

Generating *beyootiful* PDF files with FOP

Doug Tidwell

IBM developerWorks

`dtidwell@us.ibm.com`

ibm.com/developerWorks/xml

FOP

fop, *n.*, a man overly concerned
about his appearance
(Middle English *fobben*, to deceive)

0

Agenda

- Process overview
- XSL-FO basics
- A simple sample
- Blocks and inline objects
- Lists
- Tables
- Graphics
- Links and cross-references
- Complex page layouts

0

Agenda

- Running headers & footers
- Bookmarks
- Tables of contents
- Indexes
- HTML to XSL-FO
- Using FOP
- Case study: The Toot-O-Matic
- XSL-FO resources
- Wrapup

Process overview

How we get from XML to PDF

0

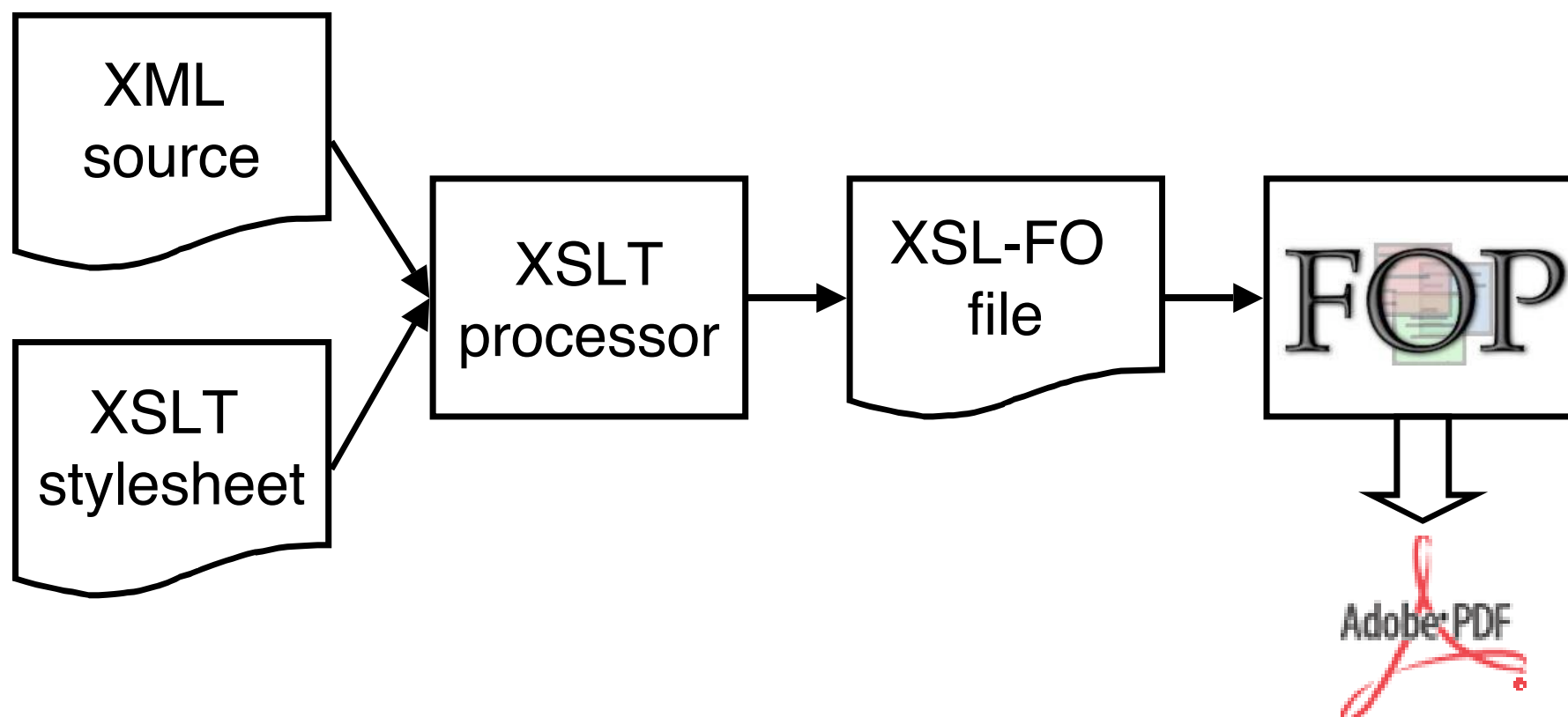
Process overview

- If we start with an XML document, how do we end up with a PDF file? We'll go through this process:
 - Use an XSLT stylesheet to convert the XML into formatting objects
 - Use FOP to convert the formatting objects into a PDF file.

ibm.com/developerWorks/xml

0

Process overview



ibm.com/developerWorks/xml

0

Process overview

- In this session, we'll focus on the formatting objects that are most useful for creating high-quality technical documents.
- We'll use FOP Version 0.20.4rc here, although there are other XSL-FO tools.
- We'll also take a look at how to create XSL-FO elements for HTML.

ibm.com/developerWorks/xml

XSL-FO basics

0

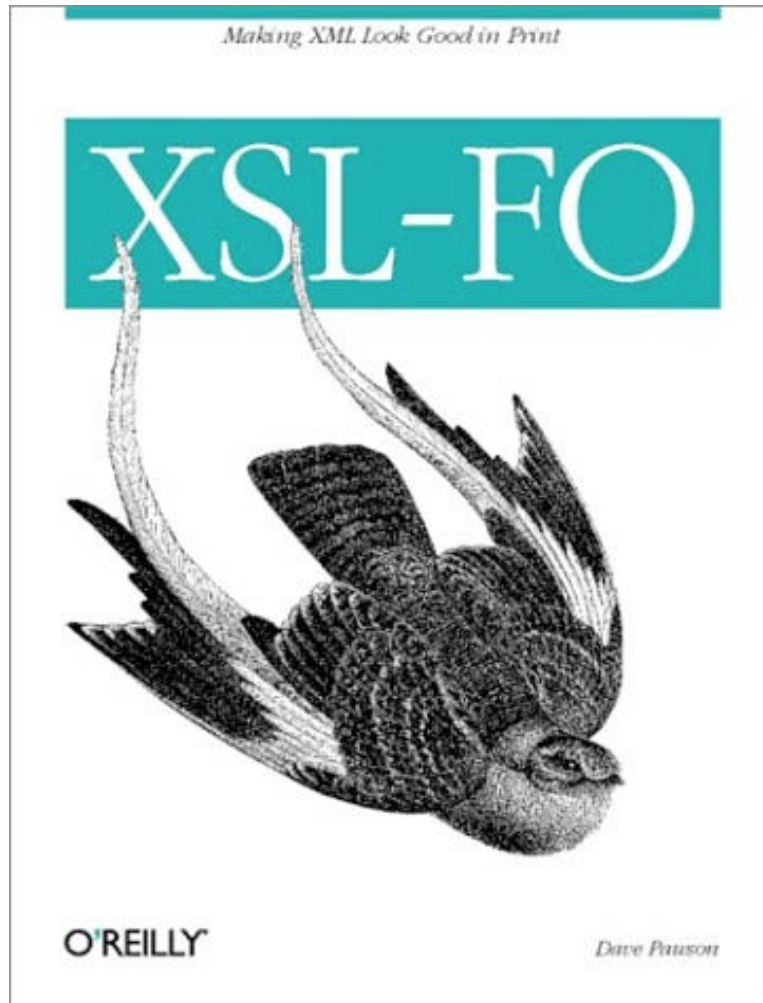
The spec

- The XSL-FO spec is at [`w3.org/TR/xsl/`](http://w3.org/TR/xsl/).
 - To assist readers in becoming confused, it's called XSL.
 - The spec is a very useful reference work; I keep a PDF of it on my machine.
- You can find XSL-FO references at [`w3.org/Style/XSL`](http://w3.org/Style/XSL).

ibm.com/developerWorks/xml

0

Buy this book!



ISBN 0596003552

It's available at
amazon . com, and is
also available under
the GFDL (the GNU
Free Documentation
License).

ibm.com/developerWorks/xml

0

XSL-FO basics

- The formatting objects spec defines dozens of elements and properties that define how content should be rendered.
 - Page size, margins, fonts, font sizes, colors, backgrounds, page numbers, cross-references, and so forth.

ibm.com/developerWorks/xml

0

XSL-FO and outputs

- Although we'll focus on converting files of XSL-FO elements into PDFs here, you can generate other result documents.
 - Voice files are one thing specifically targeted in the spec.

ibm.com/developerWorks/xml

0

A simple sample

0

A simple sample

- We'll look at a short XSL-FO file, `simple.fo`, then we'll discuss its parts.
- Don't be intimidated by `simple.fo`'s complexity; most of what's in `simple.fo` won't change from one XSL-FO file to the next.

ibm.com/developerWorks/xml

0

simple.fo

```
<fo:root
  xmlns:fo="http://www.w3.org...">
  <fo:layout-master-set>
    <fo:simple-page-master
      master-name="main"
      margin-top="25pt"
      margin-bottom="25pt"
      page-width="8.5in"
      page-height="11in"
      margin-left="75pt"
      margin-right="75pt">
```

ibm.com/developerWorks/xml

0

simple.fo

```
<fo:region-body  
  margin-bottom="50pt"  
  margin-top="50pt"/>  
</fo:simple-page-master>  
</fo:layout-master-set>  
<fo:page-sequence  
  master-reference="main">  
  <fo:flow flow-name=  
    "xsl-region-body">
```

ibm.com/developerWorks/xml

0

simple.fo

```
<fo:block font-size="14pt"
  line-height="17pt">
  This is a paragraph of text.
  Notice that as <fo:inline
  font-style="italic">this
  meaningless prose</fo:inline>
  drones on and on, the FOP
  software automatically calculates
  line breaks for us. Isn't that
  fascinating?
</fo:block>
```

ibm.com/developer/Works/xml

0

simple.fo

```
</fo:flow>
```

```
</fo:page-sequence>
```

```
</fo:root>
```

0

Page layout

- We'll discuss the page layout elements first for two reasons:
 1. You have to define the page layout before you can add any content to the XSL-FO file, and
 2. Once we've defined the page layout, it typically never changes.

ibm.com/developerWorks/xml

0

Page layout

`<fo:root>`

`<fo:layout-master-set>`

`<fo:simple-page-master>`

(maybe more than one)

`<fo:page-sequence>`

(refers to a `<fo:simple-page-master>`)

`<fo:flow>` (contains blocks,
images, tables, etc.)

ibm.com/developerWorks/xml

0

`<fo:root>`

- The `<fo:root>` element is, as you'd expect, the root of the XSL-FO document.
- It's attributes are typically namespace definitions only.
- It normally contains an `<fo:layout-master-set>`, followed by one or more `<fo:page-sequence>`s.

ibm.com/developerWorks/xml

0

<fo:layout-master-set>

- This usually contains some number of page definitions (defined in **<fo:simple-page-master>** elements).
 - For example, we might want different page layouts for even pages, odd pages, and the first page of each chapter.

ibm.com/developerWorks/xml

0

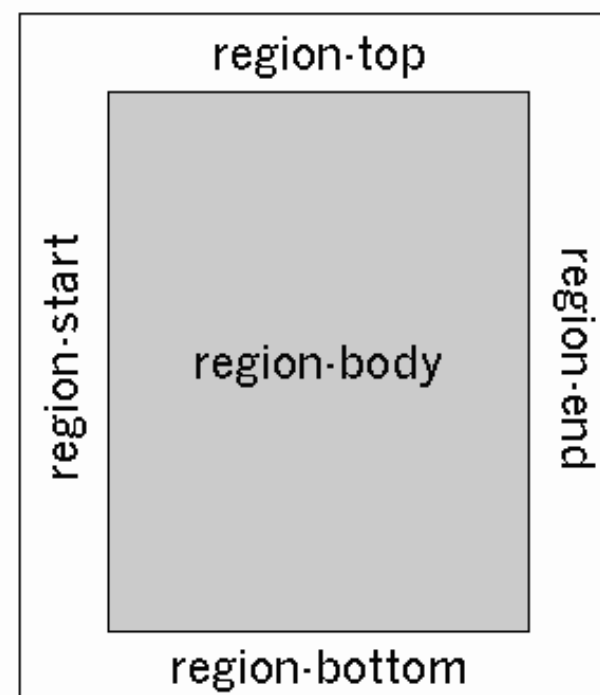
<fo:simple-page-master>

- Defines the characteristics of a single page, including its physical size.
 - The size of the page is the most common thing to change (letter, A4, ...).
 - Each <fo:simple-page-master> has a name: <fo:simple-page-master name="main" ... >. We'll use the name in a minute.
 - We can also define page margins, etc.

0

<fo:simple-page-master>

- A page has five regions. You can define the margins for each of them.



ibm.com/developerWorks/xml

0

Units of measure

- XSL-FO allows these units:
 - **cm** – centimeters
 - **mm** – millimeters
 - **in** – inches
 - **pt** – points ($1/72$ in)
 - **pc** – picas (12 pt)
 - **px** – pixels (see the spec for details)
 - **em** – The width of a capital 'M'

0

<fo:page-sequence>

- Defines the sequence in which the <fo:simple-page-master>s are used.
 - Our example used a single page layout, so it's pretty simple:
`<fo:page-sequence
 master-reference="main">`

0

<fo:flow>

- `<fo:flow flow-name="xsl-region-body">`
- This defines content that will be part of the body area of the page.
 - The names `xsl-region-body`, `xsl-region-top`, `xsl-region-bottom`, etc. are the default names. You can change them, but not reuse them.

ibm.com/developerWorks/xml

0

Go with the flow

- Within the `<fo:flow>` element, we'll put the text of our document.
- In our example, we had a block and an inline area that was italicized.
- `<fo:flow>` uses the page and column definitions from the current page master(s).

ibm.com/developerWorks/xml

Blocks and inline objects

0

Blocks and inline objects

- We'll group everything that creates simple content into two groups: blocks and inline objects.
 - A block always creates a line break.
 - An inline object never creates a line break.

ibm.com/developerWorks/xml

0

<fo:block>

- Defines a block of content. Think of this as a <p> element. Example:

```
<fo:block font-size="24pt"  
  line-height="27pt"  
  text-align-last="start"  
  space-before.optimum="24pt">  
  <!-- content goes here -->  
</fo:block>
```

ibm.com/developerWorks/xml

0

Text alignment

- There are two attributes to define the alignment of text in a block:
 - `text-align=`
`start|center|end|justify`
 - `text-align-last=`
`start|center|end|justify`

ibm.com/developerWorks/xml

0

Line height

- To get the results you want, the `line-height` of a block should be 3-6 points higher than the `font-size` of the block:

```
<fo:block ... font-size="24pt"  
  line-height="27pt">
```

0

Spacing between blocks

- You can use the **space-before** and **space-after** properties to control the space before and after a block:

```
<fo:block space-before="18pt">
```

0

Spacing between blocks

- These properties have a number of components; you can combine **space-before** and **space-after** with **.minimum**, **.maximum**, **.optimum**, and **.precedence**.

ibm.com/developerWorks/xml

0

`<fo:inline>`

- Allows you to change the properties of the text without starting a new line.
`<fo:inline text-weight="bold" color="red">`
- The XSL-FO spec defines literally dozens of properties, most of which are the same as CSS properties.

ibm.com/developerWorks/xml

0

Keeps and breaks

- There are various **keep** and **break** attributes to keep blocks of text together:

```
<fo:block ...  
  keep-with-next.within-page=  
  "always">
```
- I've had trouble getting these to work with FOP...

ibm.com/developerWorks/xml

0

Keeps and breaks

- The **break-before** and **break-after** attributes have five values:
 - **auto** – Let the formatter figure it out
 - **column** – Put a column break before this block
 - **page** – Put a page break before this block
 - **odd-page** and **even-page** – Put an odd or even page break before this block

ibm.com/developerWorks/xml

0

Keeps and breaks

- There are three compound properties: `keep-with-previous`, `keep-with-next`, and `keep-together`.
- They're combined with the components `.within-line`, `.within-column`, and `.within-page`.

ibm.com/developerWorks/xml

0

Keeps and breaks

- In other words, the attributes are:
 - `keep-with-next.within-line`
 - `keep-with-next.within-column`
 - `keep-with-next.within-page`
 - `keep-together.within-line`
 - `keep-together.within-column`
 - etc.

ibm.com/developerWorks/xml

0

Keeps and breaks

- The values for these properties are:
 - **auto** – Let the formatter figure it out
 - **always** – Always keep things together
 - You can also enter an integer value; the higher the number, the stronger the constraint. This lets you define priorities for different keep properties. **always** has the highest priority.

ibm.com/developerWorks/xml

0

Keeps and breaks

- A useful example: We always want a heading to appear with the first paragraph, so...

```
<fo:block font-size="18pt"  
  space-after.optimum="12pt"  
  keep-with-next.within-page=  
  "always">
```

Heading text

```
</fo:block>
```

ibm.com/developerWorks/xml

0



ibm.com/developerWorks/xml

Lists

0

Lists

- Paragraphs are pretty simple; now we'll move on to lists.
- Lists are more complicated because we need to define lots of things, such as:
 - The dingbat character (bullet, a number, etc.)
 - The distance between the dingbat and the text of each item
 - The vertical spacing between list items

ibm.com/developerWorks/xml

0

Lists

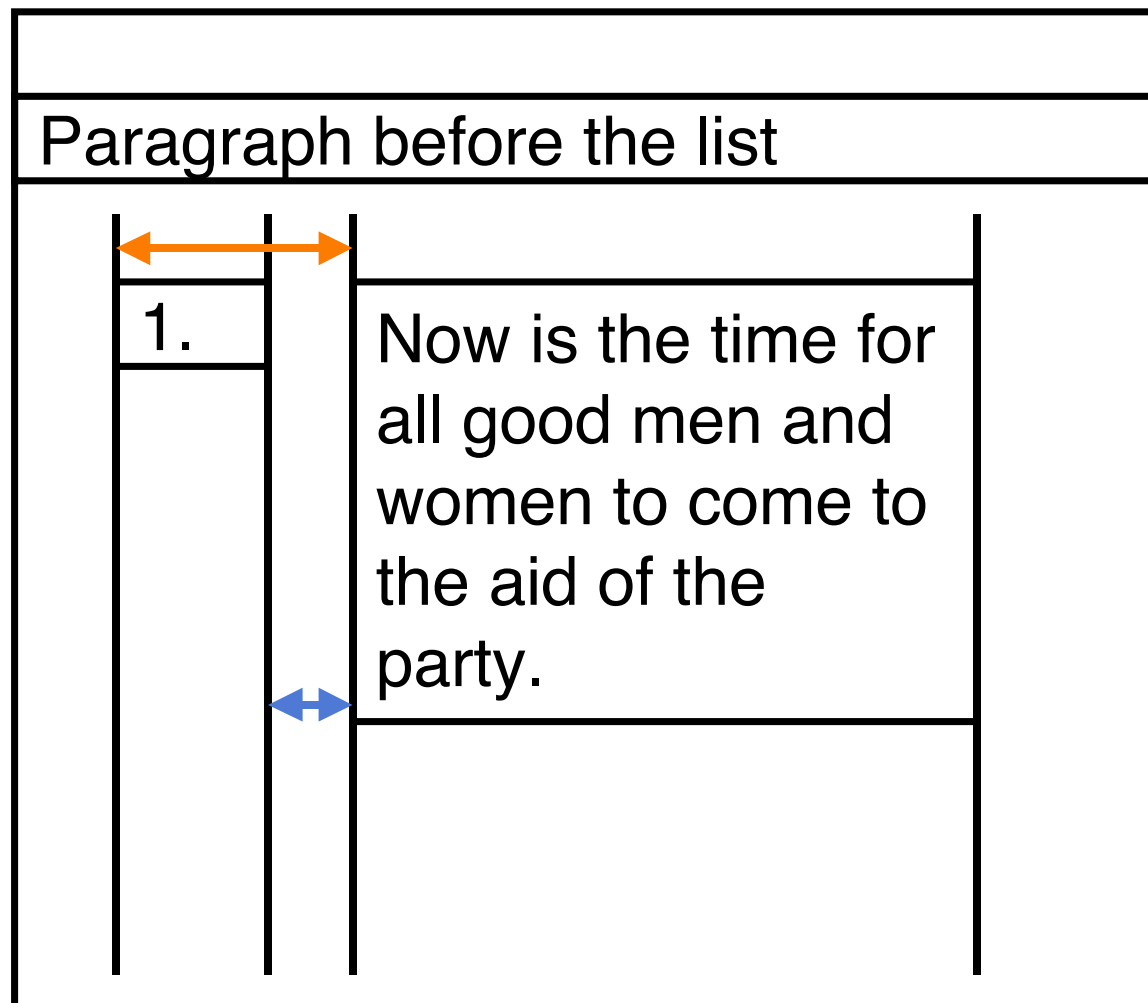
- We'll define an `<fo:list-block>`; that will contain some number of `<fo:list-item>`s.
- Each `<fo:list-item>` contains a `<fo:list-item-label>` and a `<fo:list-item-body>`.

ibm.com/developerWorks/xml

0

List properties

`fo:list-`
`block:`
`provisional-`
`distance-`
`between-`
`starts`
`fo:list-`
`block:`
`provisional-`
`label-`
`separation`



ibm.com/developerWorks/xml

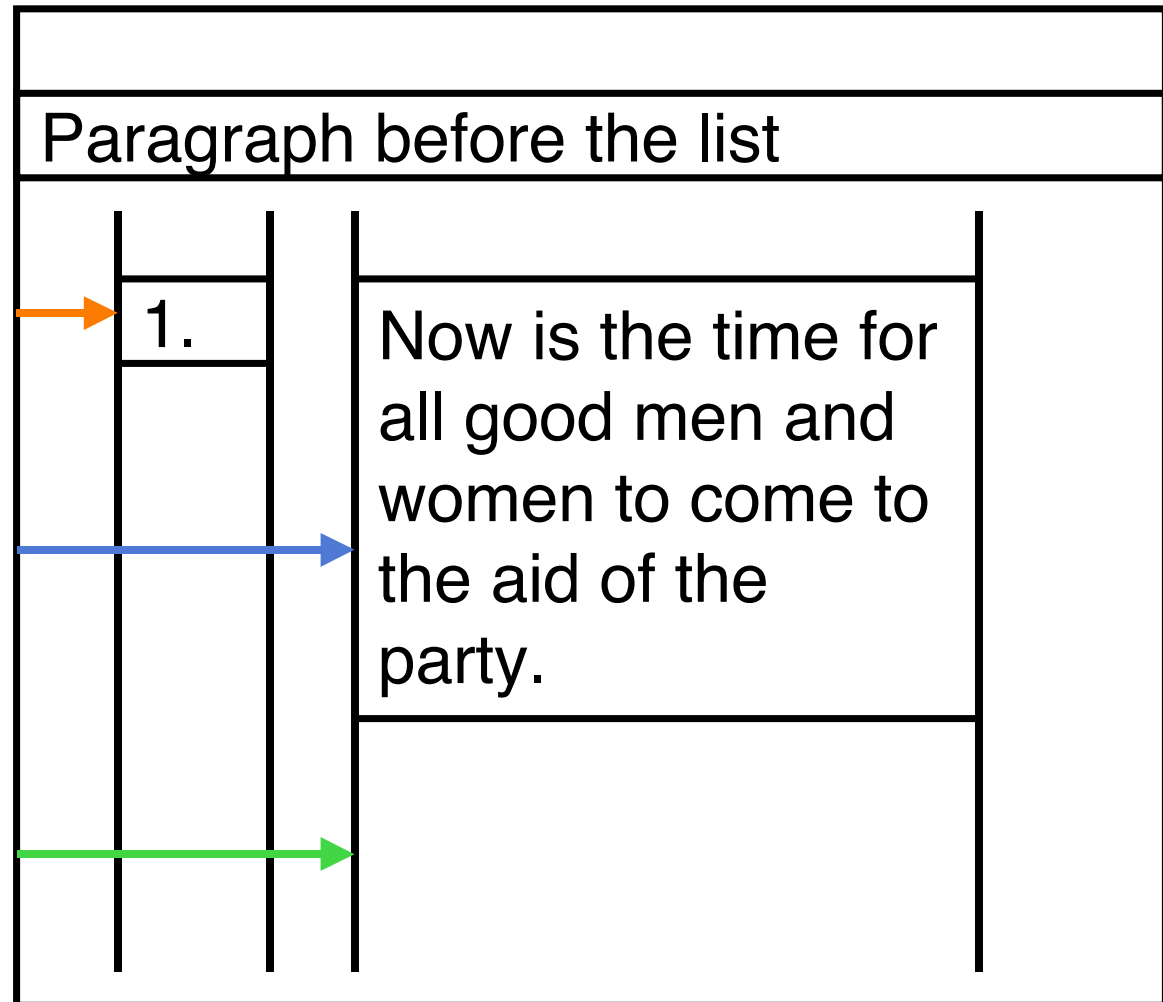
0

List properties

`list-item-label:`
`start-indent`

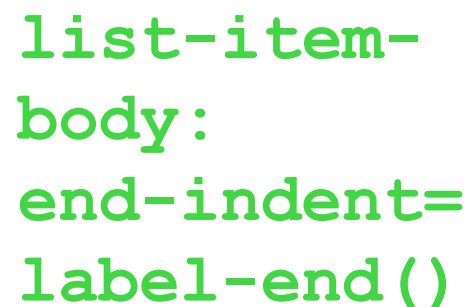
`list-item-body:`
`start-indent`

`list-item-body:`
`start-indent=`
`body-start()`



ibm.com/developerWorks/xml

ibm.com/developerWorks/xml



0

A numbered list

```
<fo:list-block provisional-  
  distance-between-starts="15mm"  
  provisional-label-  
  separation="5mm">  
  <fo:list-item>  
    <fo:list-item-label  
      start-indent="5mm"  
      end-indent="label-end()">  
      <fo:block>1.</fo:block>  
    </fo:list-item-label>
```

ibm.com/developerWorks/xml

0

A numbered list

```
<fo:list-item-body
start-indent="body-start()">
  <fo:block>
    blah blah blah
  </fo:block>
</fo:list-item-body>
</fo:list-item>
</fo:list-block>
```

ibm.com/developerWorks/xml

0

A numbered list

- To generate the item numbers in the XSL-FO file, we'll use the `<xsl:number>` element.
 - More on this later, when we cover ordered lists.

0

Bulleted lists

- For a bulleted list, we need to use a bullet character as the `<fo:list-item-label>`. Here's how that looks:

```
<fo:list-item-label  
  start-indent="5mm"  
  end-indent="label-end()">  
  <fo:block>&#x2022;</fo:block>  
</fo:list-item-label>
```

0

Definition lists

- For definition lists, there are two ways to go. If you want the formatting to look like this, use an `<fo:list-block>`:

term Here's the definition of this
term. Blah blah blah blah.

ibm.com/developerWorks/xml

0

Definition lists

- The trouble with this approach is that you need to set the `provisional-distance-between-starts` property to a value wide enough for your widest `<dt>` term.
 - We don't have any way of knowing that.

ibm.com/developerWorks/xml

0

Definition lists

- A simpler way to do things is to format the `<d1>` like this:

term

Here's the definition of this term.

Blah blah blah blah.

- You do this with simple `<fo:block>` elements, using the `start-indent` property for the `<dd>` portion.

0



Tables

ibm.com/developerWorks/xml

0

Tables

- Tables are the most complicated markup we'll use with content.
- If you've worked with HTML at all, you should have an idea of how often we'll end up using them.

ibm.com/developerWorks/xml

0

XSL-FO table elements

- `<fo:table>`
 - Defines a table. A table must contain at least one `<table-body>`, and may contain one or more `<table-column>`s, as well as an optional `<table-header>` and `<table-footer>`.

ibm.com/developerWorks/xml

0

XSL-FO table elements

- `<fo:table-column>`
 - Defines the properties of a table column. The most commonly used attribute is `column-width`.
 - For best rendering performance, you need to define all `<fo:table-columns>`, including the `column-width` of each.

ibm.com/developerWorks/xml

0

XSL-FO table elements

- **<fo:table-body>**
 - Defines the body of the table.
- **<fo:table-row>**
 - Defines a table row (<tr>)
- **<fo:table-cell>**
 - Defines a table cell (<td>)

0

XSL-FO table elements

ibm.com/developerWorks/xml

```
<fo:table>
  <fo:table-column
    column-width="100pt"/>
  <fo:table-column
    column-width="100pt"/>
  <fo:table-body>
    <fo:table-row>
      <fo:table-cell>...</fo:table-
cell>
```

0

XSL-FO table elements

```
<fo:table-cell>...</fo:table-  
cell>
```

```
</fo:table-row>
```

```
</fo:table-body>
```

```
</fo:table>
```

ibm.com/developerWorks/xml

0

XSL-FO table elements

- If you want a caption for your table, you create an `<fo:table-and-caption>` element; that contains the `<fo:table-caption>` and the `<fo:table>`.

ibm.com/developerWorks/xml

0



Graphics

ibm.com/developerWorks/xml

0

Graphics

- To add graphics to your PDF file, simply use the `<fo:external-graphic>` element.

```
<fo:external-graphic  
  src="images/x.gif"  
  width="150" height="200"/>
```

- You can put this in a `<fo:block>` if you want it separated from the surrounding text.

0

SVG

- FOP has limited support for Scalable Vector Graphics; it uses the Batik project from Apache.
- You can include SVG files with `<fo:external-graphic>` or put the SVG inline with `<fo:instream-foreign-object>`.

ibm.com/developerWorks/xml

0

SVG

- SVG inline looks like this:

```
<fo:instream-foreign-object>
  <svg:svg xmlns:svg="..."
    width="40" height="40">
    <svg:g style="fill:red;
      stroke:#000000">
      <svg:rect x="0" y="0"
        width="15" height="15"/>
```

ibm.com/developerWorks/xml

Links and cross-references

0

Cross-references

- The `<fo:basic-link>` element creates a link inside the formatted document:

```
<fo:basic-link color="blue"
  internal-destination="id37">
  xyz</fo:basic-link>
```

- This creates the link **xyz**; when you click on that link in the PDF file, you go directly to that section.

0

Cross-references

- FOP doesn't highlight links unless you tell it to. If you want the text of a link to be in blue, underlined, etc., you have to set those properties yourself.

ibm.com/developerWorks/xml

Page numbers

- The `<fo:page-number/>` element inserts the current page number. This is most commonly used in a header or footer.
 - BTW, you can change the starting page number like this:
`<fo:page-sequence
 master-reference="..."
 initial-page-number="57">`

Page-number references

- You can also include the page number of the document section you're referencing:

```
<fo:basic-link color="blue"
  internal-destination="id37">
  <fo:page-number-citation
    ref-id="id37" />
</fo:basic-link>
```

0

Cross-references

- Here's a good cross-reference:
 - For more information, see "[Building Tables](#)" on page [51](#).
- If you're reading this online, you get visual cues that the section title and page number are links.
- If you're reading a printed document, the reference is still useful.

ibm.com/developerWorks/xml

0

Page x of y references

ibm.com/developerWorks/xml

- To generate a reference like Page 8 of 10, there are two steps:
 1. Create a named block at the end of the document. `<fo:block id="TheVeryLastPage" />`
 2. To get the number of the last page, put a `<fo:page-number-citation>` element that refers to the `TheVeryLastPage` id.

0

Page *x* of *y* references

- Here's the complete markup:
`<fo:block>Page`
`<fo:page-number/>` of
`<fo:page-number-citation`
`ref-id="TheVeryLastPage"/>`
`</fo:block>`

ibm.com/developerWorks/xml

0

External links

- You can also create links to external references:

```
<fo:basic-link  
  external-destination=  
  "http://www.ibm.com/"  
  color="blue">IBM</fo:basic-link>
```

ibm.com/developerWorks/xml

0

External links

- Some viewers will display the target URL when you mouse over the link.
- A design question is how you should format a link when you're converting from (X)HTML. Do you print just the text here (IBM), or do you print the text and the URL?
 - Online viewers and hardcopy viewers have different needs...

ibm.com/developerWorks/xml

Complex page layouts

0

Complex page layouts

- To this point, we've used a single page layout (an `<fo:simple-page-master>`) to define the characteristics of a page.
- In professional documents, we frequently want different layouts for odd vs. even pages, title pages, first pages of chapters, etc.

ibm.com/developerWorks/xml

0

Complex page layouts

- We'll look at the steps to defining and using different page layouts on different pages.
 1. Define the different page layouts.
 2. Give the layouts unique names.
 3. Give the header and footer areas unique names.
 4. Define when to use each of the different page layouts.

0

Complex page layouts

- Creating different page layouts is done with multiple

`<fo:simple-page-master>`
elements:

`<fo:simple-page-master
 master-name="right">`

`<fo:simple-page-master
 master-name="left">`

`<fo:simple-page-master
 master-name="first">`

ibm.com/developerWorks/xml

0

Complex page layouts

- Within each
`<fo:simple-page-master>`, give
the header and footer regions unique
names.
`<fo:simple-page-master`
 `master-name="right">`
 `<fo:region-before`
 `region-name="before-right"`
 `extent="3cm" />`

ibm.com/developerWorks/xml

0

Complex page layouts

- More on defining page regions:

```
<fo:region-body  
  margin-top="1.5cm"  
  margin-bottom="1.5cm" />  
<fo:region-after  
  region-name="after-right"  
  extent="1cm" />  
</fo:simple-page-master>
```

ibm.com/developerWorks/xml

0

Complex page layouts

- To define when to use each of the page layouts, use the `<fo:page-sequence-master>` element.
- We'll give this a name, then refer to it later when we define the `<fo:flow>` for the main content of our document.

ibm.com/developerWorks/xml

0

Complex page layouts

- Here's the markup:

```
<fo:page-sequence-master  
  master-name="standard">  
  <fo:repeatable-page-master-  
alternatives>  
    <fo:conditional-page-  
master-reference  
      master-name="first"  
      page-position="first"/>
```

ibm.com/developerWorks/xml

0

Complex page layouts

- Markup, continued:
 <fo:conditional-page-master-
reference
 master-name="left"
 odd-or-even="even"/>
 <fo:conditional-page-master-
reference
 master-name="right"
 odd-or-even="odd"/>

ibm.com/developerWorks/xml

0

Complex page layouts

- Markup, continued:
 `</fo:repeatable-page-master-alternatives>`
 `</fo:page-sequence-master>`
- All of this markup goes up front in the `<fo:layout-master-set>` element.

ibm.com/developerWorks/xml

0

Complex page layouts

- To actually use the layouts we've defined, we reference the `<fo:page-sequence-master>` we just defined:
`<fo:page-sequence`
 `master-name="standard">`
 `<fo:flow`
 `flow-name="xsl-region-body">`
 `<!-- document content -->`

ibm.com/developerWorks/xml

Running headers & footers

0

Running headers & footers

- You use the `<fo:static-content>` element to define running headers and footers.
- Within each `<fo:static-content>` element, we refer to a `flow-name` that identifies where the static content should go.
- We'll refer back to the areas we named earlier.

ibm.com/developerWorks/xml

0

Running headers & footers

- Here's a header that contains the `<title>` of an HTML document:
`<fo:static-content
 flow-name="before-right">
 <fo:block
 text-align="center"
 font-size="10pt">
 <xsl:value-of
 select="/html/head/title">`

ibm.com/developerWorks/xml

0

Running headers & footers

- You can put tables inside the headers and footers to get the formatting you want.
- You can also use `<fo:page-number/>` to number the pages.

ibm.com/developerWorks/xml

0

Running headers & footers

- To **really** impress your friends, you can use the `<fo:marker>` element.
 - In a dictionary, the running header typically contains the first and last term on each page.
 - We'll create an `<fo:marker>` for every term, then use `<fo:retrieve-marker>` to get the marker values and put them in the header.

ibm.com/developerWorks/xml

Running headers & footers

- For each term in the glossary, we'll use this template:

```
<xsl:template match="term">
  <fo:block>
    <fo:marker
      marker-class-name="terms">
      <xsl:value-of select="."/>
    </fo:marker>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

0

Running headers & footers

- This markup puts the first term on the page in the header:

```
<fo:retrieve-marker  
  retrieve-class-name="terms"  
  retrieve-boundary="page"  
  retrieve-position=  
    "first-starting-within-page">
```

ibm.com/developerWorks/xml

0

Running headers & footers

- For the last term that ends on the page, we'll put this in the header:

```
<fo:retrieve-marker  
  retrieve-class-name="terms"  
  retrieve-boundary="page"  
  retrieve-position=  
    "last-ending-within-page">
```

ibm.com/developerWorks/xml

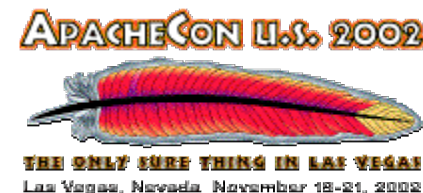
0

Running headers & footers

- There are other values for the `retrieve-boundary` and `retrieve-position` properties; check the spec for the details.

ibm.com/developerWorks/xml

0



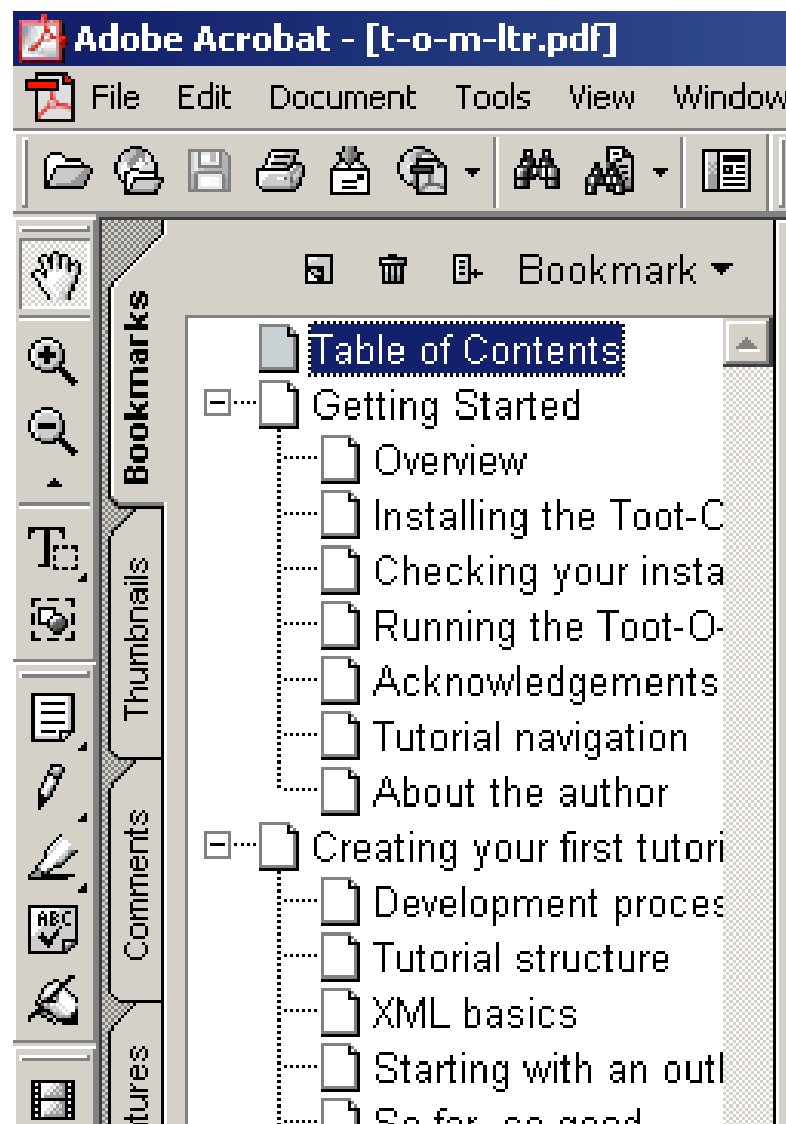
ibm.com/developerWorks/xml

Bookmarks

0

Bookmarks

- One feature of PDF files is bookmarking:



ibm.com/developerWorks/xml

Bookmarks

- Recent versions of FOP have extension elements that let you define bookmarks in the XSL-FO file:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="..."
  xmlns:fo="..."
  xmlns:fox="http://xml.apache
.org/fop/extensions"
```

0

Bookmarks

- With bookmarks, you can create an interactive table of contents for your PDF file.
- Readers can expand and collapse the list of bookmarks, and clicking any bookmark takes them to a particular page in the document.

ibm.com/developerWorks/xml

0

Bookmarks

- To create bookmarks, use the extension elements `<fox:outline>` and `<fox:label>`.
- You can nest one outline inside another to create a tree view of your document.

ibm.com/developerWorks/xml

0

Bookmarks

```
<fox:outline
  internal-destination="ch1">
  <fox:label>Chapter 1</fox:label>
  <fox:outline
    internal-destination="ch1.1">
    <fox:label>Intro</fox:label>
  </fox:outline>
</fox:outline>
```

ibm.com/developerWorks/xml

0

Bookmarks

- This short markup creates two bookmarks. The bookmark labeled "Chapter 1" is a main heading, and "Intro" appears under it.
- As you expand and collapse Chapter 1, the Intro heading appears and disappears.

ibm.com/developerWorks/xml

0

Bookmarks

- The `internal-destination` attribute must refer to the `id` of a particular element in the XSL-FO file.
- If it doesn't, you'll get an error when you try to build the PDF file.

ibm.com/developerWorks/xml

Bookmarks

- To use XSLT templates to create the bookmarks, we'll create a template for `<h1>` elements.
 - That template will create an `<fox:outline>` with the appropriate `internal-destination` and `<fox:label>`.
 - It will also call the bookmarks template for all `<h2>` elements, which calls the `<h3>` template, etc.

Bookmarks

- XSLT template:

```
<xsl:template match="h1"
  mode="bookmarks">
  <fox:outline
    internal-destination="@id">
    <fox:label>
      <xsl:value-of select="."/>
    </fox:label>
    <xsl:apply-templates
      select="h2"/>
```

0

Bookmarks

- XSLT template, continued:
 `</fox:outline>`
 `</xsl:template>`
- Notice that we used `mode="bookmarks"` to separate this template from other templates that process the content.

ibm.com/developerWorks/xml

0

Bookmarks

- The template for `<h2>` elements would work the same way, only it would call `<xsl:apply-templates>` for all `<h3>` elements it contains.
- You decide how many levels of headings you want to make bookmarks for.

ibm.com/developerWorks/xml

Tables of contents

0

Tables of contents

- For each entry in the table of contents, we'll include three things:
 1. The title of the section
 2. A row of dots (a **leader**)
 3. The page number where the section starts

ibm.com/developerWorks/xml

Tables of contents

- Here's an XSLT template for the <h1> elements:

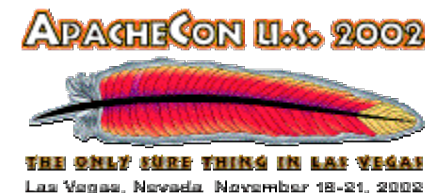
```
<xsl:template match="h1"
  mode="toc">
  <fo:block>
    <xsl:value-of select="." />
    <fo:leader
      leader-pattern="dots" />
    <fo:page-number-citation
      ref-id="{@id}" />
  </fo:block>
</xsl:template>
```

Tables of contents

- If you want to nest other headings inside the ToC, you could call the `<h2>` template:

```
<xsl:apply-templates  
select="h2" mode="toc"/>
```

0



ibm.com/developerWorks/xml

Indexes

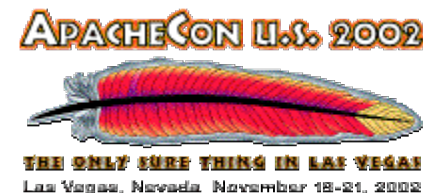
0

Indexes

- Indexes are more difficult to handle.
 - We'll create an empty `<fo:inline>` elements wherever there's an index entry.
 - We'll group all of the index entries (`<idx>` elements in DocBook). For each unique index entry, we'll print all of the page numbers.
- If you want an example, write me...

ibm.com/developerWorks/xml

0



ibm.com/developerWorks/xml

HTML to XSL-FO

0

HTML to XSL-FO

- For each of the major (X)HTML elements, we'll look at an XSLT template that converts the HTML element into XSL-FO.

ibm.com/developerWorks/xml

0

HTML to XSL-FO

- A quick note: To keep the slides relatively uncluttered, I'll use `<xsl:apply-templates/>`; the correct code is `<xsl:apply-templates select="*|text()" />`.

ibm.com/developerWorks/xml

0

``

- For a named anchor, you can create an empty `<fo:inline>` element. This doesn't cause a line break, and lets you link or refer to this spot later.

0

``

- XML source: ``

- XSLT template:

```
<xsl:template match="a[@name]">
```

```
  <fo:inline id="{@name}" />
```

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

- Results: `<fo:inline id="x" />`

0

``

- For a link to a named anchor point in this document, we'll use the `<fo:basic-link>` element and its `internal-destination` property.

0

``

- XML source:
`Chapter 1`

- XSLT template:

```
<xsl:template match="a[@href]">
  <fo:basic-link color="blue"
    internal-destination="...">
    <xsl:apply-templates/>
  </fo:basic-link>
</xsl:template>
```

0

``

- Results:
`<fo:basic-link color="blue" internal-destination="ch1">`
Chapter 1`</fo:basic-link>`
- To get the correct value for the link, we use the XSLT function `substring(@href, 2)`; we don't want the hash mark. We also need to add logic to the template...

0

``

- For a link to an external resource, we'll use the `<fo:basic-link>` element and its `external-destination` property.

0

``

- XML source:
`
IBM`
- XSLT template:
`<xsl:template match="a[@href]">
 <fo:basic-link color="blue"
 external-destination="...">
 <xsl:apply-templates/>
 </fo:basic-link>
</xsl:template>`

ibm.com/developerWorks/xml

0

``

- Results:
`<fo:basic-link color="blue"
external-destination=
"http://www.ibm.com">
IBM</fo:basic-link>`

ibm.com/developerWorks/xml

0

- To display text in boldface, we simply create an `<fo:inline>` element with the `font-weight="bold"` property.

0

- XML source: **xyz**
- XSLT template:

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

0

- Results:
`<fo:inline
font-weight="bold">
xyz</fo:inline>`

0

<big>

- To display text in a bigger font, we create an `<fo:inline>` element with a `font-size="120%"` property.

0

<big>

- XML source: <big>No!</big>
- XSLT template:

```
<xsl:template match="big">  
  <fo:inline font-size="120%">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```
- Results: <fo:inline font-size="120%">No!</fo:inline>

0

`<blockquote>`

- We handle a `<blockquote>` element with a new `<fo:block>` that has `start-indent` and `end-indent` properties.

0

<blockquote>

- XML source: <blockquote>When in the course of human events...

- XSLT template:

```
<xsl:template match="blockquote">
  <fo:block start-indent="1.5cm"
    end-indent="1.5cm">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

ibm.com/developerWorks/xml

0

<bblockquote>

- Results: `<fo:block
start-indent="1.5cm"
end-indent="1.5cm">When in
the course of human
events...</fo:block>`

ibm.com/developerWorks/xml

0

<body>

- Typically the HTML <body> element is converted into <fo:flow flow-name="xsl-region-body">.

0

<body>

- XSLT template:

```
<xsl:template match="body">  
  <fo:flow flow-name=  
    "xsl-region-body">  
    <xsl:apply-templates/>  
  </fo:flow>  
</xsl:template>
```
- This handles the <body> and everything inside it.

0

**
**

- The **
** element is also simple; we create an empty **<fo:block>**.

- XSLT template:

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

```
    <fo:block> </fo:block>
```

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

- We don't use **<xsl:apply-templates>** here because **
** is an empty element.

0

<caption>

- The HTML `<caption>` element maps nicely to the `<fo:table-caption>` element.
- If you're using a caption with a table, you need an `<fo:table-and-caption>` element which contains the `<fo:table-caption>` and the `<fo:table>`.

0

<caption>

- XSLT template:

```
<xsl:template  
    match="caption">  
    <fo:table-caption>  
        <xsl:apply-templates/>  
    </fo:table-caption>  
</xsl:template>
```

0

<center>

- The <center> element is handled by the `text-align` property of <fo:block> and other elements.

- XSLT template:

```
<xsl:template match="center">  
  <fo:block text-align="center">  
    <xsl:apply-templates/>  
  </fo:block>  
</xsl:template>
```

0

<cite>

- The simple approach to handling the `<cite>` element is to put its text inside an `<fo:block>` element with `font-style="italic"`.
- However, a `<cite>` element inside an `<i>` element should **not** be in italics to emphasize it.

0

<cite>

- XSLT template (part 1):

```
<xsl:template match="cite">
  <xsl:choose>
    <xsl:when test="parent::i">
      <fo:inline
        font-style="normal">
        <xsl:apply-templates/>
      </fo:inline>
    </xsl:when>
```

0

<cite>

- XSLT template (part 2):

```
<xsl:otherwise>  
  <fo:inline  
    font-style="italic">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:otherwise>  
</xsl:choose>  
</xsl:template>
```


0

<code>

- We'll handle the HTML `<code>` element with an `<fo:inline>` element with the `font-family="monospace"` property.

0

<code>

- XSLT template:

```
<xsl:template match="code">  
  <fo:inline  
    font-family="monospace">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```

0

<dl>, <dt>, and <dd>

- The easiest way to format a definition list is to put the term (<dt>) on one line, followed by any definitions (<dd>) starting on the next line.

fop [<dt>]

A man overly concerned about his
appearance [<dd>]

0

<dl>, <dt>, and <dd>

- XSLT template for <dl>:

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

0

<dl>, <dt>, and <dd>

- XSLT template for <dt>:

```
<xsl:template match="dt">  
  <fo:block  
    font-weight="bold">  
    <xsl:apply-templates/>  
  </fo:block>  
</xsl:template>
```

0

<dl>, <dt>, and <dd>

- XSLT template for <dd>:

```
<xsl:template match="dd">  
  <fo:block  
    start-indent="1.5cm">  
    <xsl:apply-templates/>  
  </fo:block>  
</xsl:template>
```

0

<dl>, <dt>, and <dd>

- To handle multiple <dd> elements on the same <dt>, you could set the **space-before** property:

```
<xsl:attribute  
    name="space-before">  
  <xsl:choose>  
    <xsl:when  
      test="previous-  
sibling::dd">  
      <xsl:text>14pt</xsl:text>  
    </xsl:when>
```

0

<dl>, <dt>, and <dd>

- As we mentioned earlier, you can have the terms appear in a separate column next to the definitions, but that's problematic.
 - You have to leave enough space for the longest term, and you don't have any way to determine that.

ibm.com/developerWorks/xml

0

<dfn>

- The <dfn> element is usually displayed in italics:

- XSLT template:

```
<xsl:template match="dfn">  
  <fo:inline  
    font-style="italic">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```

0

- Like <dfn>, the element is usually displayed in italics.
- XSLT template:

```
<xsl:template match="em">
  <fo:inline
    font-style="italic">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

0

``

- We'll handle the `color` attribute of the `` tag with an `<fo:inline>` element.
- XSL-FO defines 16 color names: `aqua`, `black`, `blue`, `fuchsia`, `gray`, `green`, `lime`, `maroon`, `navy`, `olive`, `purple`, `red`, `silver`, `teal`, `white`, and `yellow`.

0

- You can also specify colors with the `rgb()` function:

```
<fo:inline  
color="rgb(255,255,255)">
```

- The XSL-FO spec defines two other functions, `rgb-icc()` and `system-color()`, but those aren't currently supported by FOP.

0

``

- We'll write our XSLT template with this logic:
 - If the value of `color` begins with a hash mark, we'll assume the 2nd and 3rd characters are the red value, the 4th and 5th characters are the green value...
 - If the value of `color` doesn't begin with a hash mark, we'll assume it's a color name and pass it on to FOP.

0

- XSLT template:

```
<xsl:template match="font">
  <fo:inline>
    <xsl:attribute name="color">
      <xsl:choose>
        <xsl:when
          test="starts-with(@color,
            ' # ')">
          <xsl:text>rgb(</xsl:text>
            <xsl:value-of
              select="substring(@color,2,2)"/>
```

0

``

- To handle the face attribute of the `` element, simply copy its value to the `font-family` property.

- XSLT template:

```
<xsl:template match="font">
  <fo:inline font-
family="{@face}">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

0

``

- This one is also simple; we'll copy the **size** attribute to the **font-size** property.

- XSLT template:

```
<xsl:template match="font">
  <fo:inline font-size="{@size}">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```


0

- To handle this element correctly, we'll need to put some logic into our XSLT stylesheet.
- We'll combine the techniques we used on the last several slides.

ibm.com/developerWorks/xml

0

- XSLT template:

```
<xsl:template match="font">  
  <fo:inline>  
    <xsl:if test="@color">  
      <xsl:attribute name="color">  
        <!-- handle color attr -->  
      </xsl:attribute>  
    </xsl:if>  
  </fo:inline>  
</xsl:template>
```


etc.

0

`<h1>` through `<h6>`

- These are easy to handle, we simply need to decide how we want each of them to look.
- Let's assume we want an `<h1>` element to be 36-point type in a serif font, right-aligned. We'll make an `<h2>` be 32-point type, serif font, and left-aligned.

ibm.com/developerWorks/xml

0

<h1> through <h6>

- XSLT template for <h1>:

```
<xsl:template match="h1">  
  <fo:block font-size="36"  
    font-family="serif"  
    text-align="end">  
    <xsl:apply-templates/>  
  </fo:block>  
</xsl:template>
```

ibm.com/developerWorks/xml

0

<h1> through <h6>

- XSLT template for <h2>:

```
<xsl:template match="h2">  
  <fo:block font-size="32"  
    font-family="serif"  
    text-align="start">  
    <xsl:apply-templates/>  
  </fo:block>  
</xsl:template>
```

ibm.com/developerWorks/xml

0

<hr>

- To generate a horizontal rule, we'll use the `<fo:leader>` element.

- XSLT template:

```
<xsl:template match="hr">
  <fo:block>
    <fo:leader leader-
      pattern="rule"
      space-after="6pt">
    </fo:block>
  </xsl:template>
```

0

<i>

- The italic element is simple; we'll just create an `<fo:inline>` element with `font-style="italic"`.

- XSLT template:

```
<xsl:template match="i">  
  <fo:inline font-style="italic">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```

ibm.com/developerWorks/xml

0

- The element maps directly to the <fo:external-graphic> element.

- XSLT template:

```
<xsl:template match="img">  
  <fo:external-graphic  
    src="{@src}" />  
</xsl:template>
```

ibm.com/developerWorks/xml

0

- To help FOP as much as possible, we should include any **height** and **width** attributes from the HTML.

- A **better** XSLT template:

```
<xsl:template match="img">
  <xsl:if
    test="@width and @height">
    <fo:external-graphic
      src="{@src}" width="{@width}"
      height="{@height}"/>
```

ibm.com/developerWorks/xml

0

<kbd>

- The <kbd> tag can be rendered in a slightly larger monospaced font.

- XSLT template:

```
<xsl:template match="kbd">  
  <fo:inline  
    font-family="monospace"  
    font-size="110%">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```

0

``

- We'll look at list items when we handle the `` and `` elements.

0

<nobr>

- Although <nobr> is not part of the strict XHTML 1.0 standard, it is easily implemented.

- XSLT template:

```
<xsl:template match="nobr">
  <fo:inline wrap-option="no-
wrap">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

0

<nobr>

- We implemented <nobr> with an inline element here; you could just as easily do this with an <fo:block>.

ibm.com/developerWorks/xml

0

- For the element, we'll use the XSL-FO list elements.
- XSLT template:

```
<xsl:template match="ol">
  <fo:list-block provisional-
distance-between-starts="0.4cm"
provisional-label-separation=
"0.2cm">
    <xsl:apply-templates/>
  </fo:list-block>
</xsl:template>
```

0

- For each list item in the list, we'll use this template:

```
<xsl:template match="ol/li">
  <fo:list-item space-after="3pt">
    <fo:list-item-label
      end-indent="label-end()">
      <fo:block text-align="end">
        <xsl:number
value="position()"
      format="1." />
      </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="label-end()">
      <fo:block>
```

0

- We used the **<xsl:number>** element to insert the position of the current list item. You could change the number format if you wanted:

```
<xsl:when test="@type='a' ">
  <fo:list-item-label
    end-indent="label-end()">
    <fo:block>
      <xsl:number
        select="position()"
        format="a." />
```


0

- Another advanced technique is to change the spacing of lists that are embedded inside another list:

```
<xsl:when test="parent::ol">
```

```
  <fo:list-block  
    space-after="0pt">
```

...

```
<xsl:otherwise>
```

```
  <fo:list-block  
    space-after="14pt">
```

0

<p>

- The paragraph element is simple; we'll just make it a block:

```
<xsl:template match="p">  
  <fo:block  
    space-after="14pt">  
    <xsl:apply-templates/>  
  </fo:block>  
</xsl:template>
```

ibm.com/developerWorks/xml

0

<pre>

- The <pre> element has a couple of complications: we need to preserve all of the white space it contains, and we need to turn off word wrapping.

```
<xsl:template match="pre">  
  <fo:block  
    font-family="monospace"  
    white-space-collapse="false"  
    wrap-option="no-wrap">
```

ibm.com/developerWorks/xml

0

<samp>

- Like <kbd>, <samp> can be rendered in a slightly larger monospaced font.

- XSLT template:

```
<xsl:template match="samp">
  <fo:inline
    font-family="monospace"
    font-size="110%">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

0

<small>

- We'll handle small with a relative font-size property:

```
<xsl:template match="small">  
  <fo:inline font-size="80%">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```

ibm.com/developerWorks/xml

0

- The element is typically rendered in bold; that's not difficult.

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

ibm.com/developerWorks/xml

0

<sub>

- For subscripts and superscripts, we'll set the vertical-align property of the <fo:inline> element.

```
<xsl:template match="sub">  
  <fo:inline font-size="6pt"  
    vertical-align="sub">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```

0

<sup>

- Here's the same treatment for the <sup> element:

```
<xsl:template match="sup">
  <fo:inline font-size="6pt"
    vertical-align="super">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

ibm.com/developerWorks/xml

0

<table>

- The major difficulty in building tables with FOP is that you have to specify the width of all the columns in the table.
 - This is a FOP limitation, not an XSL-FO constraint.
- You specify the width of the columns with multiple **<fo:table-column>** elements.

ibm.com/developer/Works/xml

0

<table>

- Another difficulty in building tables is the crappy, inconsistent markup in most HTML tables.
 - <th> vs. <thead>
 - <tfoot>
 - <caption>
 - Column dimensions specified lots of different ways...

ibm.com/developerWorks/xml

0

<table>

`<fo:table>`

`<fo:table-column>`

`<fo:table-column>`

`<fo:table-header>`

`<fo:table-footer>`

`<fo:table-body>`

`<fo:table-row>`

`<table-cell>`

`<table-cell>`

ibm.com/developerWorks/xml

0

<table>

- The <table-header>, <table-footer>, and <table-body> elements have the same structure:

```
<fo:table-header> -or-  
<fo:table-footer> -or-  
<fo:table-body>
```

```
<fo:table-row> 

|              |
|--------------|
| <table-cell> |
| <table-cell> |


```

0

<table>

- If you want a table with a caption, wrap everything in an `<fo:table-and-caption>` element:

```
<fo:table-and-caption>
```

```
<fo:table-caption>
```

```
<fo:table>
```

0

<table>

- Here's the mapping between XHTML and XSL-FO:

<code><table></code>	<code><fo:table></code>
<code><tr></code>	<code><fo:table-row></code>
<code><td></code>	<code><fo:table-cell></code>
<code><th></code>	<code><fo:table-cell></code>
<code><thead></code>	<code><fo:table-header></code>
<code><caption></code>	<code><fo:table-caption></code>

0

<table>

- XSLT template:

```
<xsl:template match="table">  
  <fo:table>  
    <!-- figure out columns -->  
    <xsl:if test="thead">  
      <fo:table-header>  
        <xsl:for-each  
          select="thead/td">  
          ...  
        </xsl:for-each>
```

0

<table>

```
    </fo:table-header>
</xsl:if>
<fo:table-body>
  <xsl:for-each
    select="tbody/tr">
    <xsl:apply-templates/>
  </xsl:for-each>
</fo:table-body>
</fo:table>
</xsl:template>
```

ibm.com/developerWorks/xml

0

<table>

- XSLT template for <tr>:

```
<xsl:template match="tr">  
  <fo:table-row>  
    <xsl:apply-templates/>  
  </fo:table-row>  
</xsl:template>
```

0

<table>

- XSLT template for <td>:

```
<xsl:template match="td">  
  <fo:table-cell>  
    <xsl:apply-templates/>  
  </fo:table-cell>  
</xsl:template>
```
- You'd handle <th> the same way, only you'd want to use a **font-weight of bold**.

0

<title>

- We can handle this like a heading, but it's more common to use the text of the <title> element in a running header or footer.

0

<title>

- XSLT template:

```
<xsl:template match="/">
  <fo:static-content
    flow-name="region-after">
    <fo:block>
      <xsl:value-of
        select="/html/head/title"/>
    </fo:block>
  </fo:static-content>
</xsl:template>
```

0

<tt>

- Teletype text can be rendered with the `font-family="monospace"` property:

```
<xsl:template match="tt">
  <fo:inline
    font-family="monospace">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

ibm.com/developerWorks/xml

0

<u>

- We'll use the `text-decoration="underline"` property to create underlined text:

```
<xsl:template match="u">  
  <fo:inline  
    text-decoration="underline">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```

0

- Unordered lists are simpler than ordered lists because we don't have to number each item.
- We'll use a Unicode bullet character for the bullet, although you're certainly welcome to use whatever character you want.

0

- For the element, we'll use the XSL-FO list elements.
- XSLT template:

```
<xsl:template match="ul">
  <fo:list-block provisional-
distance-between-starts="0.4cm"
provisional-label-separation=
"0.2cm">
    <xsl:apply-templates/>
  </fo:list-block>
</xsl:template>
```


0

- For each list item in the list, we'll use this template:

```
<xsl:template match="ul/li">
  <fo:list-item space-after="3pt">
    <fo:list-item-label
      end-indent="label-end()">
      <fo:block>
        &#x2022;
      </fo:block>
    </fo:list-item-label>
```

0

<var>

- The <var> element is typically italicized, monospaced text.

- XSLT template:

```
<xsl:template match="var">  
  <fo:inline font-style="italic"  
    font-family="monospace">  
    <xsl:apply-templates/>  
  </fo:inline>  
</xsl:template>
```

0



ibm.com/developerWorks/xml

Using FOP

0

Using FOP

- To use FOP from the command line:
`java org.apache.fop.apps.Fop
test.fo test.pdf`
where `test.fo` is the name of the
XSL-FO file, and `test.pdf` is the
name of the PDF file.
- Type the command without options to
get a help screen.

ibm.com/developerWorks/xml

0

Using FOP with Cocoon

- Another way to use FOP is to serve XML documents from inside Cocoon.
- You can set up your XML documents so that the Cocoon servlet automatically invokes FOP when a client requests them. The generated PDF file is sent to the client.

ibm.com/developerWorks/xml

0

Embedding FOP

- The FOP package includes a sample Java application that embeds the FOP engine.

ibm.com/developerWorks/xml

0

Embedding FOP

```
Driver driver = new Driver();  
driver.setLogger(logfile);  
driver.setRenderer  
    (Driver.RENDER_PDF);  
driver.setOutputStream  
    (pdfFile);  
driver.render(parser,  
    inputHandler.  
        getInputSource());
```

ibm.com/developerWorks/xml

0

The FOP source code

- If you look at the FOP source code, they generate many of their Java classes with XML and XSLT.
- There are dozens, maybe even hundreds, of properties; the Java classes that manage them are generated automatically.

ibm.com/developerWorks/xml

0

FOP source code

- Here's the XML "source" for the **text-align** property:

```
<property>
  <name>text-align</name>
  <inherited>true</inherited>
  <datatype>Enum</datatype>
  <enumeration>
    <value const="CENTER">center</value>
    <value const="END">end</value>
    <value const="START">start</value>
    <value const="JUSTIFY">justify</value>
  </enumeration>
  <default>start</default>
</property>
```

ibm.com/developerWorks/xml

0

FOP source code

- FOP's build process converts the XML into Java code and compiles it.
- An alternate approach would be to define an XML Schema for XSL-FO (has someone done that already?), then generate the Java code from it.
- Consider using this technique in your own projects...

ibm.com/developerWorks/xml

Case study: The Toot-O-Matic

0

Building tutorials with XML

- At developerWorks, our most popular kind of content is our tutorials.
- We've built an XSLT-based system that converts a single XML document into multiple HTML files, PDFs, JPEGs, and a ZIP file.
- Even though the XML never leaves our server, we save lots of time & money.

ibm.com/developerWorks/xml

0

Demo time

- We'll take an XML file and convert it into a bunch of things, including two PDFs (one letter-sized, one A4-sized).

ibm.com/developerWorks/xml

0

It's open source!

- The Toot-O-Matic source code is available at:
 - `www6.software.ibm.com/dl/devworks/dw-tootomatic-p`
- Even if you're not writing tutorials (you're probably not), you can still get some useful techniques from the XSLT and Java source.

ibm.com/developerWorks/xml

XSL-FO resources

0

Other tools

- There are other (non-open-source) rendering engines out there:
 - RenderX
 - `renderx.com` – Really cool, check out their web site
 - Antenna House
 - `www.antennahouse.com`
- See `w3.org/Style/XSL/` for more resources.

ibm.com/developerWorks/xml

0

Other targets

- Braifo – FOs to Braille
 - See braifo.sourceforge.net/
- Voice - The FO spec features aural properties, such as **pitch**, **speech-rate**, **stress**, **volume**, **voice-family**, etc.
 - These are based on seldom-implemented CSS2 properties of the same name.
 - Check out www.arsdigita.com/asj/vxml/

ibm.com/developerWorks/xml

0

The Apache XML Project

- All the tools I've demoed today (XML parser, XSLT engine, FOP, etc.) came from the Apache XML project, **`xml.apache.org`**.
- If you don't visit this site, **you should**.

ibm.com/developerWorks/xml

0

A wee plug

- The developerWorks XML zone (ibm.com/developer/xml) is **chock full** of articles, news headlines, tutorials, and sample code, all of which is free.
- dW also has zones for Web services, Java, Linux, Open Source, security, Web architecture, and Unicode.

ibm.com/developerWorks/xml

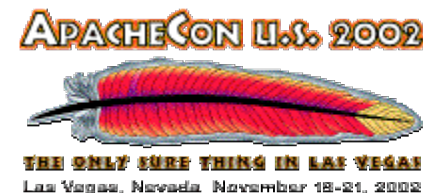
0

Converting XML to PDF

- A dW article demonstrating how to transform XML into PDF using formatting objects is at:
 - `ibm.com/developerworks/education/transforming-xml/`
- This takes six different XML documents and converts them into PDFs.

ibm.com/developerWorks/xml

0



ibm.com/developerWorks/xml

Wrapup

The home stretch

0

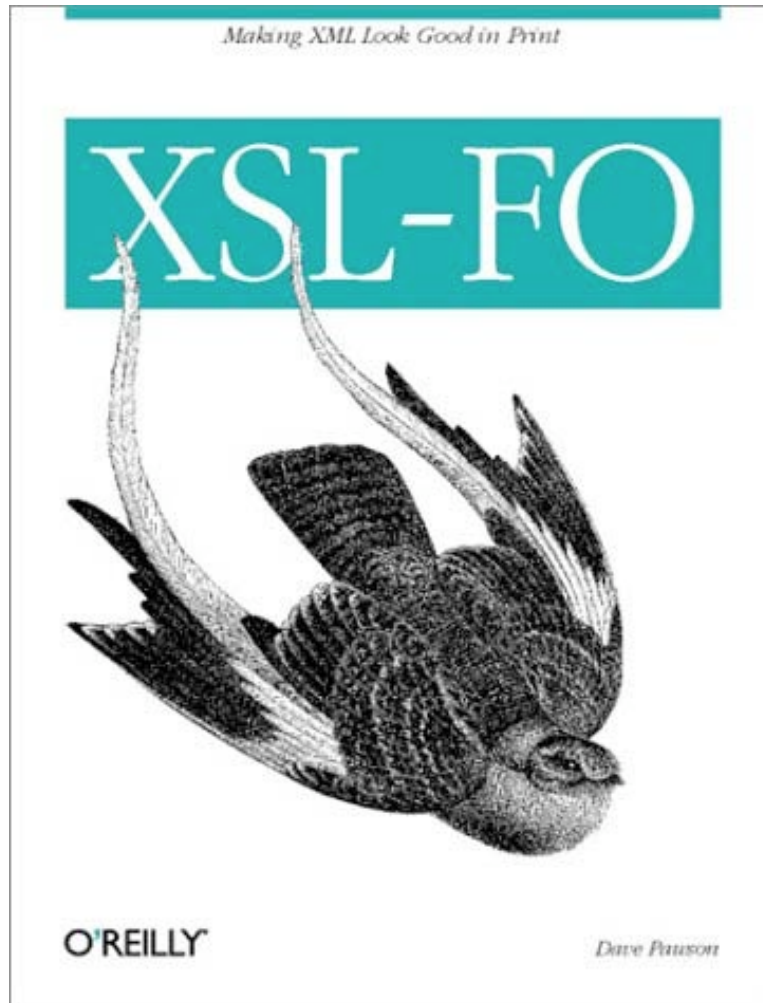
Wrapup

- All of the tools and interfaces we've covered today are:
 - Based on open standards
 - Available on any Java-enabled platform
 - Open source
 - Available at no cost to you, the home viewer

ibm.com/developerWorks/xml

0

Buy this book!



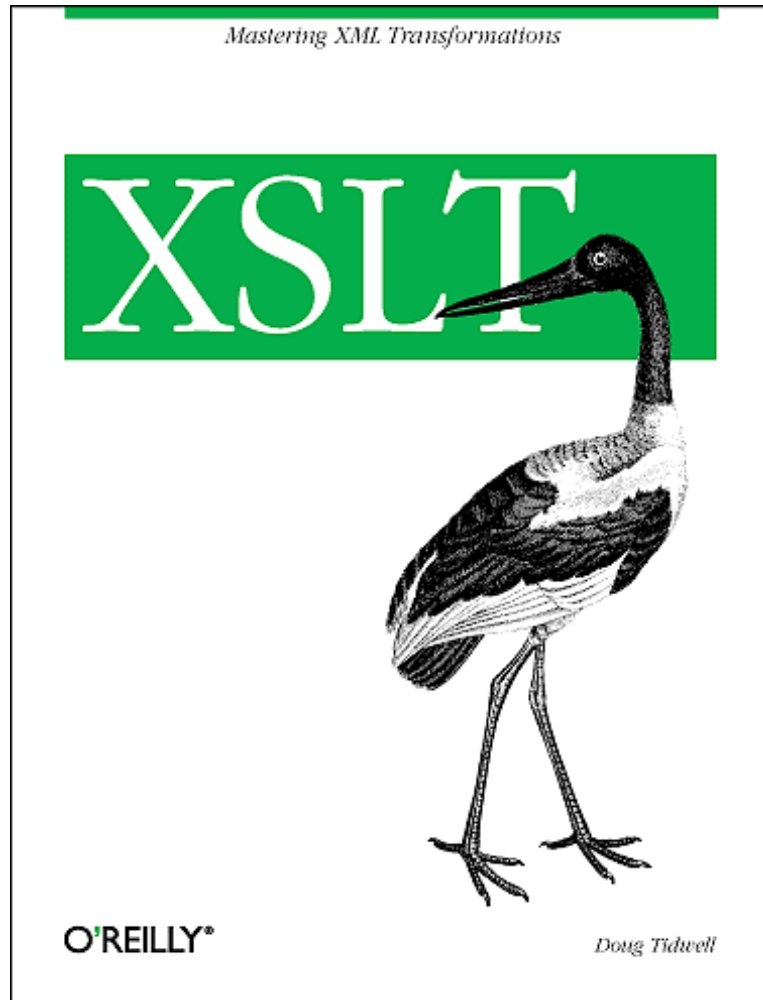
ISBN 0596003552

It's available at
amazon . com, and is
also available under
the GFDL (the GNU
Free Documentation
License).

ibm.com/developerWorks/xml

0

Shameless self-promotion



ISBN 0596000537

Order your copy at
amazon.com today!

Makes a great holiday gift!

Show your loved ones
how much you care
by showing them the power
of XSLT!

ibm.com/developerWorks/xml

0

Schamlos – Teil zwei

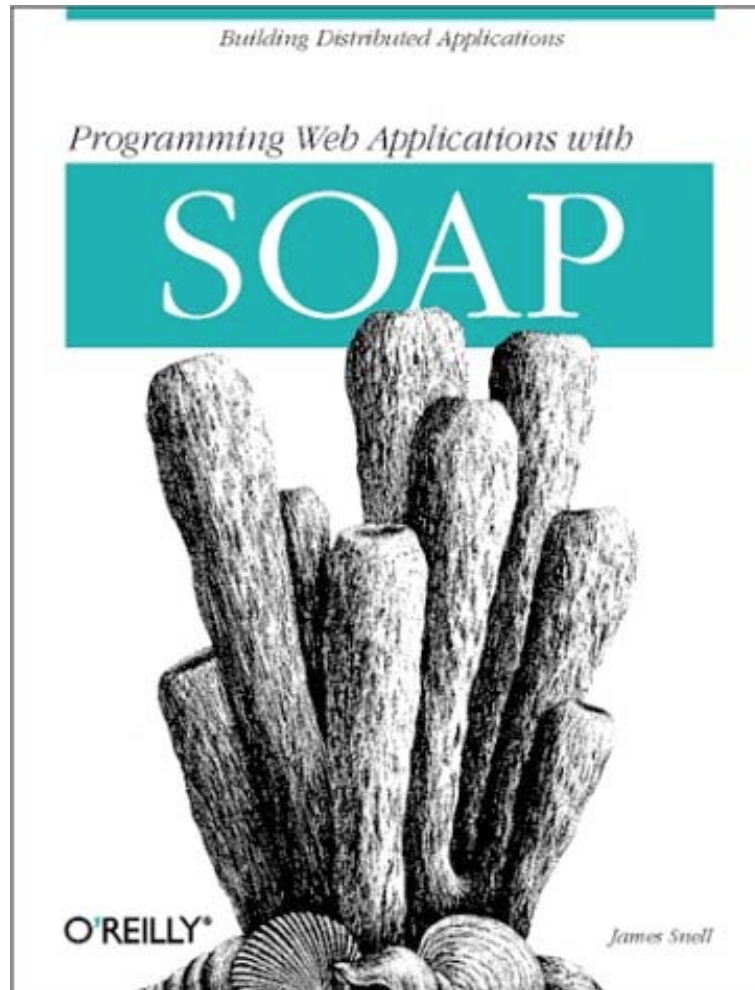


Wer XML bereits kennt und ein Problem schnell lösen möchte oder noch nach einem Problem sucht, für den ist XSLT genau richtig. Doug Tidwell richtet sich an XML-erfahrene Entwickler, die schnell in die komplexen Sphären von XSL eintauchen möchten und Antworten auf ihre eigenen Problemstellungen suchen. – Norbert Hartl
See www.oreilly.de/catalog/xsltger.

ibm.com/developerWorks/xml

0

Shameless – Part 3



ISBN 0596000952

Co-Written with James
Snell (soap-wrc.org)
and Paul Kulchenko.

Available at amazon.com
as well.

ibm.com/developerWorks/xml

0

These slides and samples...

- ...are available at
`www.ibm.com/developerworks/
speakers/dtidwell.`

[ibm.com/developerWorks/xml](http://www.ibm.com/developerWorks/xml)

Thanks for coming!

Doug Tidwell

IBM developerWorks

`dtidwell@us.ibm.com`