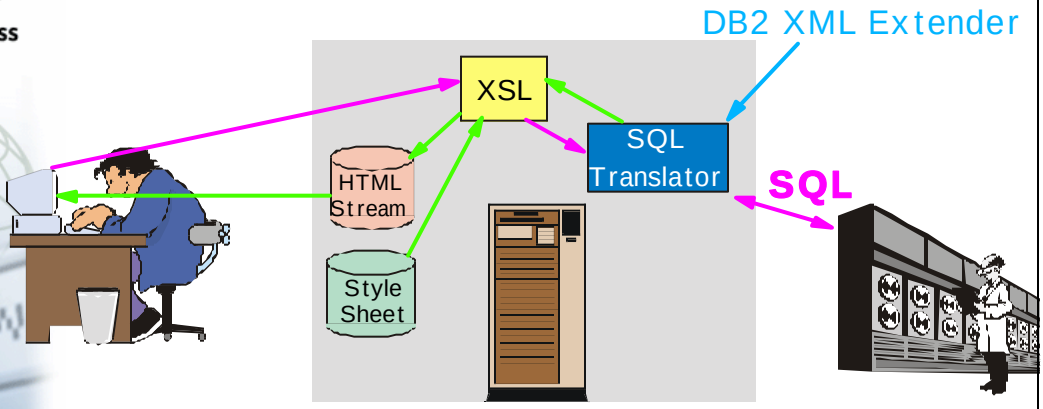


Transformation language

a language for addressing parts of an XML document

- an XML Vocabulary
- for specifying
- formatting semantics

How is XSL used?



Client

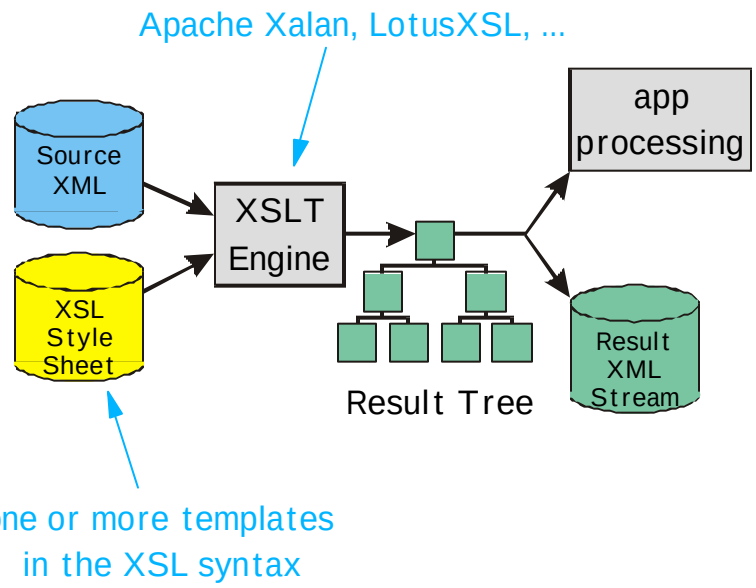
Middle-tier Server

Legacy Data Store

IBM Websphere, Lotus Domino, Apache, etc

DB2 Universal Database

XSL Processing Overview



XSLT: A Simple Example

(...maybe just a little too simple!)

Problem: Display the title (only) in html as <h1> headline

```
<book>
    <author>Tom Wolfe</author>
    <title>The Right Stuff</title>
    <price>$0.00</price>
</book>
```

XML Input File: Book0.xml

XSL PROCESSOR

```
<xsl:stylesheet>
  <xsl:template match="title">
    <h1>
      <xsl:apply-templates/>
    </h1>
  </xsl:template>
</xsl:stylesheet>
```

XSL Style Sheet Input File: Book0.xsl

```
<h1>
The Right Stuff
</h1>
```

HTML Output

Book0.xml

Book0.xsl

Anatomy of a stylesheet

identify XML document

must enclose the entire stylesheet

import, include, output, strip-space, key, param, variable, ...

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet>
  <!-- other directives -->
  <xsl:template match="title">
    <h1>
      <xsl:apply-templates/>
    </h1>
  </xsl:template>
</xsl:stylesheet>
```

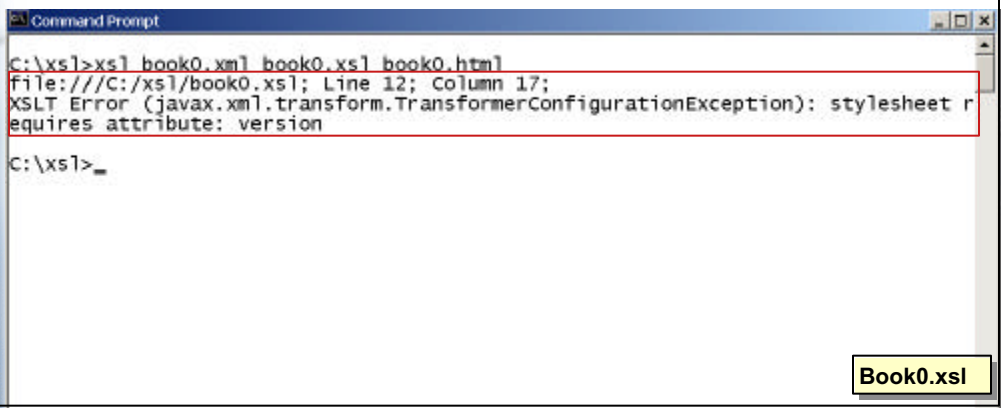
one or more template groups

literal result text and/or xsl processing directives, mixed freely

Book0.xsl

Hey! That example doesn't work!

- Right. I was trying to keep your first example REAL EASY to understand.
- But XSL is not quite THAT simple (sigh)
- Now we'll figure out what needs to be fixed



Book0.xsl

<xsl:stylesheet> element

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <!-- other elements -->
</xsl:stylesheet>
```

Specifies:

- a string that specifies a particular vocabulary (in this case, the W3C XSLT 1.0 vocabulary)
- an abbreviation used to label every XSLT statement
- a version number attribute

There must be exactly one <xsl:stylesheet> element (and end tag); it encloses all other stylesheet elements.

By the way...

If the <xsl:stylesheet> element does not correctly specify the xsl: namespace definition, in Xalan 2.0 you may get the same error message

➡ stylesheet requires attribute: version

...even when you do have a "version" attribute

```
<xsl:stylesheet
  xmlns:xsl="incorrectly typed, or missing"
  version="1.0">
```

Now <xsl:stylesheet> is correct...

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="title">
     <H1>
       <xsl:apply-templates/>
     </H1>
  </xsl:template>
</xsl:stylesheet>
```

...but why does it have the <author> and <price> values?

...and here's the output:

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

Tom Wolfe
 <H1>The Right Stuff</H1>
$6.00
```

Book2.xsl

<xsl:template> element

```
<xsl:template match="match expression">
   <!-- literal result text, XSLT elements -->
</xsl:template>
```

Specifies:

- a *match expression* that defines when this rule will be used - this is an XPath expression
- literal result elements to copy to output, and/or other XSLT elements to cause other actions
- priority and/or template-name (not covered)

A stylesheet has one or more templates.

- When <xsl:template> is missing, and literal result elements are found, a match="/" template is assumed

<xsl:apply-templates/> element

Invokes rules as appropriate to process children of the current node. These are:

- the template match rules you define
- default rules built-in to the XSL processor

In our stylesheet, <xsl:apply-templates/> is used in the rule we defined for <title>:

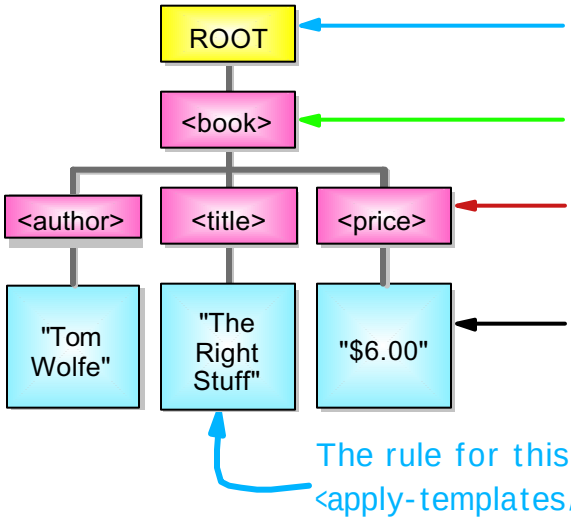
```
<xsl:template match="title">
   <H1>
     <xsl:apply-templates/>
   </H1>
</xsl:template>
```

So what are the children of the current node?

Tree representation of the sample XML document

```
<book>
   <author>Tom Wolfe</author>
   <title>The Right Stuff</title>
   <price>$6.00</price>
</book>
```

Book0.xml



The tree is created by parsing XML document

Tree has an implicit root node

<book> is outermost group for this doc

sub-elements (children) of <book>

these children are text nodes

The rule for this text node is invoked by <apply-templates/> used in <title>'s rule

Built-in template rules (1)

- **Text and attribute nodes:**
  copy the **text** to output

```
<tag>text<tag>
<tag attribute="text"/>
```

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

....that's how the <title> text node gets written out

- For processing-instructions and comments, do nothing.

```
<?xml version="1.0"?>
<!-- comment -->
```

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

Note: this is a short cut for, and equivalent to:

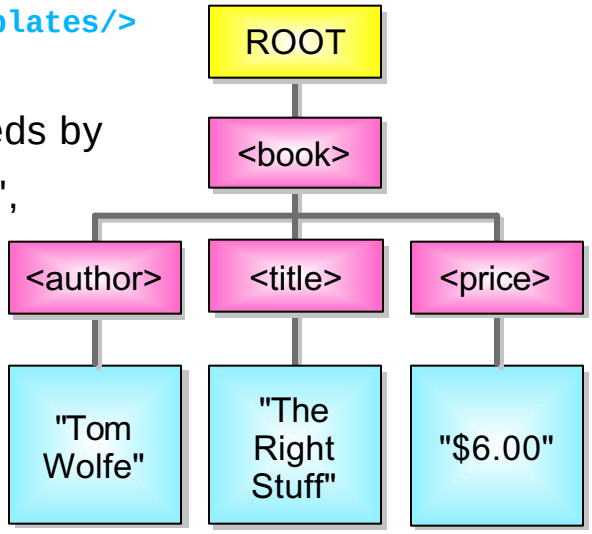
```
<xsl:template
   match="processing-instruction()|comment()">
</xsl:template>
```

Built-in template rules (2)

- **Element (and root) nodes:**
  apply-templates to each child

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

Processing proceeds by "walking the tree", in a recursive fashion.



How does this apply to our example?

- Since there is no explicit rule for <book>, templates for <title>, <author>, and <price> are applied
- There's an explicit template for <title>, to write its text between <h1> ... </h1> tags
- Since there is no explicit template for <author> and <price>, the built-in rule is invoked for each of them...
- ...which causes the built-in rule for text nodes to be invoked for text children of <author> and <price>...
- ...which writes out <author> and <price> text. **oops!**

How can we stop the effect of the built-in rule for <author> and <price> text children?

Overriding built-in rules

built-in rule has apply-templates; override to do nothing instead.

Just add new rules.

We don't want to apply-templates for <author> or <price>.

```
<xsl:template match="author">
</xsl:template>
<xsl:template match="price">
</xsl:template>
```

vertical bar (Shift+backslash) means "OR"

One template can match two tags:

```
<xsl:template match="author|price">
</xsl:template>
```

standard XML trick (means there is no data or sub-elements)

even more concisely:

```
<xsl:template match="author|price"/>
```

One way to fix the problem...

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="title">
     <h1>
       <xsl:apply-templates/>
     </h1>
  </xsl:template>
  <xsl:template match="author|price"/>
</xsl:stylesheet>
```

Book3.xsl

...here's the output:

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

<H1>The Right Stuff</H1>
```

What if there are lots of other tags in <book>?

Other ways to fix the problem

```
<xsl:template match="title">
   <h1>
     <xsl:apply-templates/>
   </h1>
</xsl:template>
<xsl:template match="author|price"/>
```

Book2.xsl

Instead of choking off the data we don't want to see,

```
<xsl:template match="book">
   <h1>
     <xsl:apply-templates
         select="title"/>
   </h1>
</xsl:template>
```

Book4.xsl

we can apply-templates selectively (e.g, just for <title>),

```
<xsl:template match="book">
   <h1>
     <xsl:value-of select="title"/>
   </h1>
</xsl:template>
```

Book5.xsl

OR we can just pull out the data for the child node we want to show.

One more detail

```
<xsl:stylesheet

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="book">
     <h1>
       <xsl:apply-templates select="title"/>
     </h1>
  </xsl:template>
</xsl:stylesheet>
```

Book4.xsl

Here's the output:

```
<?xml version="1.0" encoding="UTF-8"?>
<H1>The Right Stuff</H1>
```

Listen... we're producing HTML here...
Can we lose that funky <?xml> tag?

Output control

```
<xsl:stylesheet

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="book">
     <H1>
       <xsl:apply-templates select="title"/>
     </H1>
  </xsl:template>
</xsl:stylesheet>
```

Book6.xsl

Here's the output:

```
<H1>The Right Stuff</H1>
```

✓ Now we're done!

Summary of Basics

- <xsl:stylesheet> must surround all else, and requires a particular syntax to make it all work
- one or more templates are used to indicate what to generate for a given match, via literal result text/tags mixed with xsl directives
- XML data is processed as a "tree"
- apply-templates invokes rules for all children, but it can be applied selectively
- some built-in rules handle common requirements, so stylesheet is simpler (usually)
- for any given problem, there are usually a few ways to solve it, some better than others

Part 2: Generating A Complete Web Page

The new input XML data

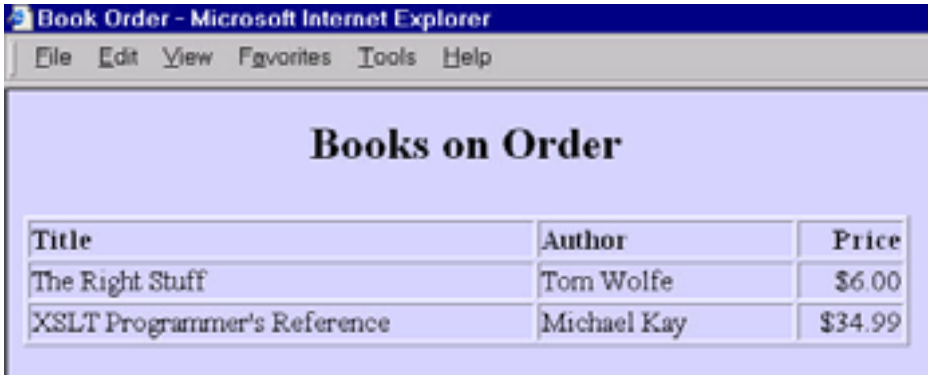
```
<book-order>
  <book>
    <author>Tom Wolfe</author>
    <title>The Right Stuff</title>
    <price>$6.00</price>
  </book>

  <book>
    <author>Michael Kay</author>
    <title>XSLT Programmer's Reference</title>
    <price>$34.99</price>
  </book>

</book-order>
```

Book1.xml

We added a second book description...

...so we also had to add an outer group to contain the book groups (XML rules)

The desired results



Book1.xml

Book7.xsl

Book7.html

What we need to do...

1. Generate prologue - start the <html> group, create <head> and <title> groups, start a <body>, set up the <table>, caption and column headings
   - ➡ all this must be first, and just once
2. Generate a row of data for each <book>, for any number of books in the source xml
3. Generate epilogue - close the <table> definition, close the <body> and <html> groups
   - ➡ this must be at the end, and just once

Technique: One-time content at beginning and end

```
<xsl:output method="html">
<xsl:template match="/">
  <!-- Begin:  content that precedes the data -->
       <b>This will precede the data.</b>
  <!-- End:    content that precedes the data -->
  <xsl:apply-templates/>
  <!-- Begin:  content that follows the data -->
       <b>This will follow the data.</b>
  <!-- End:    content that follows the data -->
</xsl:template>
```

This matches the ROOT node. There's only one root, and it's always the first node to be processed. We'll use it for once-only stuff at begin or end

For match="/", content before after before or after apply-templates will be beginning and end of the beginning or end of the result html, respectively

Our HTML file: prologue

```
<xsl:output method="html">
<xsl:template match="/">
   <html>
      <head><title>Book Order</title></head>
      <body bgcolor="#d0d0ff">
        <table border="border">
          <caption><h2>Books on Order</h2></caption>
          <thead>
            <tr>
              <th align="left">Title</th>
              <th align="left">Author</th>
              <th align="right">Price</th>
            </tr>
          </thead>
          <tbody>
<xsl:apply-templates/>
          <!-- see next page -->
</xsl:template>
```


Our HTML file: epilogue

```
<xsl:output method="html">
<xsl:template match="/">
   <!-- see previous page -->
<xsl:apply-templates/>
            </tbody>
          </table>
        </body>
      </html>
</xsl:template>
```

What we need to do...

- ✓ Generate prologue - start the <html> group, create <head> and <title> groups, start a <body>, set up the <table>, caption and column headings
  - ➡ all this must be first, and just once
- 2. Generate a row of data for each <book>, for any number of books in the source xml
- ✓ Generate epilogue - close the <table> definition, close the <body> and <html> groups
  - ➡ this must be at the end, and just once

Generating the table data rows

```
<xsl:template match="book">
          <!-- generate a table row for a book -->
          <tr>
            <td align="left" width="300">
              <xsl:value-of select="title"/>
            </td>
            <td align="left" width="150">
              <xsl:value-of select="author"/>
            </td>
            <td align="right" width="60">
              <xsl:value-of select="price"/>
            </td>
          </tr>
</xsl:template>
```

- This template matches each <book> group
- Each <book> generates one <tr> group & contents
- <xsl:value-of> selects data from the child nodes to be included in the table cells

Book1.xml
Book7.xsl
Book7.html

e-business

Part 3: Sorting and Numbering

e-business

Sorting

- Suppose we want to sort the book order by author's name.
- This can be done by adding a rule for <book-order>:

```
<xsl:template match="book-order">
  <xsl:apply-templates>
    <xsl:sort select="author"/>
  </xsl:apply-templates>
</xsl:template>
```

Books on Order

Title	Author	Price
XSLT Programmer's Reference	Kay, Michael	$34.99
The Right Stuff	Wolfe, Tom	$7.99
The Electric Kool-Aid Acid Test	Wolfe, Tom	$14.95

Book2.xml

Book8.xsl

Book8.html

<xsl:sort>

- Used inside <xsl:apply-templates> group
  - affects processing order only (no change to DOM)
  - (or <xsl:for-each>, as we'll see later)
- Example:

```
<xsl:sort select="author"/>
<xsl:sort select="price"
          order="descending"
          data-type="number"/>
```

Can sort on multiple elements by adding additional <xsl:sort>'s

Specify descending order (default: ascending)

Specify numeric sort (default: text)

Books on Order

Title	Author	Price
XSLT Programmer's Reference	Kay, Michael	34.99
The Electric Kool-Aid Acid Test	Wolfe, Tom	14.95
The Right Stuff	Wolfe, Tom	7.99

Book2.xml
Book9.xsl
Book9.html

Numbering

Suppose we need a numbered list like this

1 The Right Stuff
2 XSLT Programmer's Reference
3 The Electric Kool-Aid Acid Test

This is easy:

```
<xsl:template match="book">
  <br/>
    <xsl:number/>
    <xsl:text> </xsl:text>
     <xsl:value-of select="title"/>
</xsl:template>
```

This puts a blank between number and title

The <xsl:number> element inserts a sequence number for us, as if by magic.
Let's see how it works.

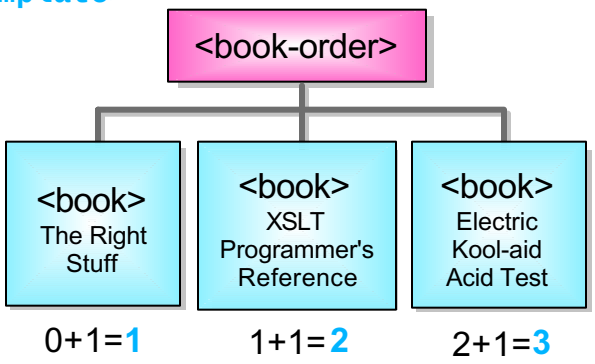
Book2.xml
Book10.xsl
Book10.html

<xsl:number>

Counts number of preceding siblings of the element in tree, adds 1, gives result as text

```
<xsl:template match="book">
  <br/>
    <xsl:number/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="title"/>
</xsl:template>
```

1 The Right Stuff
2 XSLT Programmer's Reference
3 The Electric Kool-Aid Acid Test



<xsl:number format="format ">

1 one	1. one	1) one	a. one	A. one	(i) one	I. one
2 dos	2. dos	2) dos	b. dos	B. dos	(ii) dos	II. dos
3 drei	3. drei	3) drei	c. drei	C. drei	(iii) drei	III. drei
4 quatre	4. quatre	4) quatre	d. quatre	D. quatre	(iv) quatre	IV. quatre
5 cinque	5. cinque	5) cinque	e. cinque	E. cinque	(v) cinque	V. cinque
"1"	"1. "	"1) "	"a. "	"A. "	"(i) "	"I. "

"ア"	Katakana numbering
"イ"	Katakana numbering in the "iroha" order
"๑"	numbering with Thai digits
"א" letter-value="traditional"	"traditional" Hebrew numbering
"ა" letter-value="traditional"	Georgian numbering
"α" letter-value="traditional"	"classical" Greek numbering
"а" letter-value="traditional"	Old Slavic numbering

files: numbers.xml, number*.xsl

Multi-part numbering

Consider a paper with chapters, sections, subsections, and appendices: Multi.xml

<xsl:number> can format multi-part section numbers, like this:

It can mix numbering styles in different parts of the numbers, too.

1 The first chapter
1.1 The first section
1.1.a First subsection
1.2 Second section
2 The second chapter
2.1 First section
2.2 Second section

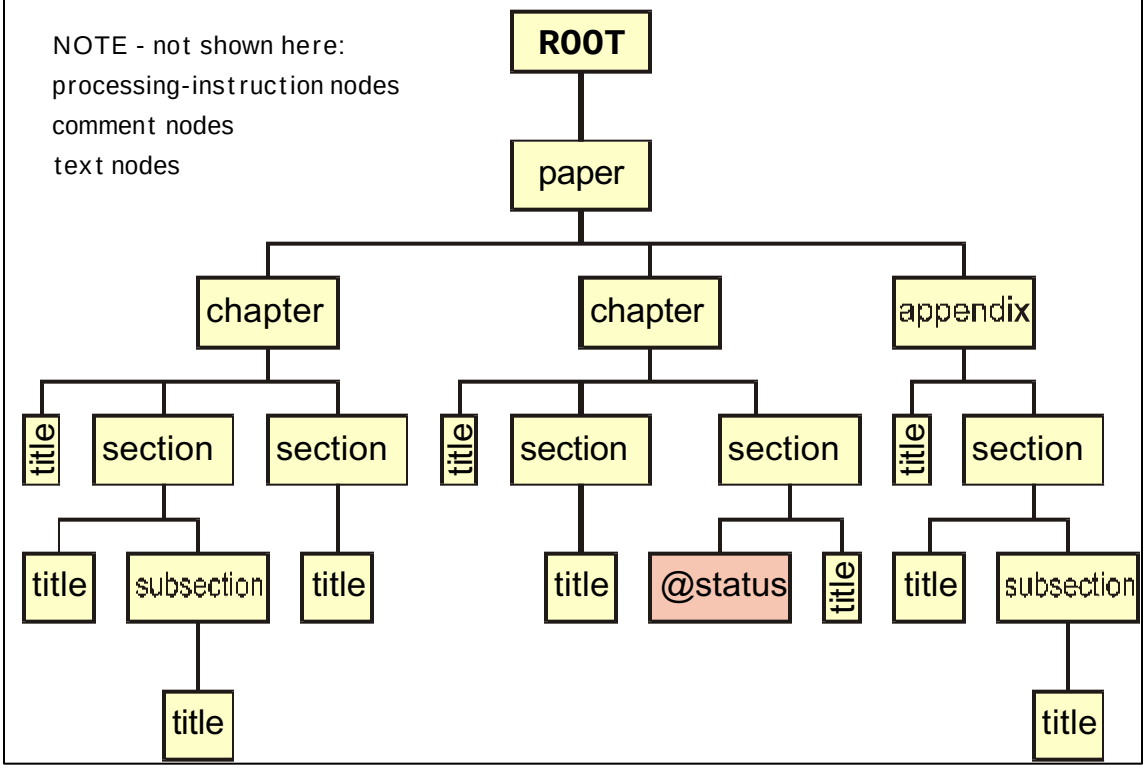
Multi-layered example: XML source

```
<?xml version='1.0'?>
<paper>
  <chapter>
    <title>The first chapter</title>
    <section>
      <title>The first section</title>
      <subsection><title>First subsection</title></subsection>
    </section>
    <section><title>Second section</title></section>
  </chapter>
  <chapter>
    <title>The second chapter</title>
    <section><title>First section</title></section>
    <section><title>Second section</title></section>
  </chapter>
  <appendix>
    <title>appendix</title>
    <section><title>Section in appendix</title>
      <!-- Mark, should there be a subsection here? -->
      <subsection><title>Subsection in appendix</title></subsection>
    </section>
  </appendix>
</paper>
```

Multi.xml

Multi-layered example: tree

NOTE - not shown here:
processing-instruction nodes
comment nodes
text nodes



Multi-part numbering stylesheet

```
<xsl:template match="title">
  <br/>
  <xsl:number level="multiple"
    count="chapter|section|subsection"
    format="1.1.a "/>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="appendix"/>
```

specify the nodes to be numbered

show the desired appearance

Tell xsl this is a multi-part number

gag <appendix> for now

Multi.xml
Multi0.xsl
Multi0.html

Adding the Appendix, etc

- This shows that you can use different rules for numbering different parts of the paper.
- Here's the XML source again: Multi.xml
- Here's what we'll change:

"Chapter" is in front of the number for chapters

There is a <p> space between chapter groups

Appendices are labelled A,B,C... instead of 1,2,3...

Chapter 1: The first chapter
1.1 The first section
1.1.a First subsection
1.2 Second section

Chapter 2: The second chapter
2.1 First section
2.2 Second section

Appendix A: appendix
A.1 Section in appendix
A.1.1 Subsection in appendix

<chapter> title presentation

```
<xsl:template match="chapter/title">
  <p/>Chapter
  <xsl:number level="multiple"
              count="chapter"
              format="1: "/>
  <xsl:apply-templates/>
</xsl:template>
```

This applies to <title>s of <chapter> groups only

output <p> break and the word "Chapter"

chapter number is followed by ':' for chapter title lines only

Note: chapter sections and subsections are not changed from previous example

<appendix> titles

match ANY <title> in <appendix> (including sections)

```
<xsl:template match="appendix//title">
  <xsl:if test="name(..)='appendix'">
    <p/>Appendix <xsl:number format="A: "/>
  </xsl:if>
  <xsl:if test="not(name(..)='appendix')">
    <br/><xsl:number level="multiple"
count="appendix|section|subsection"
                format="A.1.1 "/>
  </xsl:if>
  <xsl:apply-templates/>
</xsl:template>
```

almost same as for <chapter>

almost same as for <chapter> sections/subsections

Here we have one rule for appendix titles, and use <xsl:if> to distinguish two cases

Multi.xml

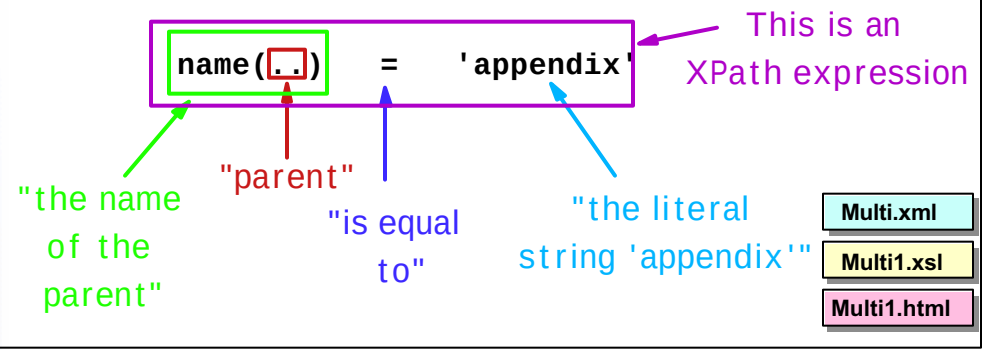
Multi1.xsl

Multi1.html

Part 4: Conditional Execution and Repetition

<xsl:if>: conditional code

```
<xsl:template match="appendix//title">
    <xsl:if test="name(..)='appendix'">
      <p/>Appendix <xsl:number format="A: "/>
    </xsl:if>
    <xsl:if test="not(name(..)='appendix')">
      <br/><xsl:number level="multiple"

count="appendix|section|subsection"
                format="A.1.1 "/>
    </xsl:if>
```



Multi.xml
Multi1.xsl
Multi1.html

<xsl:choose>: conditional code

```
<xsl:template match="appendix//title">
  <xsl:choose>
    <xsl:when test="name(..)='appendix'">
      <p/>Appendix <xsl:number format="A: "/>
    </xsl:when>
    <xsl:otherwise>
      <br/><xsl:number level="multiple"
                count="appendix|section|subsection"
                format="A.1.1 "/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Let's combine sort and number

```
<xsl:template match="book-order">
  <xsl:apply-templates>
    <xsl:sort select="title"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="book">
  <br/>
    <xsl:number/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="title"/>
</xsl:template>
```

3 The Electric Kool-Aid Acid Test
1 The Right Stuff
2 XSLT Programmer's Reference

The titles are sorted ascending...
but the numbers are wrong!

```
<book-order>
  <book>
    <author>Wolfe, Tom</au
    <title>The Right Stuf
    <price>7.99</price>
  </book>
  <book>
    <author>Kay, Michael</
    <title>XSLT Programmer
    <price>34.99</price>
  </book>
  <book>
    <author>Wolfe, Tom</au
    <title>The Electric K
    <price>14.95</price>
  </book>
</book-order>
```

Book2.xml
Book11.xsl
Book11.html

<xsl:for-each> and <xsl:sort>

```
<xsl:template match="book-order">
  <xsl:for-each select="book">
    <xsl:sort select="title"/>
    <br/>
      <xsl:number value="position()"
                  format="1. "/>
      <xsl:value-of select="title"/>
  </xsl:for-each>
</xsl:template>
```

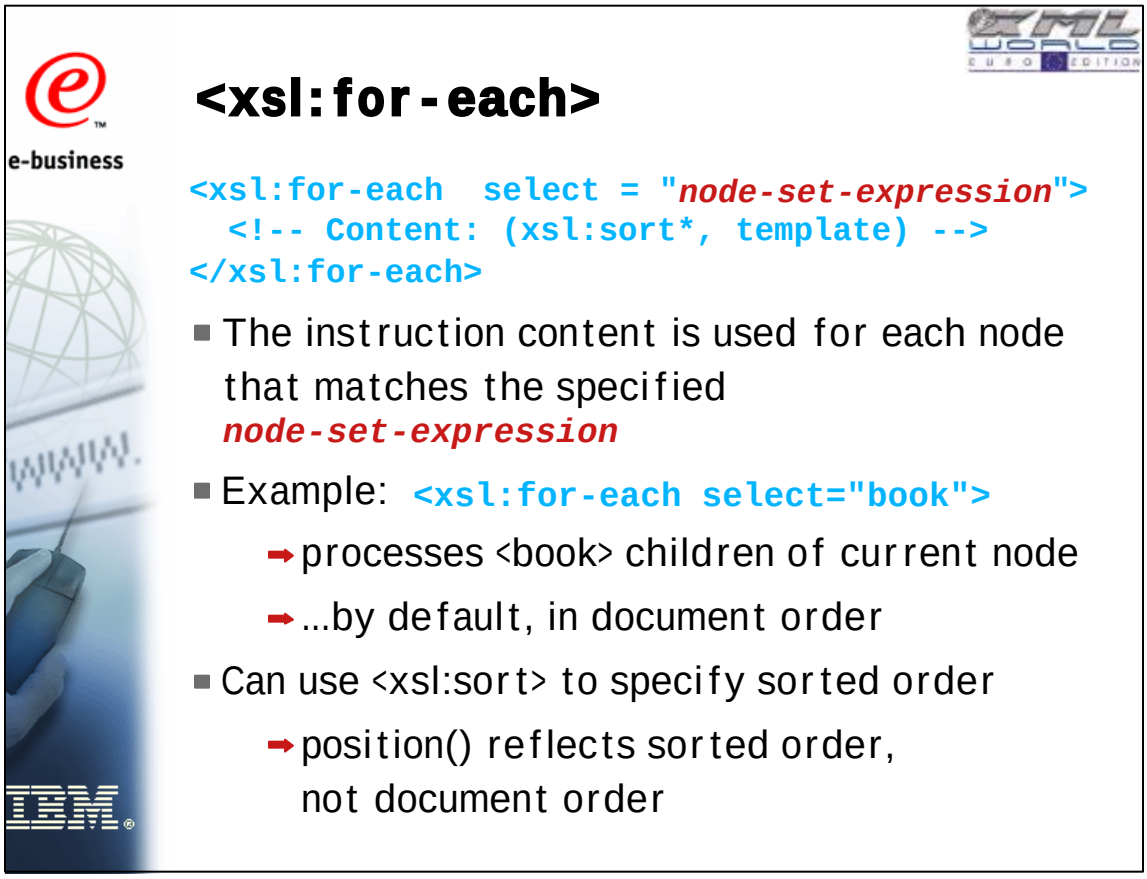
1. The Electric Kool-Aid Acid Test
2. The Right Stuff
3. XSLT Programmer's Reference

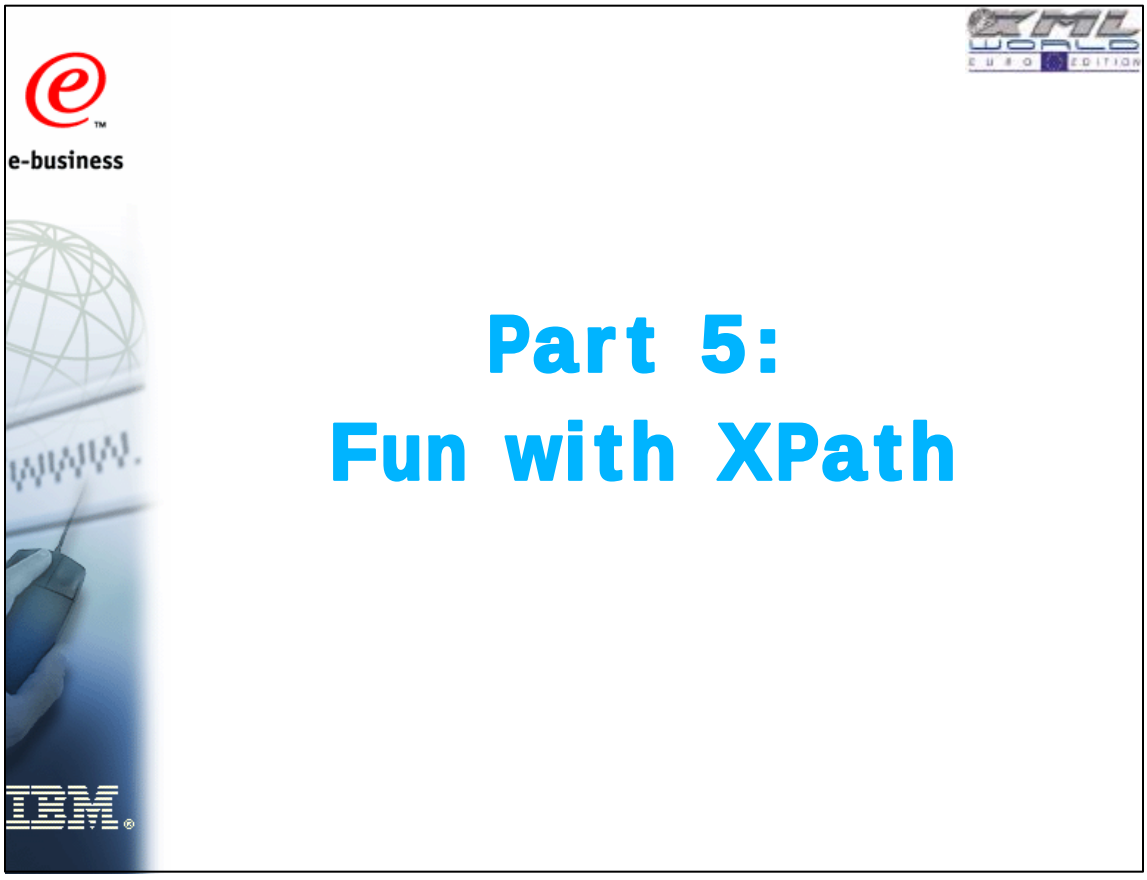
✓ The titles are ascending, so are the numbers.

```
<book-order>
  <book>
    <author>Wolfe, Tom</a
    <title>The Right Stu
    <price>7.99</price>
  </book>
  <book>
    <author>Kay, Michael<
    <title>XSLT Programme
    <price>34.99</price>
  </book>
  <book>
    <author>Wolfe, Tom</a
    <title>The Electric
    <price>14.95</price>
  </book>
</book-order>
```

Book2.xml
Book12.xsl
Book12.html

back


e-business
<xsl:for-each select = "node-set-expression">
<!-- Content: (xsl:sort*, template) -->
</xsl:for-each>
The instruction content is used for each node that matches the specified node-set-expression
Example: <xsl:for-each select="book">
processes <book> children of current node
...by default, in document order
Can use <xsl:sort> to specify sorted order
position() reflects sorted order, not document order
IBM


e-business
Part 5:
Fun with XPath
IBM

What is XPath?

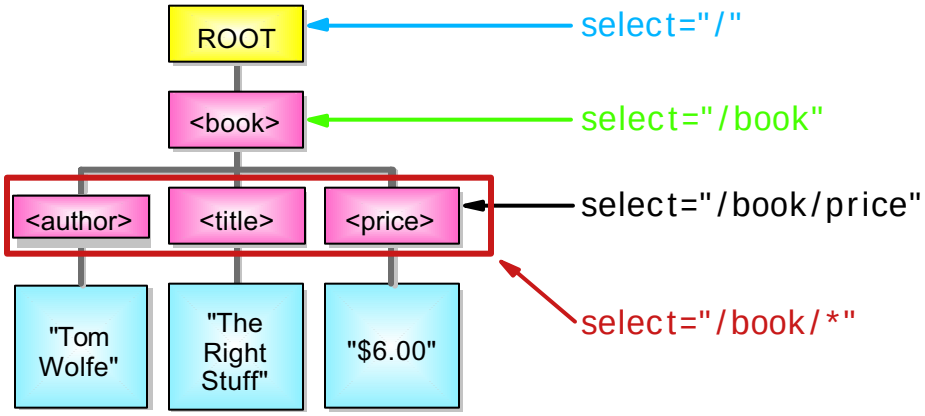
- a language for addressing parts of an XML doc represented as tree of DOM nodes
- separate specification: designed to be used by XSLT, XPointer, simple queries, ...
- compact, non-XML syntax, can be used in URI's and XML attributes
- has a natural subset that can be used for pattern matching (e.g., in XSLT)
  - ➡ we'll focus on features useful in XSLT

Why is it called XPath?

- Expressions describe a path to a given node or set of nodes
- Consider the DOS, Unix, or URI syntax for addressing files in a directory structure
  - /publications/articles/Transformations.xml
  - this is called a pathname to the file
  - it describes the path to follow, from the "root", thru a tree of directories (folders), to locate a given file
- ➡ XPath also uses a forward slash to separate components (nodes) of a path

Absolute Addressing examples

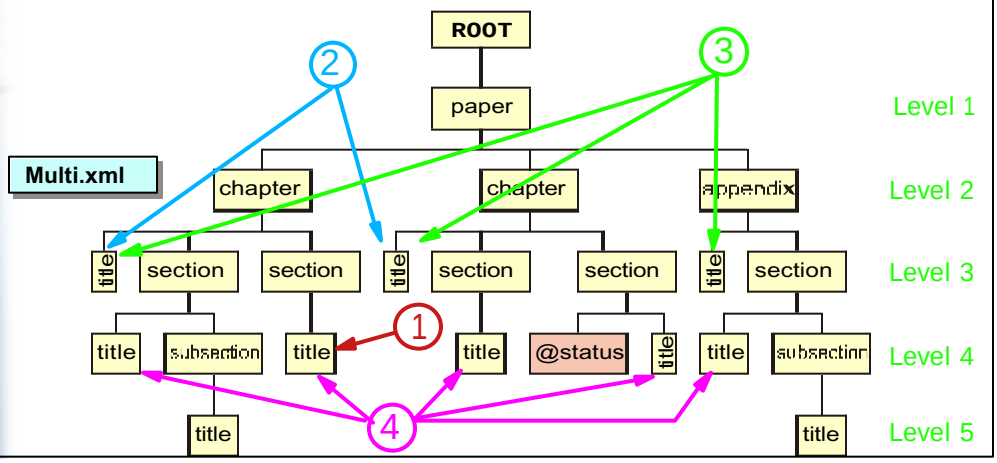
```
<book>
   <author>Tom Wolfe</author>
   <title>The Right Stuff</title>
   <price>$6.00</price>
</book>
```
Book0.xml



e.g. <xsl:value-of select="/book/price"> returns $6.00

Absolute Addressing: variations

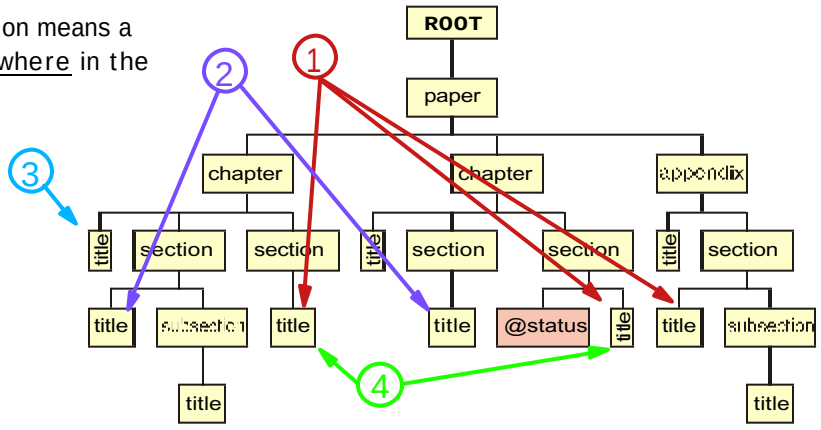
1. /paper/chapter[1]/section[2]/title — title for section 1.2
2. /paper/chapter/title — titles for all chapters
3. /paper/*/title — titles three levels down from root
4. /paper//section/title — titles for all sections, any level
5. //title — all titles anywhere in document



Absolute addressing w/predicates

1. `//section[last()]/title` titles for last sections
2. `//section[last()-1]/title` ...second-to-last sections
3. `//section[title='The first section']/title`
   titles for all section that have the specified title
4. `//section[position() mod 2 = 0]/title`
   titles for all even-numbered sections

Note: //section means a <section> anywhere in the document



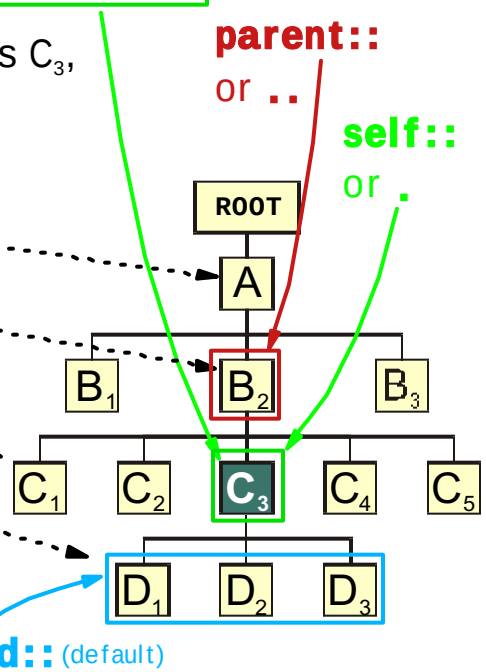
Multi.xml

Relative Addressing

```
<xsl:template match="/A/B[2]/C[3]">
```

in a template that matches C_3,
i.e. relative to C_3:

"../.."
".."
"../C[1]"
"D[1]"



e-business

XPath in a Nutshell

axis::nodetest[predicate]

ancestor
ancestor-or-self
attribute ("@")
child (default)
descendant
descendant-or-self
following
following-sibling
preceding
preceding-sibling
namespace
parent ("..")
self (".")

name <name> node
* any element node
comment() <!comment> nodes
text()
node()
processing-instruction()
processing-instruction('foo')
namespace:name <name> in ns
namespace:* any element in ns

[1] first node
[last()] last node
[position() mod 2 = 0] even nodes
speaker[firstname="Mark"]

e-business

XPath Features

- URL-like path selection
- Axis traversal semantics
- Predicates
- Basic data types: number, boolean, string, node-set
- Path expressions
- Math, string, boolean expressions
- Access by index
- Namespaces
- Variable use (but not declaration)
- Extension mechanism

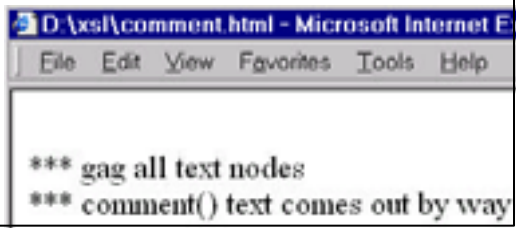
Processing comments (!!)

XSL even lets you process <!-- comments --> from an XML document!

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="comment()">
    <br/>*** <xsl:value-of select="."/>
  </xsl:template>
  <!-- gag all text nodes -->
  <!-- comment() text comes out by way of
  xsl:value-of above -->
  <xsl:template match="text()
</xsl:stylesheet>
```

XPath expression to match comment nodes

This is the output when you run this stylesheet on itself



Part 6: XML-to-XML Vocabulary Translation

<term>Vocabulary</term>

- A definition of
  - the names in the tags
  - the arrangement of the tags (structure)
  - the types, default values, valid values
- Written as:
  - Now: DTD (Document Type Definition)
  - Soon: XML Schema (adds type support)
- Industry-specific vocabularies are developed by cooperating companies, then registered at XML.ORG to become standard.

The source XML document

```
<company>
    <div no="7a">
        <dept no="42">
            <emp no="123456" name="Whoptimone, Ida"/>
            <emp no="651432" name="Tirebiter, George"/>
        </dept>
        <dept no="51">
            <emp no="832953" name="Danger, Nick"/>
        </dept>
    </div>
    <div no="2b">
        <dept no="57">
            <emp no="283412" name="Boss, Yuda"/>
        </dept>
    </div>
</company>
```

company.xml

The required XML result document

```
<company>
    <employee id="123456">
        <name>Whoptimone, Ida</name>
        <division>7a</division>
        <department>42</department>
    </employee>
    <employee id="651432">
        <name>Tirebiter, George</name>
        <division>7a</division>
        <department>42</department>
    </employee>
    <employee id="832953">
        <name>Danger, Nick</name>
        <division>7a</division>
        <department>51</department>
    </employee>
    <employee id="283412">
        <name>Boss, Yuda</name>
        <division>2b</division>
        <department>57</department>
    </employee>
</company>
```

What changed:

1. Restructured. Was a company list by div, dept, & emp; now a company list by employee.
2. Data from most attributes is now in sub-elements
3. Element and attribute names

employee.xml

back

The Stylesheet

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
 <xsl:template match="/">
  <xsl:element name="company">
   <xsl:for-each select="/company/*/*/emp">
    <xsl:element name="employee">
     <xsl:attribute name="id">
      <xsl:value-of select="@no"/>
     </xsl:attribute>
     <name><xsl:value-of select="@name"/></name>
     <xsl:element name="division">
      <xsl:value-of select="../../@no"/>
     </xsl:element>
     <department>
      <xsl:value-of select="../@no"/>
     </department>
    </xsl:element>
   </xsl:for-each>
  </xsl:element>
 </xsl:template>
</xsl:stylesheet>
```

employee.xsl

<xsl:element>

- Creates an element in the result tree
- Content (inside xsl:element) can be
  - <xsl:attribute> (create attribute)
  - <xsl:element> (create child element)

```
<xsl:element name="element-name">
  <!-- content: attributes, child elements-->
</xsl:element>
```

- Alternative: <element-name> literal result text

```
<element-name>
  <!-- content: attributes, child elements-->
</element-name>
```

back

<xsl:attribute>

- Creates an attribute in the result tree
- Content is the text value for the attribute

```
<xsl:attribute name="attribute-name">
  <!-- content: text value -->
</xsl:attribute>
```

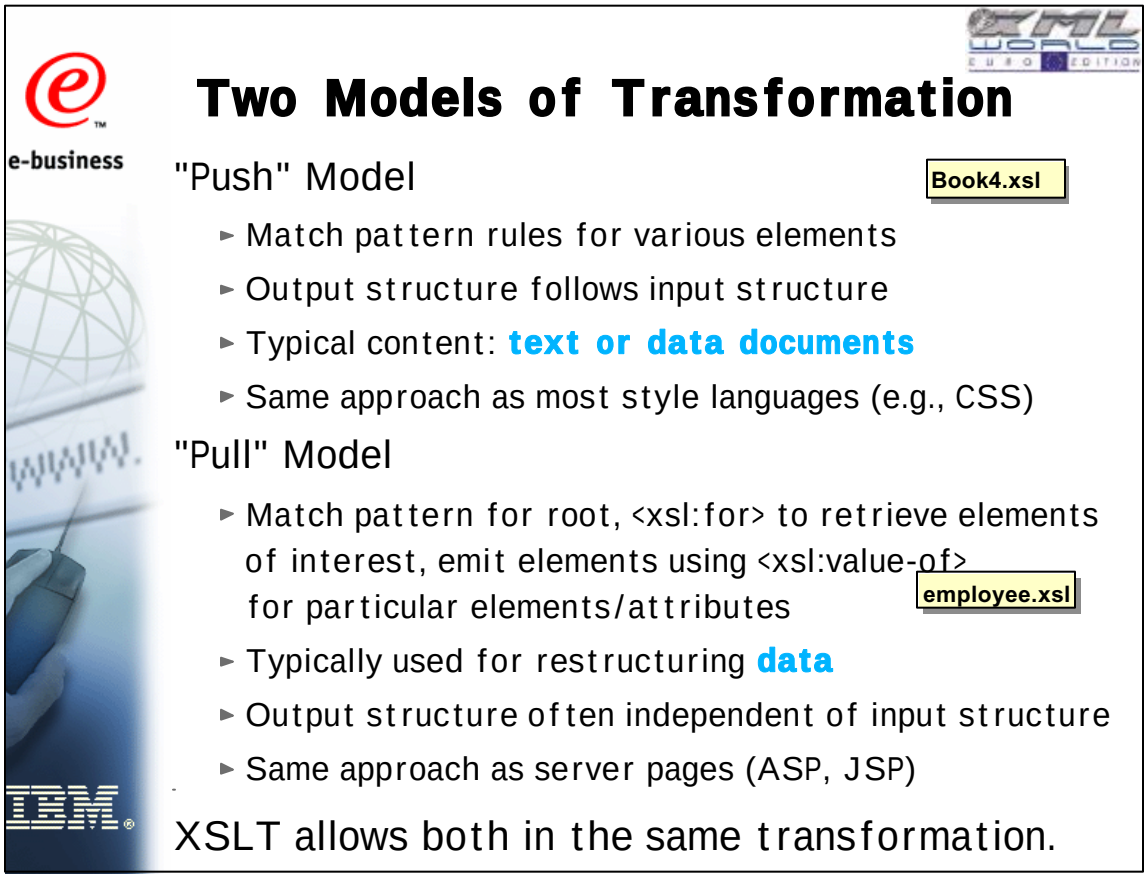
- Example:

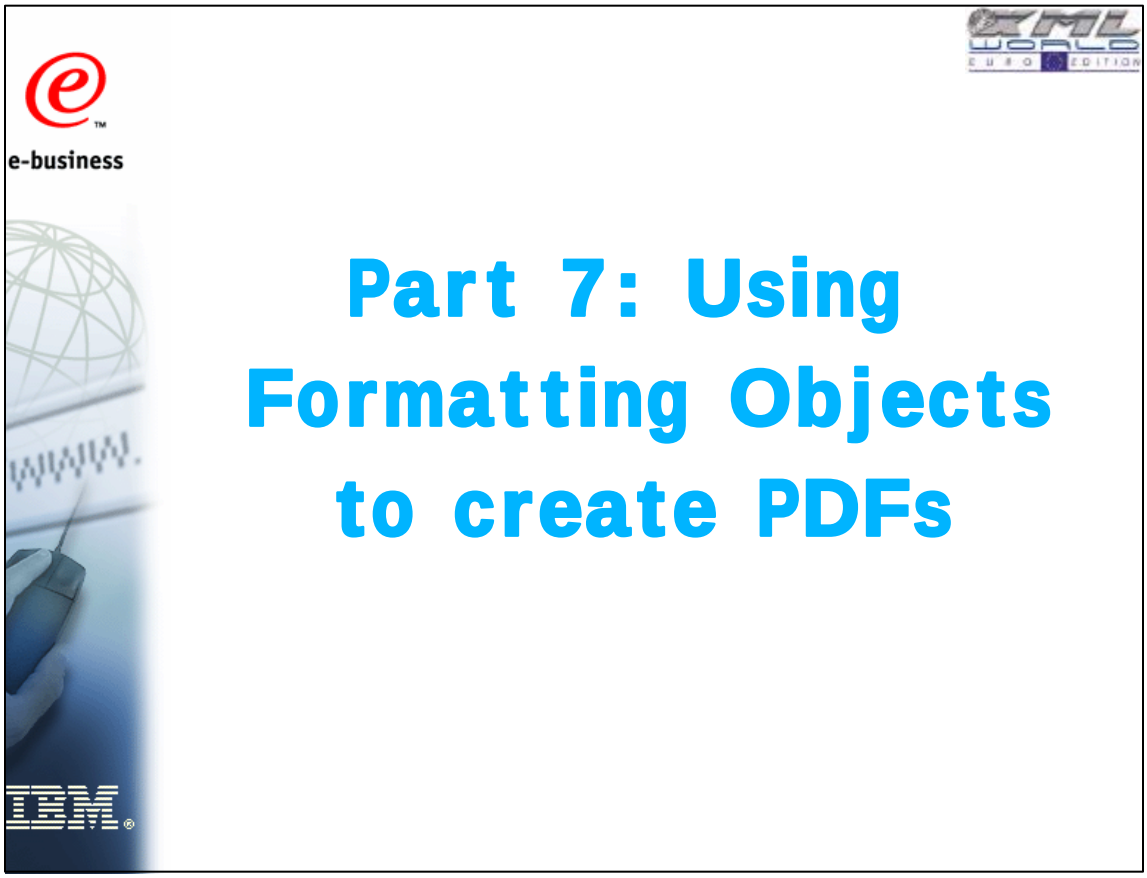
```
<xsl:attribute name="id">
  <xsl:value-of select="@no"/>
</xsl:attribute>
```

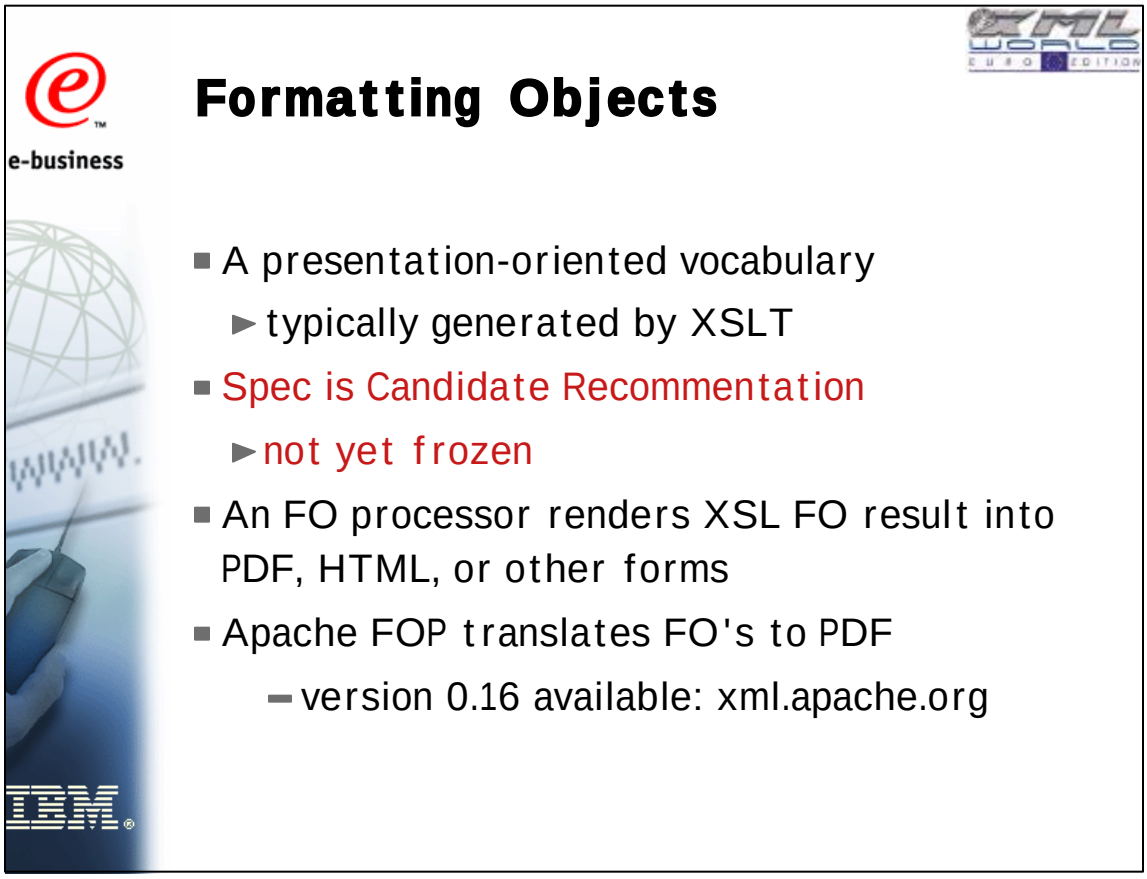
create an attribute named "id"

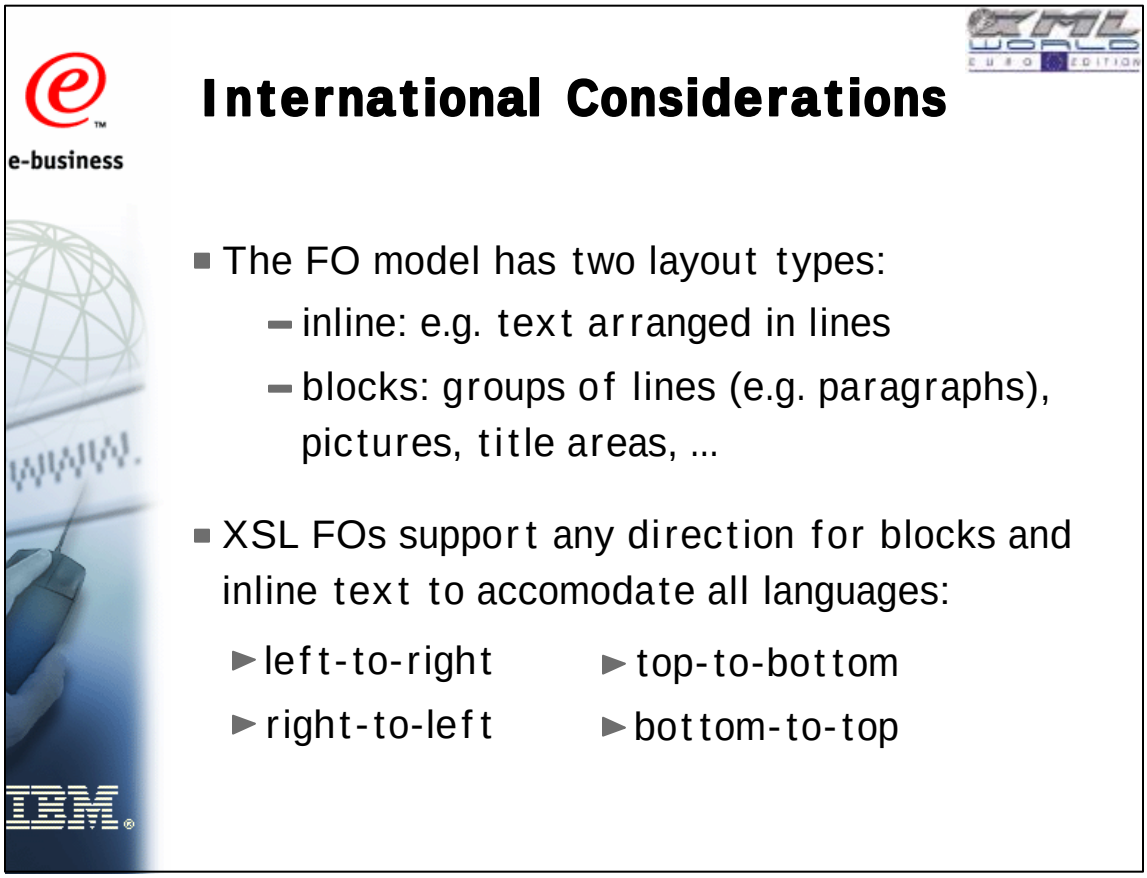
XPath: attribute "no" of current node

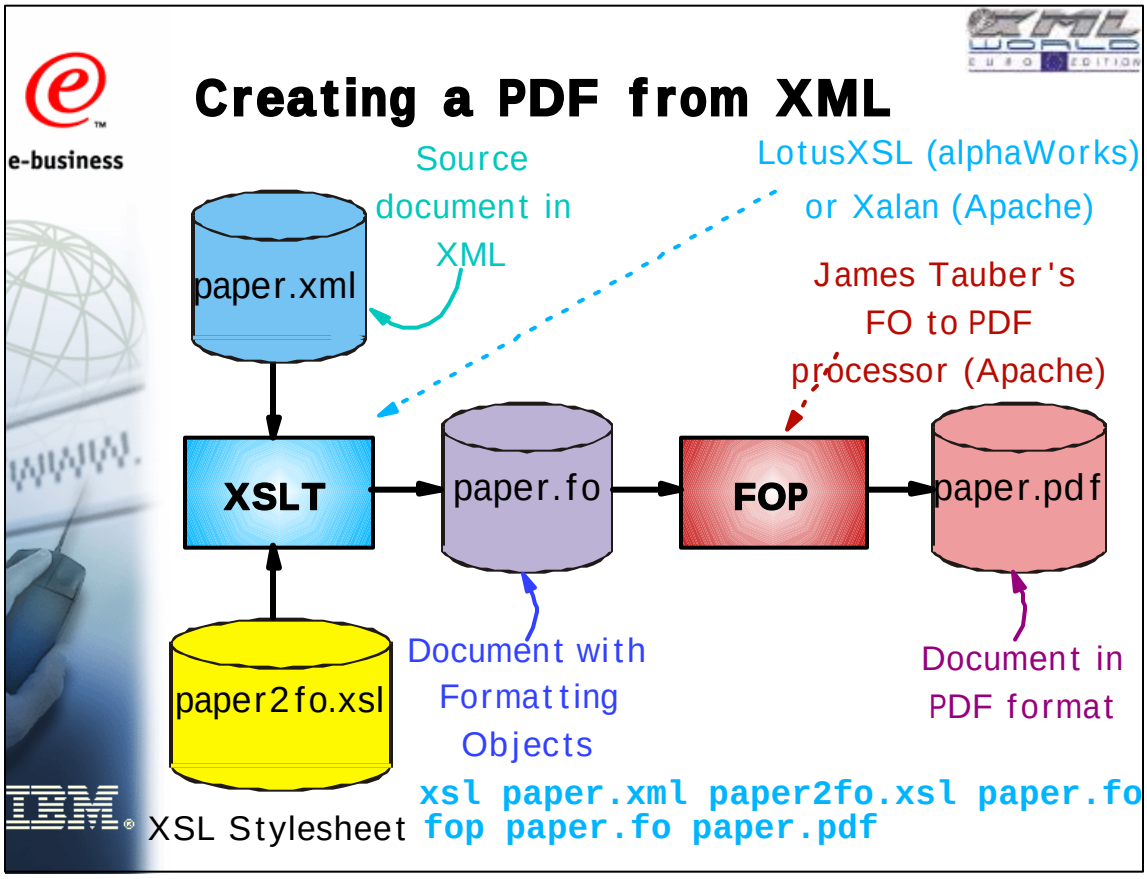
the value of the id attribute is the value of attribute "no" of the current node


e-business
Two Models of Transformation
"Push" Model
Book4.xsl
Match pattern rules for various elements
Output structure follows input structure
Typical content: text or data documents
Same approach as most style languages (e.g., CSS)
"Pull" Model
Match pattern for root, <xsl:for> to retrieve elements of interest, emit elements using <xsl:value-of> for particular elements/attributes
employee.xsl
Typically used for restructuring data
Output structure often independent of input structure
Same approach as server pages (ASP, JSP)
XSLT allows both in the same transformation.
IBM


e-business
Part 7: Using Formatting Objects to create PDFs
IBM

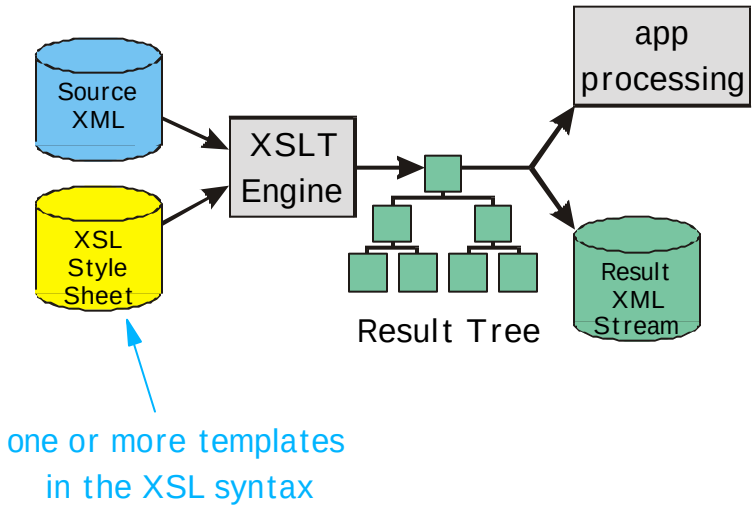

e-business
Formatting Objects
A presentation-oriented vocabulary
typically generated by XSLT
Spec is Candidate Recommentation
not yet frozen
An FO processor renders XSL FO result into PDF, HTML, or other forms
Apache FOP translates FO's to PDF
version 0.16 available: xml.apache.org
IBM


e-business
International Considerations
The FO model has two layout types:
inline: e.g. text arranged in lines
blocks: groups of lines (e.g. paragraphs), pictures, title areas, ...
XSL FOs support any direction for blocks and inline text to accomodate all languages:
left-to-right
right-to-left
top-to-bottom
bottom-to-top
IBM

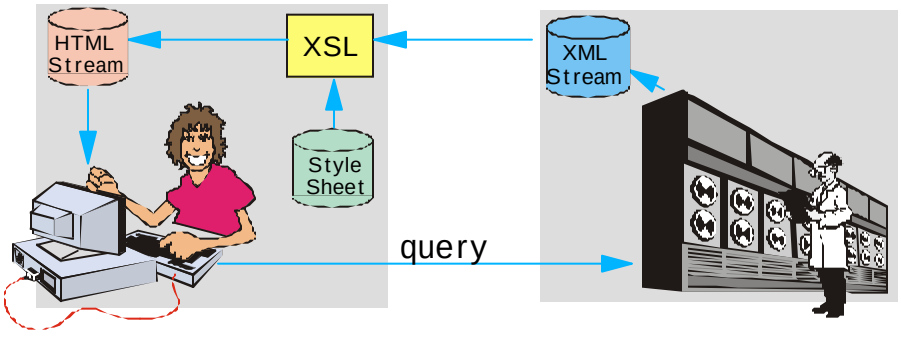

Creating a PDF from XML
e-business
Source document in XML
LotusXSL (alphaWorks) or Xalan (Apache)
James Tauber's FO to PDF processor (Apache)
paper.xml
XSLT
paper.fo
FOP
paper.pdf
paper2fo.xsl
Document with Formatting Objects
Document in PDF format
XSL Stylesheet
xsl paper.xml paper2fo.xsl paper.fo
fop paper.fo paper.pdf

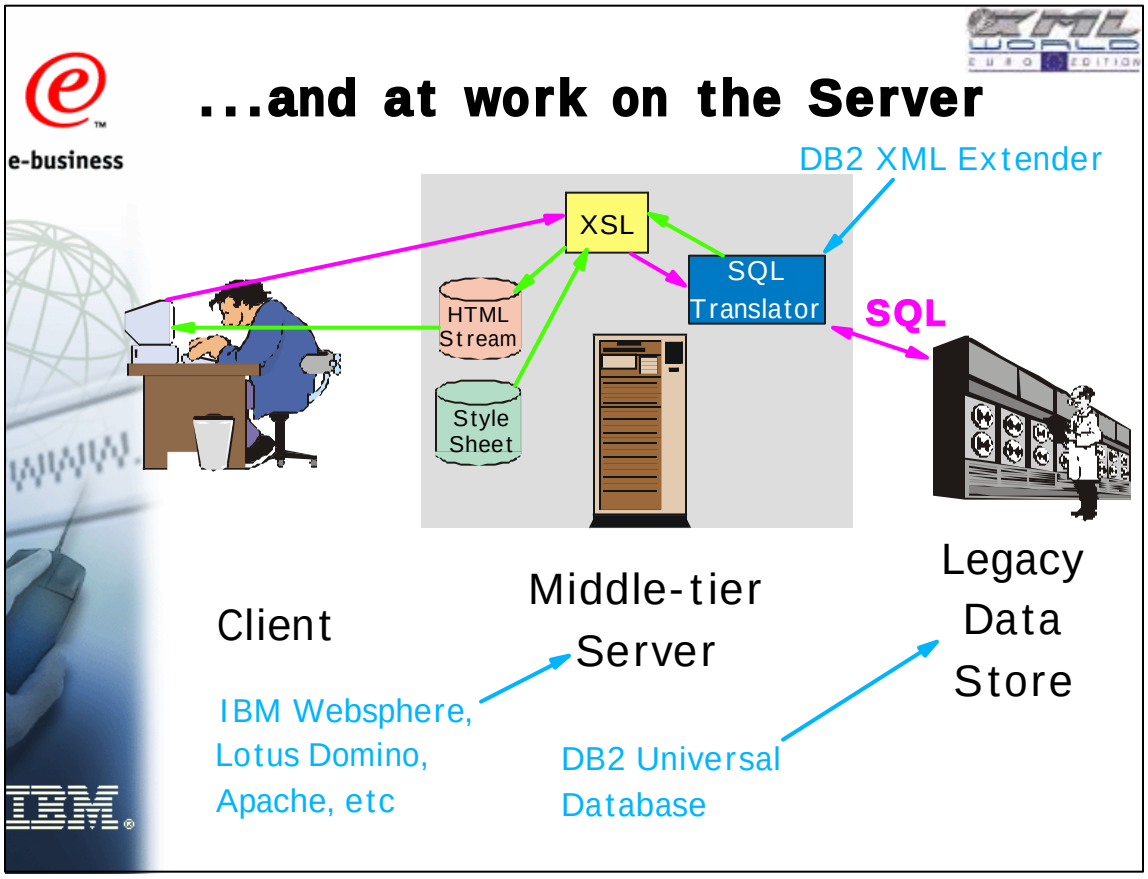

e-business
XSL in Application Architectures

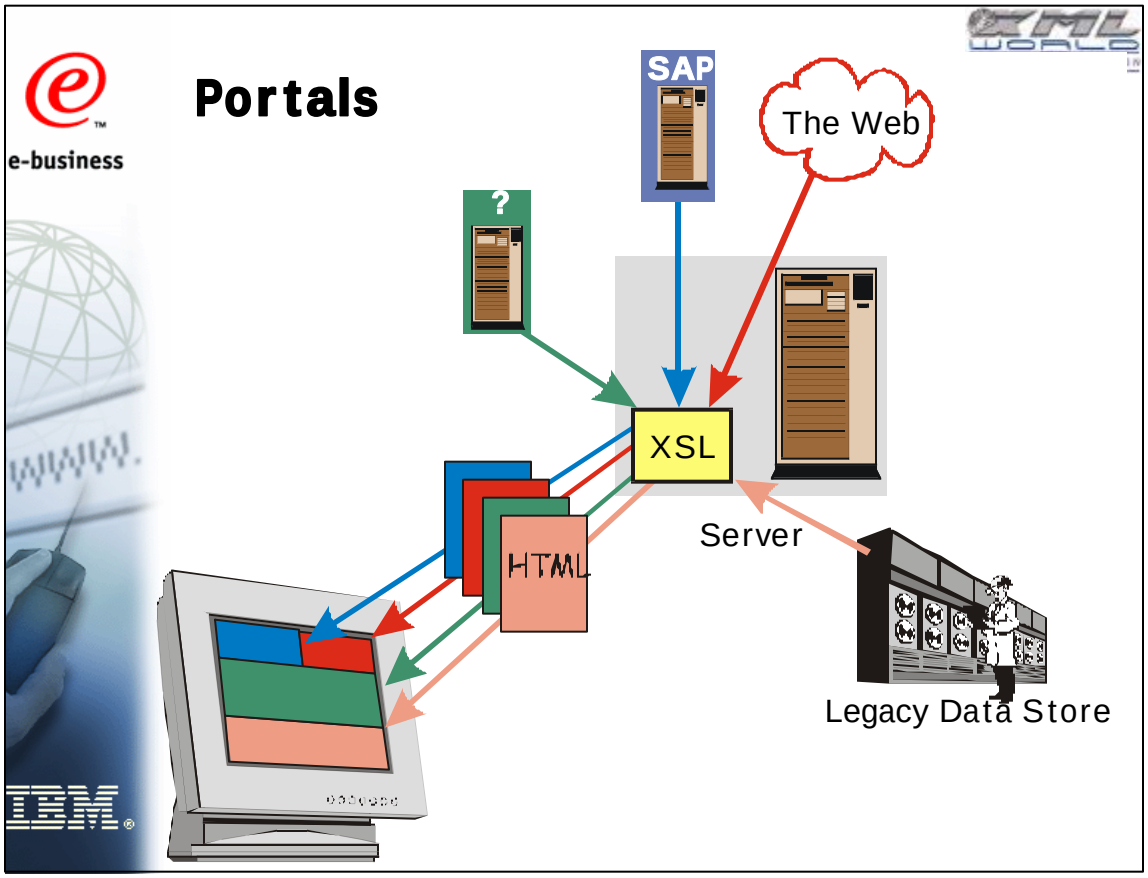
XSL Processing Overview



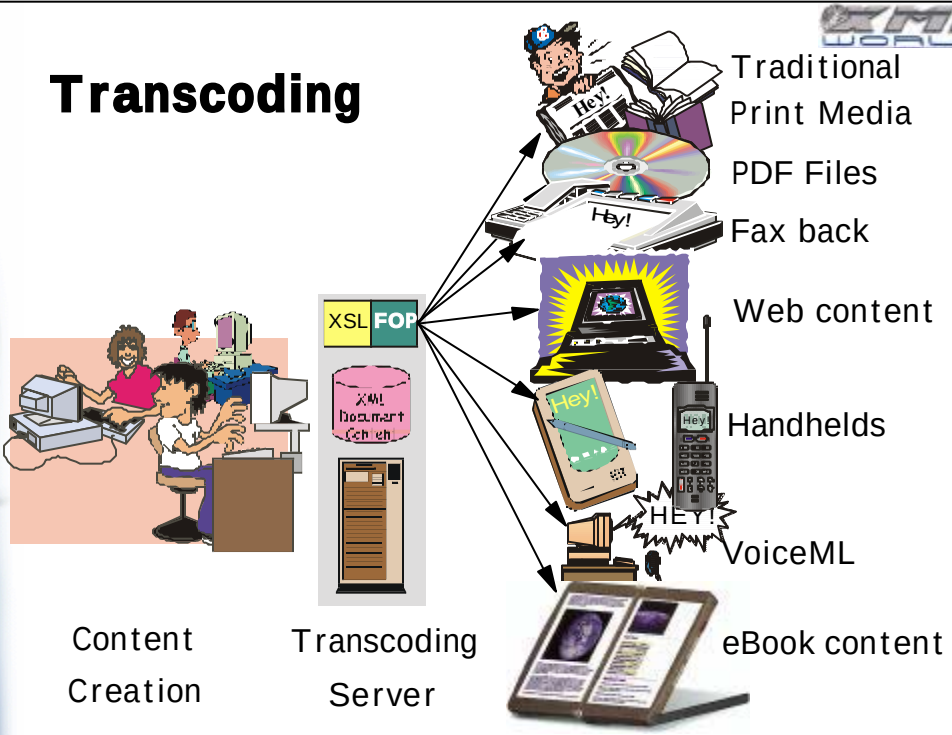
XSL is at home on the Client...




...and at work on the Server
e-business
DB2 XML Extender
XSL
SQL Translator
SQL
HTML Stream
Style Sheet
Client
Middle-tier Server
Legacy Data Store
IBM Websphere, Lotus Domino, Apache, etc
DB2 Universal Database

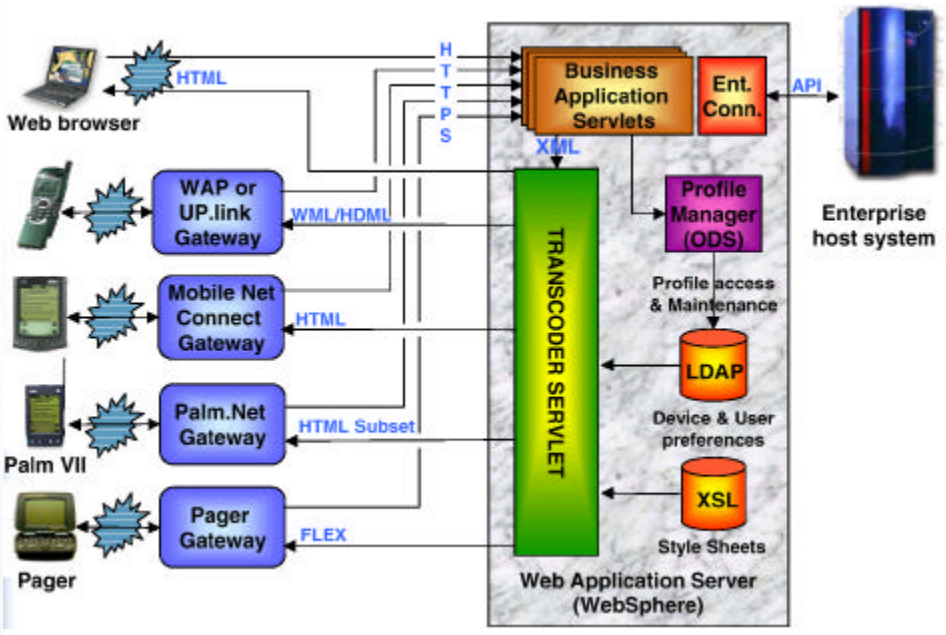

Portals
e-business
SAP
The Web
?
XSL
HTML
Server
Legacy Data Store

Transcoding

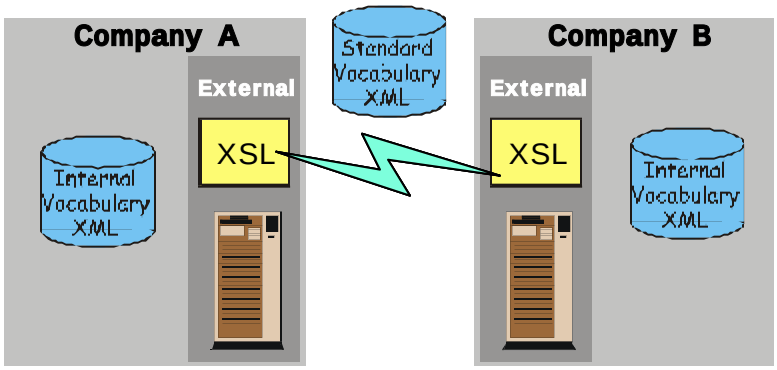


SABRE/Nokia Architecture

using WebSphere Transcoding Publisher

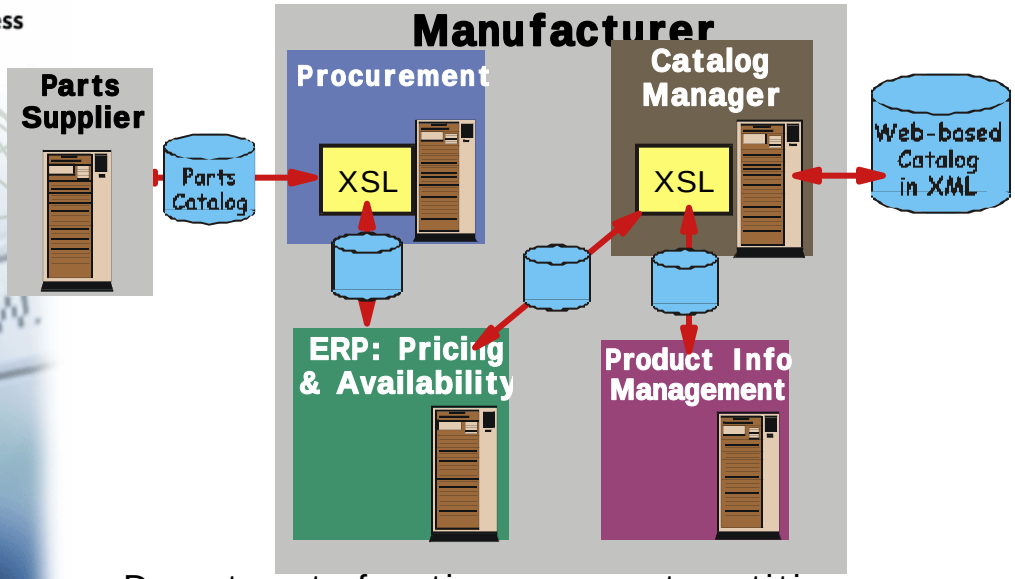


Business-to-Business



- XSL at external server: internal to standard vocabulary transformation

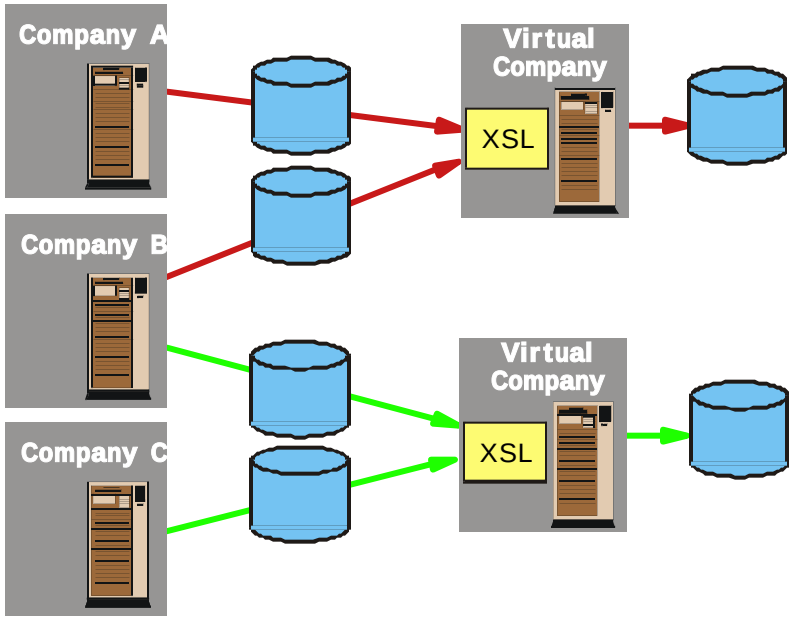
Application Integration



- Departments function as separate entities
- Vocabularies may need XSL transformation for integration with catalog producer

e-business

Business Integration



IBM

e-business

Part 8:
XSL Development

IBM

Developing with XSL

- download / install latest JDK (1.1.x or 1.3)
  - ➡ http://ibm.com/java/jdk/ (IBM's JDK 1.3)
  - ➡ http://java.sun.com/j2se/ (Sun's JDKs)
- download / install latest Xalan / LotusXSL
  - ➡ http://xml.apache.org/xalan/index.html
  - ➡ http://www.alphaworks.ibm.com
- java org.apache.xalan.xslt.Process -in {xml-in} -xsl {xsl-in} -out {output-file} -indent 3
- run the batch / shell file to execute XSL*

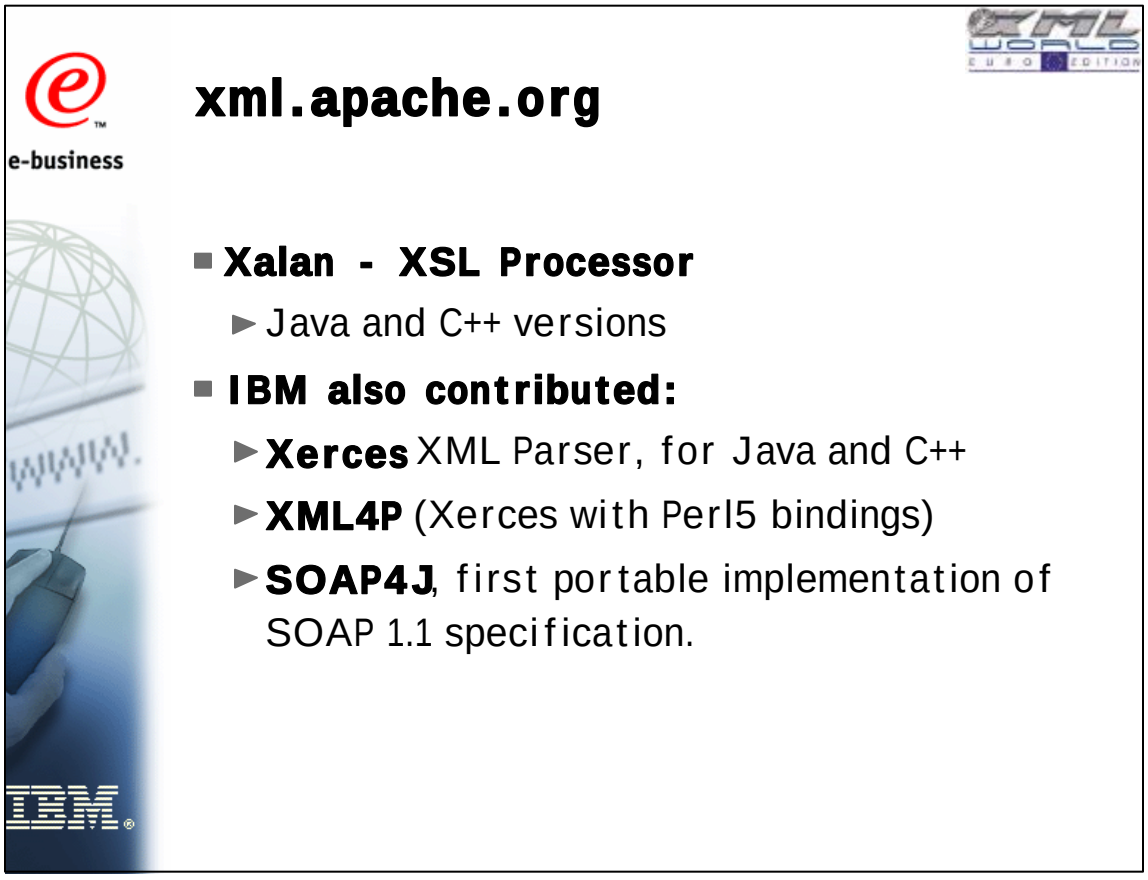
* Note: batch file, and source xml and xsl files for this class are available for free download from my website: http://ibm.com/software/developer/speaker/colan

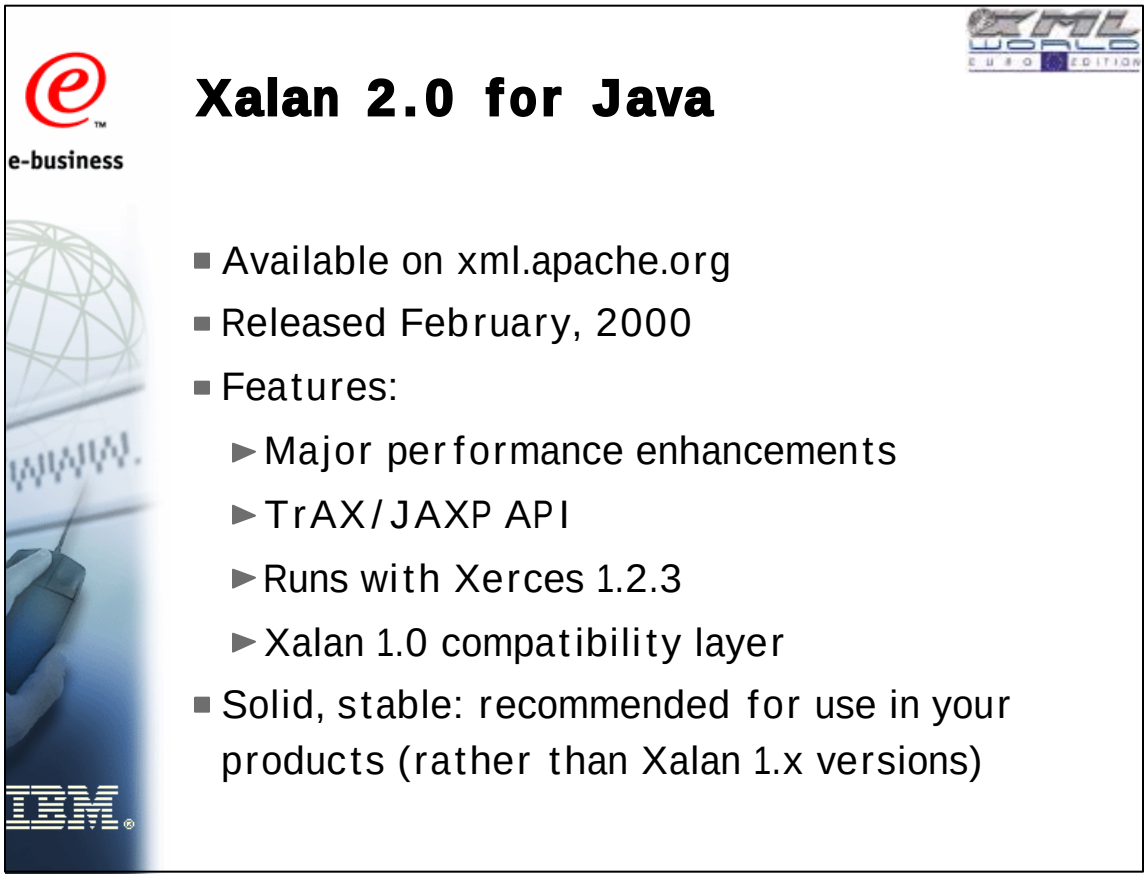
  - ➡ Grab a revised PDF of this presentation, too!

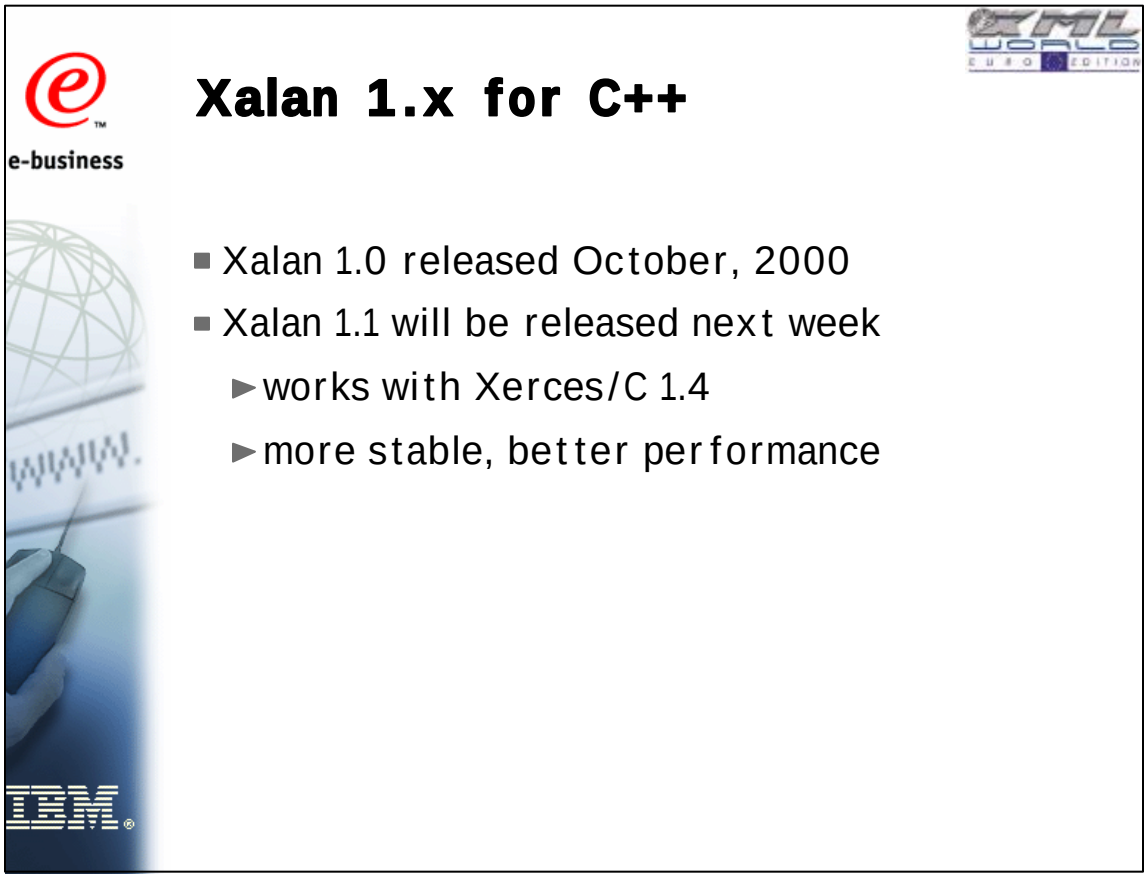
Tips on Developing with XSL

- For web pages to be created with XSL,
  - ▶ start by creating a working HTML prototype
  - ▶ then copy parts of it into stylesheet
- For vocabulary translation,
  - ▶ make sure you know the exact meaning and format for each element in both vocabularies
  - ▶ make a list of data not present in the source
- When creating PDFs with Formatting Objects
  - ▶ create a prototype in a page layout program
- Check out new XSL editor / trace tools coming from alphaworks.ibm.com


e-business
Tools and Resources:
XSL Programming
IBM


e-business
xml.apache.org
Xalan - XSL Processor
Java and C++ versions
IBM also contributed:
Xerces XML Parser, for Java and C++
XML4P (Xerces with Perl5 bindings)
SOAP4J, first portable implementation of SOAP 1.1 specification.
IBM


e-business
Xalan 2.0 for Java
Available on xml.apache.org
Released February, 2000
Features:
Major performance enhancements
TrAX/JAXP API
Runs with Xerces 1.2.3
Xalan 1.0 compatibility layer
Solid, stable: recommended for use in your products (rather than Xalan 1.x versions)
IBM


e-business
Xalan 1.x for C++
Xalan 1.0 released October, 2000
Xalan 1.1 will be released next week
works with Xerces/C 1.4
more stable, better performance
IBM

Other XSL Processors

- XT (by James Clark, now fading away)
- SAXON (from Michael Kay, author of **XSLT Programmer's Reference**, Wrox Press)
- IE5 (Microsoft)
- Oracle XML Dev Kit (Java, C++)
- iXSLT (infoteria, C++)
- EZ/X (Activated Intelligence, Java)
- XML::XSLT (Geert Josten, Perl)
- XPort (TIMELUX, C++ COM Object)

Formatting Objects Processors

- Apache: FOP version 0.16
  - create a PDF from XSL:FO document
  - supports XSL Candidate Recommendation
  - not yet a complete implementation
  - xml.apache.org

XSL Resources from IBM

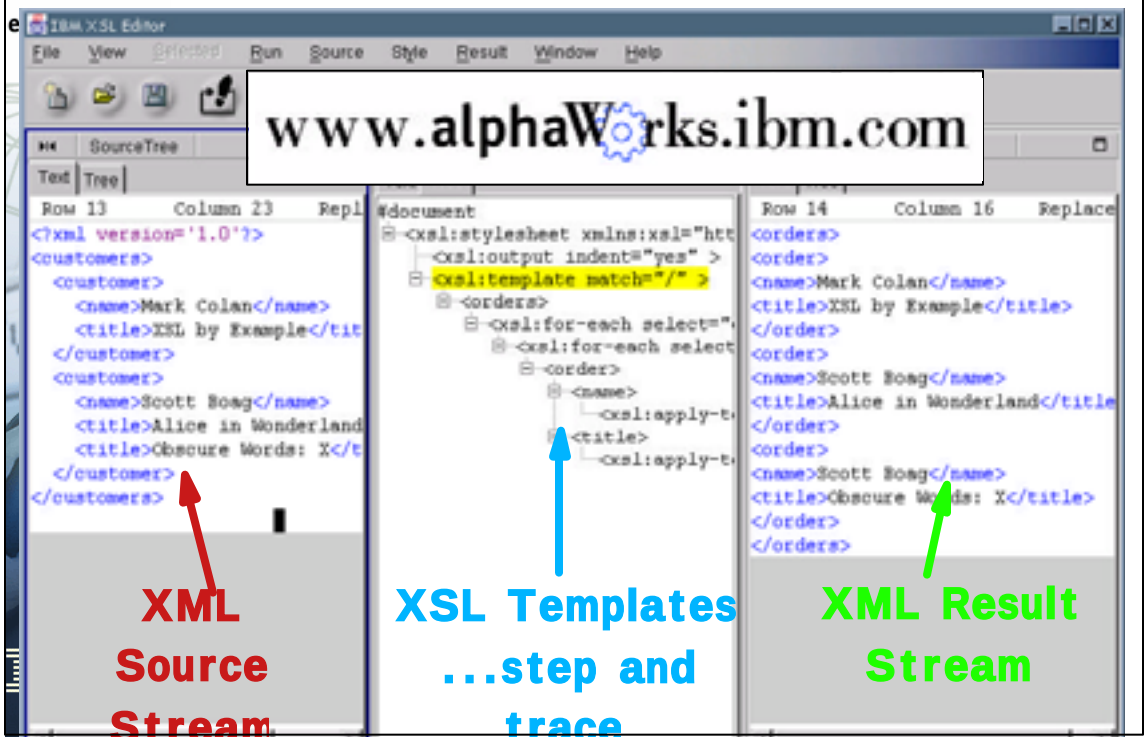
- **ibm.com**/alphaworks
  - XML processors, tools, editors, diff tools
  - XSLT processor, editor, trace
- **ibm.com**/developerWorks/xml
  - articles, tutorials, source code
  - several tutorials on XSL, more to come

    www6.software.ibm.com/reg/xml/transformxml-i
- **ibm.com**/developerWorks/speakers/colan
  - XSL by Example presentation, companion files
  - other presentations on XML and Web Services
- xml.apache.com - Xalan XSL Processor

XSL Tools on Alphaworks ibm.com/alphaworks

- LotusXSL - Apache Xalan wrapped with
  - Domino, DB2, generic db access features
  - the older LotusXSL API
- XSL Editor (version 2.0 posted 1/13/00)
- XSL Trace
- Visual Transformation
- many other XML tools, software aids, etc
  - XML Master (XMas) - generate custom Beans for working with a particular XML document from DTD
  - VoiceXML - industry-defined voice markup language

XSL Editor 2.0



XSLT Resources

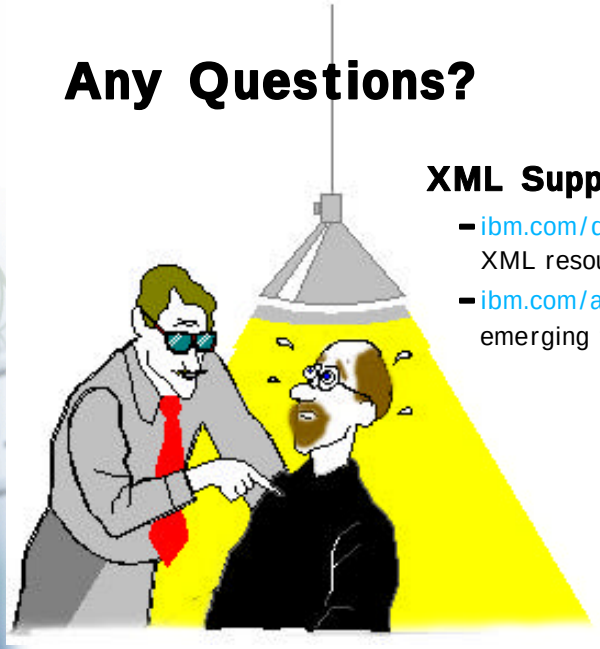
e-business

- W3C Specifications (XSL, XSLT, XPath)
  - ► http://www.w3.org/Style/XSL/
- XSLT Programmer's Reference by Michael Kay, WROX Press
- XSL Companion by Neil Bradley, Addison-Wesley
- Chapter 14 of XML Bible by Elliotte Rusty Harold
  - ► IDG Books, July 1999, also available on web at:
  - ► metalab.unc.edu/xml/books/bible/updates/14.html
- Practical Transformation Using XSLT and XPath
  - ► training materials, Ken Holman, Crane Softwrights Ltd
  - ► http://www.cranesoftwrights.com/training/#ptux
- www.mulberrytech.com/xsl/xsl-list/
  (mailing list for general XSL questions)
- www.xslt.com - various XSL-specific references

XSL Formatting Objects resources

- W3C Specifications (XSL, XSLT, XPath)
  - http://www.w3.org/Style/XSL/
- Chapter 15 of XML Bible by Elliotte Rusty Harold
  - IDG Books, July 1999, also available on web at:
  - www.ibiblio.org/xml/books/bible/updates/15.html
- FOP processor: xml.apache.org

Any Questions?



XML Support sites

- ibm.com/developerWorks/xml - XML resources: tutorials, news, code
- ibm.com/alphaworks - site for free tools & emerging technologies from IBM.

email: MColan@us.ibm.com

visit: ibm.com/developerWorks/speakers/colan

Latest version of this talk, and all example files from talk