



# CHAPTER 1

## GETTING STARTED

## OBJECTIVES

*After completing “Getting Started,” you will be able to:*

- Describe the role of the eXtensible Stylesheet Language in producing user-readable presentations from XML data.
- Describe the architecture of XSL as comprising XSLT and XSLFO.
- Implement basic text formatting using XSLFO blocks and properties.

# Formatting XML

---

- XML was first received as “a better HTML.”
- It has flourished primarily in other contexts:
  - **Data-centric**, not presentation-centric
  - **Business-to-business**, not business-to-consumer
- Still, presentation of information to humans is a key part of almost any application.
- Several possible processing paths compete today:
  - **XSLT** alone can provide XML-to-HTML transformation, either server-side or in-browser.
  - **CSS** provides direct styling of XML – primarily in-browser.
  - **XSLFO** offers a means of formatting XML, primarily for print but also to screen, audio, and other media.

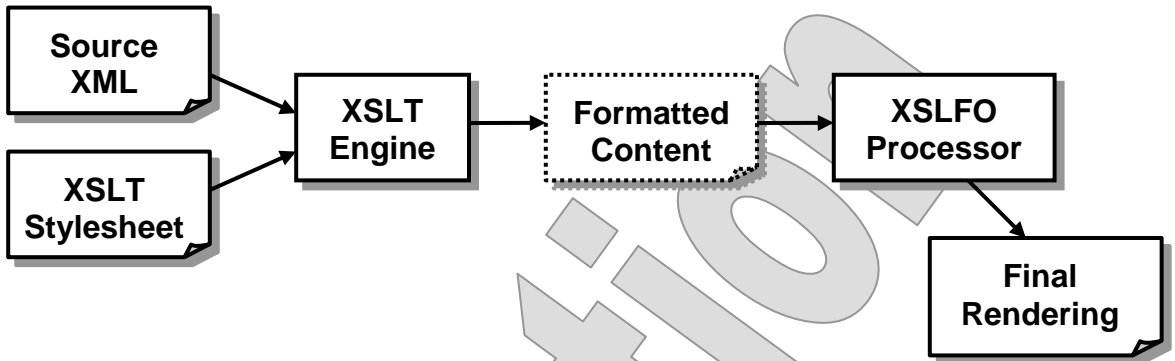
# XSLT and XSLFO

---

- Originally the W3C meant to create a single recommendation called **XSL**.
  - a.k.a. **eXtensible Stylesheet Language**
- It became clear that a two-phase process would be necessary:
  - **Transformation** from source data to formatting objects
  - **Formatting** of formatting objects to final medium
- The software industry embraced **XSLT**, the transformations piece, well before the full XSL specification was broadly adopted.
  - XSLT proved to be useful for a range of applications, including XML-to-HTML for websites as an alternative to CSS or XSL
- **XSLFO** (“formatting objects”) defines the formatting and rendering part of the total process.
- **XSLT and XSLFO work collaboratively.**
  - XSLT is one option for the transformation from source data to the intermediate formatting-object document – not the only one, but the original idea, and still the de-facto standard.
  - XSLFO defines the formatting-object model and requirements for rendering to final media such as PDFs

# Flow of Information

---

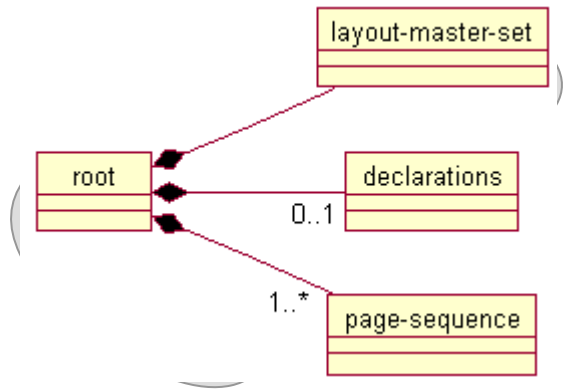


- The complete process of rendering XML information includes two steps:
  - **Tree transformation**, in which an XSLT processor consumes a **source document** (.xml) and a **stylesheet** (.xsl or .xslt) and produces the **result tree** (typically .fo).
  - **Formatting**, in which a **formatter** consumes the result tree and produces output in the desired medium, which might be HTML, PDF, PostScript, WML, etc.
- We'll focus on the formatting stage, assuming a working knowledge of XSLT for the transform stage.

# The Formatting Objects

- XSLFO defines a model of **formatting objects** as XML elements.
  - The FO namespace URI, closely related to the XSLT namespace, is: <http://www.w3.org/1999/XSL/Format>
  - We'll refer to names from this model with no prefix in our text, but use the common prefix **fo:** in example code.
- These formatting objects, or some of them, actually contain final data for presentation.
  - The **separation of content and presentation** logic, which we've loved so well, **ends** with the tree transformation.
  - Thus the result tree is almost always generated, rarely hand-authored; in fact it may never exist as a persistent document.
- From a root element `<root>`, the result tree is broken out into three main sections:

- Layout templates, via the `<layout-master-set>`
- Optional `<declarations>`
- Actual formatted content that uses page masters from the layout section, in one or more `<page-sequence>`s



# XSLFO Tools and Setup

---

- A formatting objects processor will be required for course exercises.
- Our standard setup uses **Apache FOP**, which implements most, though not all, of the 1.0 specification.
- FOP should be set up on your systems and the directory in which **fop.exe** lives should be on your executable path.
  - Your instructor will know the location of the tool.
- Use FOP as follows:
  - Inputs can be an XML source and an XSLT stylesheet or an intermediate FO document.
  - Outputs can be the FO document, a PDF, or a preview window based on Java AWT.

```
fop -xml Source.xml -xsl Transform.xsl  
-pdf Result.pdf
```

```
fop -xml Source.xml -xsl Transform.xsl Result.fo
```

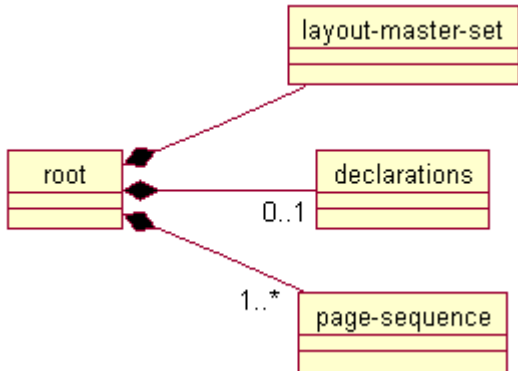
```
fop -xml Source.xml -xsl Transform.xsl -awt
```

```
fop -fo Result.fo -pdf Result.pdf
```

- Note that when reviewing PDFs produced by FOP, Acrobat Reader will lock the PDF; you must close Acrobat to run another FOP transformation.

- In **Examples\LostCat\Step1**, there is a hand-written formatting objects file **Missing.fo** that illustrates simple paragraph-style text formatting.
- The overall structure of the page is shown below, with the details elided:

```
<fo:root>  
  <fo:layout-master-set>  
    ...  
  </fo:layout-master-set>  
  <fo:page-sequence>  
    <fo:flow>  
      ...  
    </fo:flow>  
  </fo:page-sequence>  
</fo:root>
```



- Review the contents of the page sequence, which are the actual objects to be rendered to the printed page:
  - The page sequence includes a single `<flow>`, which is an ordered collection of **blocks** that should be laid out one after the other in the vertical direction.

```
<fo:page-sequence
  master-reference="Main"
>
```

```
<fo:flow flow-name="xsl-region-body">
```

- Each title or paragraph is a `<block>`, which is an element that occupies its own vertical space – i.e. no two blocks will share a vertical position on the page.
- Attributes of the `<block>` element, which XSLFO calls **properties**, define various aspects of presentation.

```
<fo:block
  font-size="48pt"
  font-weight="bold"
  text-align="center"
  space-after="24pt"
>MISSING</fo:block>
```

```
<fo:block
  font-size="14pt"
  space-after="12pt"
>Help! We've lost our cat! His ...</fo:block>
```

...

- Preview the formatted page using FOP:

```
fop -fo Missing.fo -awt
```



# MISSING

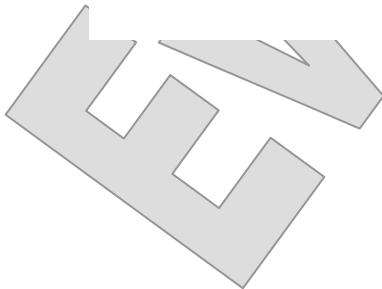
Help! We've lost our cat! His name is **Elvis**, and he's a tiger-stripe tomcat with greenish eyes; muscular-looking, about 12 pounds, with a low, scratchy voice. When lost he was wearing a collar and tags, including his name and our phone number.

Elvis disappeared three days ago from 209 Magnolia Drive. He is friendly, but may be shy of strangers while out and about. If you see him, please let us know.

**787-2303**

or

**hastings@isp.net**



# XSLFO Properties

---

- XSLFO provides a rich set of **properties** that can be applied to any of the formatting objects.
- These are expressed as attributes on the XML elements that represent the formatting objects.
- Thus there is a common set of attributes that is technically legal on all objects.
  - In fact this is the reason given for the complete absence of a normative XML Schema for XSLFO.
- However, not every property is truly applicable to every object.
- We will highlight properties that are most useful and/or most common on various objects, but will not cover every possible combination of element and attribute.
- Those familiar with CSS will find most XSLFO property names and meanings familiar; this is by design.

# Property Inheritance

---

- Many, though not all, properties can be defined on a parent element and will then apply to the parent and to all of its children and their children – that is, to all **descendants**.
  - Terms: parent:ancestor::child:descendant
- This greatly reduces the work involved in defining, say, a common font or color for all the text in a table or span of paragraphs.
- Some properties are not inheritable, often because in common usage scenarios they would more often have to be switched back by an inheriting child object.

# Shorthand vs. Short Form

---

- Some groups of properties can be defined using a single **shorthand** for convenience.
  - For instance **margin** defines all four of **margin-top**, **margin-left**, **margin-right**, and **margin-bottom** to be the same value.
- **Compound properties do the same thing by different semantics.**
  - For instance **space-before** is actually a group of compound properties **space-before.minimum**, **space-before.optimum**, and **space-before.maximum**.
  - A similar convenience mechanism for compound properties is called the **short form**, by which only the root token of a compound name is specified to define values for the whole group.
  - That is, **space-before** can be set to define the same value for minimum, optimum and maximum.
- **In case things weren't complicated enough, short-form properties are **never** inherited.**
  - But their component properties might be!

# Conformance

---

- The XSLFO specification defines three levels of possible conformance:
  - **Basic**, which includes the entire formatting object structure but only some of the properties
  - **Extended**, which expands the set of available properties but not shorthands
  - **Complete**, which encompasses the entire spec
- FOP is only a basic-level processor.
  - It does implement many of the extended properties, but not all of them.
- This is true of many FO processors today.
  - In fact, implementation of most shorthands (complete level) is more common than some of the extended features.
- XSLFO is still quite young in the marketplace.
  - Different conformance levels allow processors to be brought out for various purposes, not all of which require complex constraint resolution and other features involved in the complete specification.
- It is important to understand what level of conformance you enjoy with your current processor, and on which your XSLFO code relies.
  - To use anything beyond basic property sets is to risk a loss of portability to other conformant processors.

In this lab you will enhance the lost-cat poster by adding formatting properties in several places.

Detailed instructions are contained in the Lab 1 write-up at the end of the chapter.

Suggested time: 15 minutes.

Evaluation  
Only

## SUMMARY

- XSLFO defines a model from which information can be precisely rendered to presentation media.
- Information is typically placed in this form as an intermediate stage between source information (usually itself in an XML form) and final rendering.
- XSLT is the recommended means for creating the FO tree, but others are possible.
- XSLFO uses a weakly-typed model of properties, many of which are applicable to broad ranges of components.
  - There is no normative XML Schema for XSLFO.
  - There are clear rules about what applies to what, and some processors apply application or transformation logic to an FO document as a way of validating it.