Published on XML.com http://www.xml.com/pub/a/2002/03/20/xsl-fo.html
See this if you're having trouble printing code examples

What Is XSL-FO

By G. Ken Holman
March 20, 2002

Editor's Note

I'm pleased to be able to present extended excerpts from Ken Holman's well known and respected training materials on XSL-FO. For reasons of internal consistency, we have kept the section numbering from the original material -- this means that some sections will not be numbered contiguously.

I wish to extend my thanks to Ken for permitting XML.com to publish this excellent introduction to the W3C's XSL Formatting Objects.

Edd Dumbill, Editor, XML.com

Prologue

We often take the printed form of information for granted, yet how many of us are satisfied with the print-screen functionality from a web browser? How many times have you printed a lengthy web document and found the paginated result to be as easily navigated as the electronic original?

Navigating a paginated document is very different than navigating a web page, and browser-based navigation mechanisms understandably will not work on printed output. How would we follow a hyperlink when the visible clickable content hides the underlying hyperlink target address?

When we want to produce a paginated presentation of our XML information, we necessarily must offer a different set of navigation tools to the consumers of our documents. These navigational aids have been honed since bound books have been used: headers, footers, page numbers and page number citations are some of the characteristics of printed pages we use to find our way around a collection of fixed-sized folios of information.

This collection of fixed-sized folios may, indeed, have different geometries of page sizes and margin widths used therein, but each page once rendered is fixed in its particular geometry. Layout and typesetting controls give us the power to express our information on pages in a visually pleasing and perhaps meaningful set of conventions conveying information in the presentation itself.

Many aspects of layout are, indeed, applicable on electronic displays and Recommendations such as Cascading Stylesheets (CSS) have defined presentation semantics in areas such as font, margin, and color properties. Paginating marked-up information is not something new, in that the Document Style Semantics and Specification Language (DSSSL) is an international standard for use originally with SGML documents, though it also works unchanged with XML documents.

Accepting that HTML and CSS are suitable and sufficient for browser-oriented rendering of information, the W3C set out to define a collection of pagination semantics for print-oriented rendering. These pagination semantics are equally suitable for an electronic display of fixed-size folios of information, such as page-turner browsers and Portable Document Format (PDF) readers.

The Extensible Stylesheet Language (XSL), also known colloquially in our community as the Extensible Stylesheet Language Formatting Objects (XSLFO), combines the heritage of CSS and DSSSL in a well-thought-out and robust specification of formatting semantics for paginating information.

The Recommendation itself is a rigorous, lengthy, and involved technical specification of the processes and operations engaged by a formatting engine to effect consistent paginated results compared to other formatting engines acting on the same inputs. Well-written for its intended purpose, the document remains out of reach for many people who just want to write stylesheets and print their information.

In its ever-growing collection of training material, Crane Softwrights Ltd. has published Practical Formatting Using XSLFO covering every formatting object of XSLFO and their properties, according to the final XSL 1.0 Recommendation of October 15, 2001. The first two chapters of this book have been rewritten in prose and are made available here as an introduction to the technology and its use. This material assumes no prior knowledge of XSLFO and guides the reader through background, context, structure, concepts, introductory terminology, and a short introduction of each of the formatting objects.

Note that neither the Recommendation itself, nor Crane's training material, attempt to teach facets of typography and attractive or appropriate layout style, only the semantics of formatting, the implementation of those semantics, and the nuances of control available to the stylesheet writer and implemented by the stylesheet formatting tool. XSLFO is a very powerful language with which we can possibly create very ugly or very beautiful pages from our XML-based information.

Table of Contents

This is a prose version of an excerpt from an edited version of the book "Practical Formatting Using XSLFO" (First Edition ISBN 1-894049-07-1 at the time of this writing) published by Crane Softwrights Ltd., written by G. Ken Holman.

1. The context of XSLFO

This chapter reviews the roles of the following Recommendations in the XML family and an International Standard in the SGML family, and overviews contexts in which XSLFO is used.

Extensible Markup Language (XML)

We use XML to express information hierarchically in a sequence of characters according to a vocabulary of element types and their attributes. Using various Recommendations and other industry standards, we can formally describe the makeup and constraints of this vocabulary in different ways to validate the content against our desired document model.

Cascading Stylesheets (CSS)

Initially created for the rendering HTML documents in browsers, CSS formatting properties can ornament the document tree described by a sequence of markup following that specific SGML vocabulary. CSS was later revised to describe the ornamentation of XML documents so that CSS-aware browsers can render the information found in a decorated document tree described by any XML vocabulary. Browsers recognizing these properties can render the contents of the tree according to the semantics of the formatting model governing the property interpretation.

Document Style Semantics and Specification Language (DSSSL)

The International Organization for Standardization (ISO) standardized a collection of style semantics in DSSSL for formatting paginated information. DSSSL also includes a specification language for the transformation of Standard Generalized Markup Language (SGML) documents of any vocabulary, and implementations have since been modified to support the styling of XML documents of any vocabulary. This introduced the concept of a flow object tree comprising objects and properties reflecting the internationalized semantics of paginated output.

Extensible Stylesheet Language Family (XSLT/XSL/XSLFO)

Two vocabularies specified in separate W3C Recommendations provide for the two distinct styling processes of transforming and rendering XML instances.

The Extensible Stylesheet Language Transformations (XSLT) is a templating markup language used to express how a processor creates a transformed result from an instance of XML information.

The Extensible Stylesheet Language Formatting Objects (XSLFO) is a pagination markup language describing a rendering vocabulary capturing the semantics of formatting information for paginated presentation. Formally named Extensible Stylesheet Language (XSL), this Recommendation normatively incorporates the entire XSLT Recommendation by reference and, historically, used to be defined together in a single W3C draft Recommendation.

While XSLT is designed primarily for the kinds of transformation required for using XSL, it can also be used for arbitrary transformation requirements.

1.1.4 Styling structured information

Styling is transforming and formatting information

Styling is the rendering of information into a form suitable for consumption by a target audience. Because the audience can change for a given set of information, we often need to apply different styling for that information to obtain dissimilar renderings to meet the needs of each audience. Perhaps some information needs to be rearranged to make more sense for the reader. Perhaps some information needs to be highlighted differently to bring focus to key content.

It is important when we think about styling information to remember that two distinct processes are involved, not just one. First, we must transform the information from the organization used when it was created into the organization needed for consumption. Second, when rendering we must express the aspects of the appearance of the reorganized information, whatever the target medium.

Consider the flow of information as a streaming process where information is created upstream and processed or consumed downstream. Upstream, in the early stages, we should be expressing the information abstractly, thus preventing any early binding of concrete or final-form concepts. Midstream, or even downstream, we can exploit the information as long as it remains flexible and abstract. Late binding of the information to a final form can be based on the target use of the final product; by delaying this binding until late in the process, we preserve the original information for exploitation for other purposes along the way.

It is a common but misdirected practice to model information based on how you plan to use it downstream. It does not matter if your target is a presentation-oriented structure, for example, or a structure that is appropriate for another markup-based system. Modeling practice should focus on both the business reasons and inherent relationships existing in the semantics behind the information being described (as such the vocabularies are then content-oriented). For example, emphasized text is often confused with a particular format in which it is rendered. Where we could model information using a `<b>` element type for eventual rendering in a bold face, we would be better off modeling the information using an `<emph>` element type. In this way we capture the reason for marking up information (that it is emphasized from surrounding information), and we do not lock

the downstream targets into only using a bold face for rendering.

Many times the midstream or downstream processes need only rearrange, re-label or synthesize the information for a target purpose and never apply any semantics of style for rendering purposes. Transformation tasks stand alone in such cases, meeting the processing needs without introducing rendering issues.

One caveat regarding modeling content-oriented information is that there are applications where the content-orientation is, indeed, presentation-oriented. Consider book publishing where the abstract content is based on presentational semantics. This is meaningful because there is no abstraction beyond the appearance or presentation of the content.

Consider the customer information in Example 1-1. A web user agent doesn't know how to render an element named `<customer>`. The HTML vocabulary used to render the customer information could be as follows:

```
01  <p>From: <i>(Customer Reference) <b>cust123</b></i>
02  </p>
```

Example 1-2: HTML rendering semantics markup for example

The rendering result would then be as in Figure 1.1, with the rendering user agent interpreting the markup for italics and boldface presentation semantics:

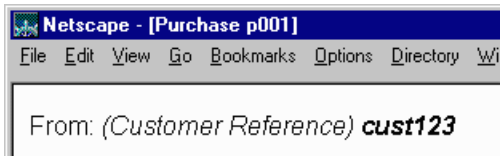

Figure 1.1: HTML rendering for example

The figure illustrates these two distinct styling steps: *transforming* the instance of the XML vocabulary into a new instance according to a vocabulary of rendering semantics; and *formatting* the instance of the rendering vocabulary in the user agent.

Two W3C Recommendations

To meet these two distinct processes in a detached (yet related) fashion, the W3C Working Group responsible for the Extensible Stylesheet Language (XSL) split the original drafts of their work into two separate Recommendations: one for transforming information and the other for paginating information.

The XSL Transformations (XSLT) Recommendation describes a vocabulary recognized by an XSLT processor to transform information from an organization in the source file into a different organization suitable for continued downstream processing.

The Extensible Stylesheet Language (XSL) Recommendation describes a vocabulary (often called XSLFO for "Formatting/flow Objects") reflecting the semantics of paginating a stream of information into individual pages. The XSLFO Recommendation normatively includes XSLT and historically both Recommendations were expressed in a single document.

Both XSLT and XSLFO are endorsed by members of WSSSL, an association of researchers and developers passionate about the application of markup technologies in today's information technology infrastructure.

This is a prose version of an excerpt from an edited version of the book "Practical Formatting Using XSLFO" (First Edition ISBN 1-894049-07-1 at the time of this writing) published by Crane Softwrights Ltd., written by G. Ken Holman.

1.1.6 Extensible Stylesheet Language (XSL/XSLFO)

XSL (or XSLFO) describes formatting and flow semantics for paginated presentation that can be expressed using an XML vocabulary of elements and attributes:

- `http://www.w3.org/TR/xsl`

Paginated formatting and flow semantics vocabulary

This hierarchical vocabulary captures formatting semantics for rendering textual and graphic information in different media in a paginated form. A rendering agent is responsible for interpreting an instance of the vocabulary for a given medium to reify a final result.

This is no different in concept and architecture than using HTML and Cascading Stylesheets (CSS) as a hierarchical vocabulary and formatting properties for rendering a set of information in a web browser. Such user agents are not pagination-oriented and effectively have an infinite page length and variable page width.

Indeed, the printed paged output from a browser of an HTML page is often less than satisfactory. Paginated information includes navigation tools such as page numbers, page number citations, headers, footers, etc. to give the reader methods of finding information or finding their location in a printed document.

In essence, when doing any kind of presentation, we are transforming our XML documents into a final display form by transforming instances of our XML vocabularies into instances of a particular rendering vocabulary that expresses the formatting semantics of our desired result. Our choice of vocabulary must be able to express the nature of the formatting we want accomplished. We can choose to transform our information into a combination of HTML and CSS for web browsers and can choose an alternate transformation of XSLFO for paginated display (be that paginated to a screen, to paper, or perhaps even aurally using sound).

In this way XSLFO can be considered a pagination markup language.

Target of transformation

When using the XSLFO vocabulary as the rendering language, the objective for a stylesheet writer is to convert an XML instance of some arbitrary XML vocabulary into an instance of the formatting semantics vocabulary. This formatting instance is the information rearranged into an expression of the intent of the paginated result as a collection of layout constructs populated with the content to be laid out on the rendered pages.

This result of transformation cannot contain any user-defined vocabulary constructs (e.g.: an "address", "customer identifier", or "purchase order number" construct) because the rendering agent would not know what to do with constructs labeled with these foreign, unknown identifiers.

Consider again the two examples: HTML for rendering on a single page infinite length in a web browser window, and XSLFO for rendering on multiple separated pages on a screen, on paper or audibly. In both cases, the rendering agents only understand the vocabulary expressing their respective formatting semantics and wouldn't know what to do with alien element types defined by the user.

Just as with HTML, a stylesheet writer utilizing XSLFO for pagination must transform each and every user construct into a rendering construct to direct the rendering agent to produce the desired result. By learning and understanding the semantics behind the constructs of XSLFO, the stylesheet writer can create an instance of the formatting vocabulary expressing the desired layout of the final result (e.g. area geometry, spacing, font metrics, etc.), with each piece of information in the result coming from either the source data or the stylesheet itself.

Consider once more the customer information in Example 1-1. An XSLFO rendering agent doesn't know how to render a marked up construct named `<customer>`. The XSLFO vocabulary used to render the customer information could be as follows:

```
01  <fo:block space-before.optimum="20pt" font-size="20pt">From:
02  <fo:inline font-style="italic">(Customer Reference)
03  <fo:inline font-weight="bold">cust123</fo:inline>
04  </fo:inline>

05  </fo:block>
```

Example 1-7: XSLFO rendering semantics markup for example

The rendering result when using the Portable Document Format (PDF) would then be as in Figure 1.2, with an intermediate PDF generation step interpreting the XSLFO markup for italics and boldface presentation semantics.

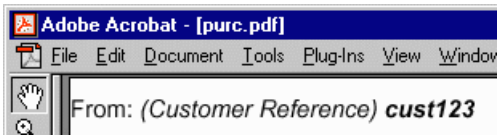

Figure 1.2: XSLFO rendering for example

The figure again illustrates the two distinctive styling steps: *transforming* the instance of the XML vocabulary into a new instance according to a vocabulary of rendering semantics; and *formatting* the instance of the rendering vocabulary in the user agent.

The formatting semantics of the XSLFO vocabulary are described for both visual and aural targets, so we can use one set of constructs regardless of the rendering medium. It is the rendering agent's responsibility to interpret these constructs accordingly. In this way, the XSLFO semantics can be interpreted for print, display, audio, or other presentations. There are, indeed, some specialized semantics we can use to influence rendering on particular media, though these are just icing on the cake. Dynamic behaviors can be specified for a highly interactive electronic display that would not function at all, obviously, in the paper form.

1.1.7 Transforming and rendering XML information using XSLFO

When the result tree in an XSLT process is specified to utilize the XSLFO pagination vocabulary, the normative behavior of an XSLFO processor incorporating an XSLT processor is to interpret the result tree. This interpretation reifies the semantics expressed in the constructs of the result tree to some medium, for example pixels on a screen, dots on paper, sound through a synthesis device (see Figure 1.3).

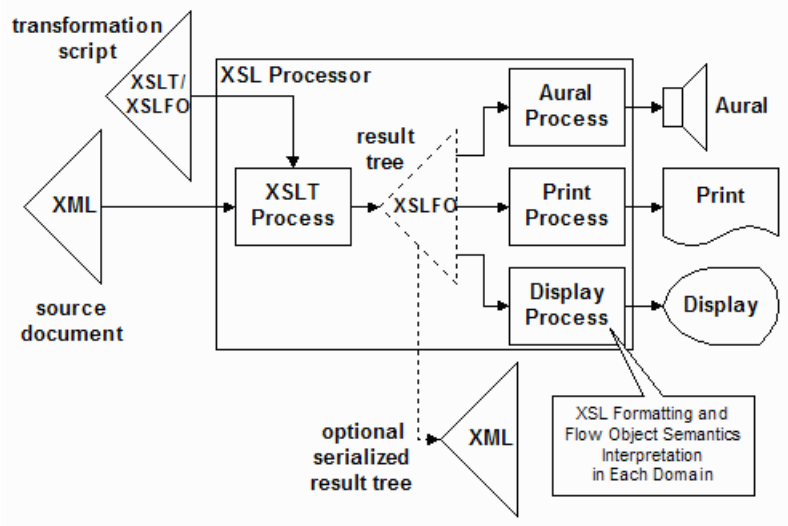

Figure 1.3: Transformation from XML to XSL Formatting Semantics

The stylesheets used in this scenario contain the transformation vocabulary and any custom extensions, as well as the desired result XSLFO formatting vocabulary and any foreign object vocabularies. There are no other element types from our XML vocabularies are in the result. If there were, rendering processors would not inherently know what to do with an element of type `custnbr` representing a customer number; it is the stylesheet writer's responsibility to transform the information into information recognized by the rendering agent.

There is no obligation for the rendering processor to serialize the result tree created during transformation. The feature of serializing the result tree to XML markup is, however, quite useful as a diagnostic tool, revealing to us what we really asked to be rendered instead of what we thought we were asking to be rendered when we saw incorrect results. There may also be performance considerations of taking the reified result tree in XML markup and rendering it in other media without incurring the overhead of performing the transformation repeatedly.

1.1.8 Interpreting XSLFO instances directly

The XSLFO and foreign object vocabularies can also be used in a standalone XML instance, perhaps as the result of an XSLT transformation using an outboard XSLT processor. The XSLT processor serializes a physical entity from the transformation result tree, and that XML file of XSLFO vocabulary being interpreted by a standalone XSLFO processor.

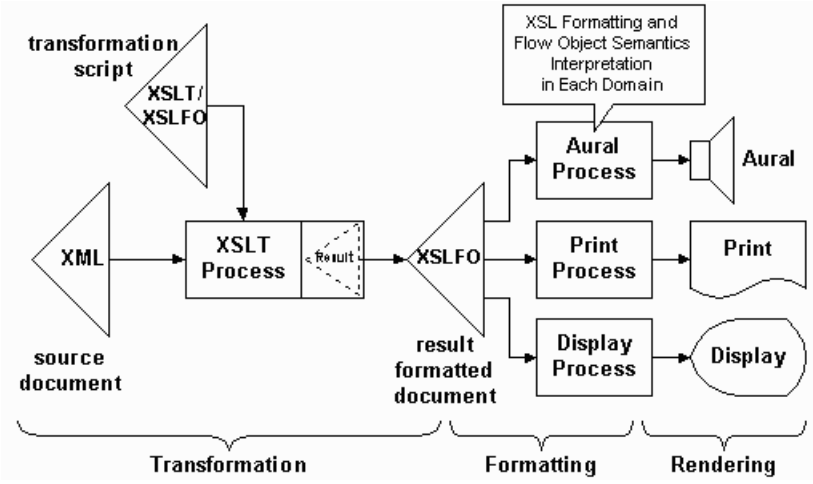

Figure 1.4: Creating standalone XML instances of XSL vocabulary

This diagram delineates three distinct phases of the process that are also phases when the XSLT and XSLFO processors are combined into a single application. The transformation phase creates the XSLFO expressing our intent for formatting the source XML. The XSLFO processor first interprets our intent into the information that is to be rendered on the device, then effects the rendering to reify the result.

1.1.9 Generating FO instances

XSLFO need not be generated by XSLT in order to be useful. Consider that when we learned HTML as the rendering vocabulary for a web user agent, we either coded it by hand or we wrote applications that generated the HTML from our information. This information may have come from some source, such as a database.

Learning XSLT, we can express our information in XML and then either transform the XML into HTML to send to the user agent, or send the XML directly to an XSLT process in the user agent.

The typical generation of XSLFO would be from our XML using an XSLT stylesheet, though this need not be the case at all. We may have situations where our applications need to express information in a paginated form, and these applications could generate instances of the XSLFO vocabulary directly to be interpreted for the output medium.

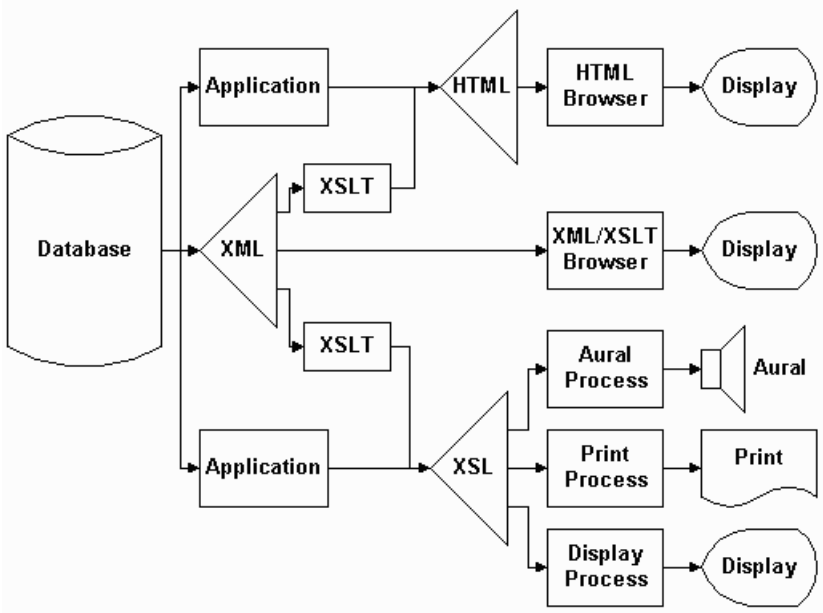

Figure 1.5: Generating XML instances of XSL vocabulary

We need to remember that XSLFO is just another vocabulary, able to be expressed as an XML instance, requiring an application to interpret our intent for formatting in order to effect the result. This is no different than the use of the HTML vocabulary for a web browser.

The sole requirement is that the namespace of the vocabulary in the instance be "`http://www.w3.org/1999/XSL/Format`" for the labeled information in the instance to be recognized as expressing the semantics described by the XSLFO Recommendation.

Note 3:	The default namespace may be used for the XSLFO vocabulary, just as is true with any vocabulary. Personally, I don't use the popular "`fo:`" prefix in my stylesheets, as it is my habit to use the default namespace and not prefix my XSLFO names in any way. This practice reinforces for me that this is just as simple as HTML, where I don't use any namespace at all in my own stylesheets. There are processors that interpret standalone XSLFO instances interactively on the screen in a GUI environment. To learn much of the nuances of XSLFO, I often hand-author XSLFO instances experimenting with various objects and properties in elements and attributes, tweaking values repeatedly and examining the results interactively with the formatting tool. Having hand-authored HTML, using the default namespace for XSLFO is very natural and saves on the amount of typing as well.

This is a prose version of an excerpt from an edited version of the book "Practical Formatting Using XSLFO" (First Edition ISBN 1-894049-07-1 at the time of this writing) published by Crane Softwrights Ltd., written by G. Ken Holman.

1.2 Examples

1.2.1 Hello world example

Consider a simple, but complete, XSLFO instance `hellofo.fo` for an A4 page report:

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <root xmlns="http://www.w3.org/1999/XSL/Format"
03   font-size="16pt">
04   <layout-master-set>

05   <simple-page-master
06   margin-right="15mm" margin-left="15mm"
07   margin-bottom="15mm" margin-top="15mm"
08   page-width="210mm" page-height="297mm"
09   master-name="bookpage">
10   <region-body region-name="bookpage-body"
```

```
11   margin-bottom="5mm" margin-top="5mm" />
12   </simple-page-master>
13   </layout-master-set>
14   <page-sequence master-reference="bookpage">
15   <title>Hello world example</title>

16   <flow flow-name="bookpage-body">
17   <block>Hello XSLFO!</block>
18   </flow>
19   </page-sequence>

20  </root>
```

Example 1-8: A simple example

We can see the definition on line 2 of the default namespace being the XSLFO namespace, thus un-prefixed element names refer to element types in the XSLFO vocabulary. There are no prefixed element types used by any of the elements, thus the entire content is written in XSLFO.

The document model for XSLFO dictates the page geometries be summarized in `<layout-master-set>` on lines 4 through 13, followed by the content to be paginated in a sequence of pages in `<page-sequence>` on lines 14 through 19. The instance conforms to this and conveys our formatting intent to the formatter. The formatter needs to know the geometry of the pages being created and the content belonging on those pages.

Think of the parallel where we learned the document model for HTML requires the metadata in the `<head>` element and the displayable content in the `<body>` element. Both elements are required in the document model, the first to contain the mandatory title of the page and the second to contain the rendered information.

However we learned the vocabulary for HTML, when we create a page we know where the required components belong in the document. The same is true for XSLFO, in that we learn what information is required where and we express what we need in the constructs the formatter expects.

In this simple example the dimensions of A4 paper are given in a portrait orientation on line 8. Margins are specified on lines 6 and 7 to constrain the main body of the page within the page boundaries. That body region itself, described on lines 10 and 11, has margins to constrain its content, and is named so that it can be referenced from within a sequence of pages.

The sequence of pages in this example refers to the only geometry available and specifies on line 16 that the flow of paginated content is targeted to the body region on each page. The sequence is also titled on line 15, which is used by rendering agents choosing to expose the title outside the canvas for the content.

Consider two conforming XSLFO processors to process the simple `hellofo.fo` example, one interactively through a GUI window interface, and the other producing a final-form representation of the page:

- Antenna House XSL Formatter (an interactive XSLFO rendering tool)
  - http://www.AntennaHouse.com
- Adobe Acrobat (a Portable Document Format (PDF) display tool)
  - PDF created by RenderX (a batch XSLFO rendering tool)
    - http://www.RenderX.com

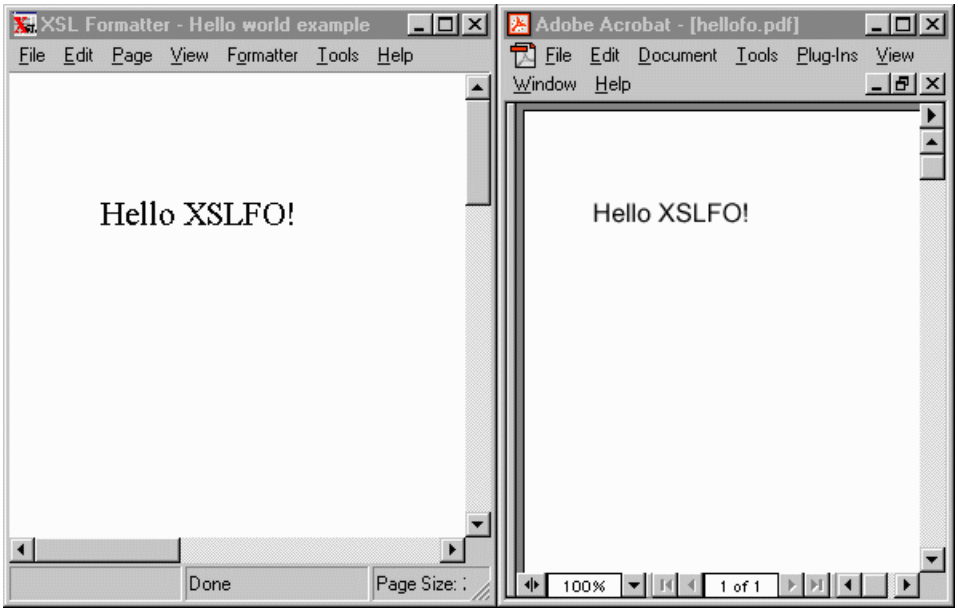

Figure 1.6: A simple XSLFO instance example

Note how the two renderings are not identical. If the XSLFO instance is insufficient in describing the entire intent of the formatting, the rendering

may engage certain property values of its own choosing. Page fidelity is not guaranteed if the instance does not express the entire intent of formatting. Even within the expressiveness of the XSLFO semantics, there are some decisions still left up to the formatting tool.

This is not different than two web browsers with different user settings for the displayed font. A simple web page that does not use CSS stylesheets for font settings relies on the browser's tool options for the displayed font choice. The intent of the web page may be to render "a paragraph", but if two users have different tool option defaults for the font choice, there is no fidelity in the web page between the two renditions if the formatting intent is absent.

1.2.2 Training material example

Consider an excerpt of a more complex use of formatting objects to produce a page from an early draft of the instructor-led derivative of this training material:

```
 <flow flow-name="pages-body"><table>
  <table-column column-width="( 210mm - 2 * 15mm ) - 2in"/>
  <table-column column-width="1in"/>
  <table-column column-width="1in"/>

  <table-body><table-row><table-cell><block text-align="start">
  <block font-size="19pt">Training material example</block>
  <block font-size="10pt" space-before.optimum="10pt">Module

 1 - The context of XSLFO</block>
  <block font-size="10pt">Lesson 2 - Examples</block></block>
  </table-cell>
  <table-cell><block text-align="end">

  <external-graphic src="..\whitesml.bmp"/></block></table-cell>
  <table-cell><block text-align="start">
  <external-graphic src="..\cranesml.bmp"/></block></table-cell>

  </table-row></table-body></table>
  <block line-height="3px"><leader leader-pattern="rule"
  leader-length.optimum="100%" rule-thickness="1px"/></block>
  <block space-before.optimum="6pt" font-size="14pt">

 This page's material as an instructor-led handout:</block>
  <list-block provisional-distance-between-starts=".43in"
  provisional-label-separation=".1in" space-before.optimum="6pt">
  <list-item relative-align="baseline">
  <list-item-label text-align="end" end-indent="label-end()">

  <block>-</block></list-item-label>
  <list-item-body start-indent="body-start()">
  <block font-size="14pt">excerpts of formatting objects
 created through the use of an XSLT stylesheet</block>

  </list-item-body></list-item></list-block>
  <block space-before.optimum="12pt div 2" font-family="Courier"
  linefeed-treatment="preserve" white-space-collapse="false"
  white-space-treatment="preserve" font-size="12pt"><inline
 font-size="inherited-property-value(font-size) div 2">01 </inline

 >&lt;flow flow-name="pages-body"&gt;&lt;table&gt;
 <inline font-size="inherited-property-value(font-size) div 2"
 >02 </inline> &lt;table-column column-width...
```

Example 1-9: Formatting objects (excerpt) for a page of handout material

The nesting of the hierarchy of the formatting objects in the example page:

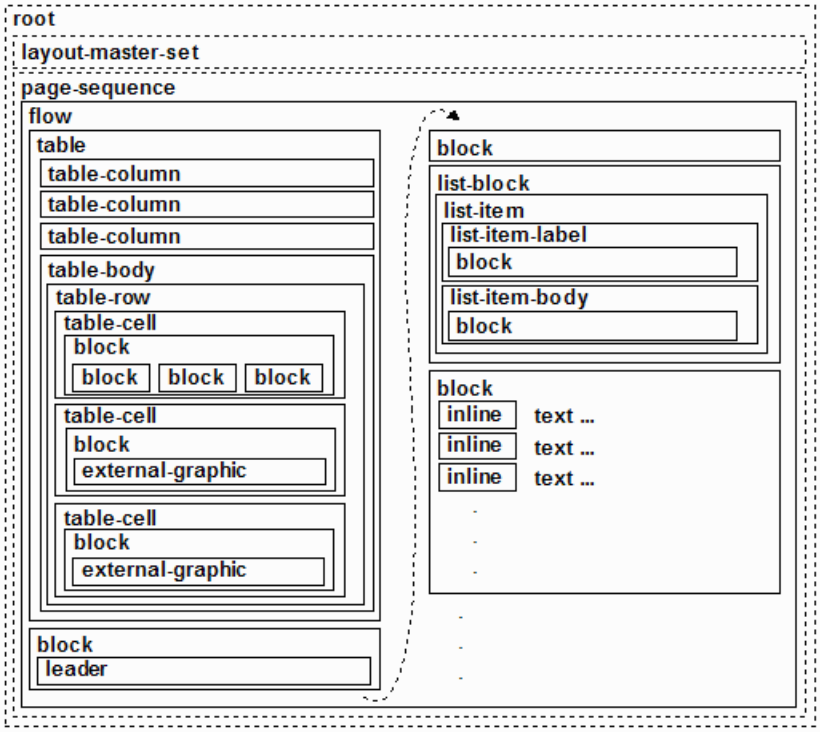

Figure 1.7: The nesting of XSLFO constructs in the example

The page rendered in an interactive XSLFO rendering tool:

Figure 1.8: A page of handouts rendered in XSLFO

The information above the horizontal rule is rendered using a borderless table. Lines 1 through 15 describe the three columns of information: the page title and context, a placebo white box in place of the branding logo for the licensee of the training material, and the Crane registered trademark. The table cell with the page information contains text in different point sizes on lines 6 through 9.

Note how attribute value specified on line 2 is an expression, not a hard value. There is an expression language in XSLFO that is a superset of the expression language of XSLT. This can make an XSLT stylesheet easier to write by having it convey property values in a piecemeal fashion in an expression to be evaluated, rather than trying to calculate the resulting value in XSLT.

The horizontal rule below the title information needs to be block-oriented in that it needs to break the flow of information and be separate from the surrounding information. To achieve this effect with the inline-oriented leader construct, note on lines 16 and 17 how the leader is placed inside of a block. Note also how the line height of the block is adjusted in order to get the desired spacing around the leader.

The block on lines 18 and 19 lay out a simple paragraph.

Lines 20 through 28 lay out a list construct, where the labels and bodies of list items are synchronized and layout out adjacent to each other in the flow of information. This side-by-side effect cannot be achieved with simple paragraphs, and could be achieved to some extent with borderless tables, but the use of the list objects gives fine control over the nuances of the layout of a list construct.

The list block itself has properties on lines 20 and 21 governing all members of the list, including the provisional distance between the start edges of the list item label and the list item body, and the provisional label separation. These provisional values are very powerful constructs in XSLFO. They allow us to specify contingent behavior for the XSLFO processor to accommodate the varying lengths of the list item labels of the items of the list.

Note 4:	Remember one of the design goals of XML was that "terseness is of minimal importance" (could they have found a terser way of saying that?). Note how the attribute name specifying the first of these provisional property values is 35 characters long. It is not uncommon to need to use lengthy element and attribute names, and an XSLFO instance always seems to me to be so very verbose to read.

Note on lines 23 and 25 how functions can be used in attribute values. XSLFO defines a library of functions that can be invoked in the expression language. The `label-end()` and `body-start()` functions engage the appropriate use of one of the two provisional list construct properties based on the length of the item's label. This illustrates how XSLFO can offload layout decisions from the XSLT stylesheet, especially when it would be impossible for the XSLT stylesheet to know precise placement details that are effected by font and other issues being tracked by the formatting process.

Line 29 begins the block containing the listing of markup. To ensure a verbatim rendering of edited text, line 30 specifies that all linefeeds in the block of content be preserved, and not to collapse the white-space characters. This disengages the default behavior of treating linefeeds as white-space and collapsing white-space to a single space character, as would be typical for proportional-font paragraphs of prose.

Lines 31 and 32 show an inline sequence of text being formatted differently than the remainder of the text of the block. The desired effect of the line number being half the current font size is specified through the use of the function "`inherited-property-value(font-size)`", though there are two alternate ways of specifying the same relative value: "`50%`" and "`.5em`". Using any of these expressions would both produce the same result.

The escaped markup on lines 33 and 35 may look incorrect, but this is an XML serialization of the XSLFO instance, hence, sensitive markup characters must be escaped in order to be recognized as text, and not as markup. Since this is a page describing markup, the markup being described needs to be distinguished from the markup of the document itself.

This is a prose version of an excerpt from an edited version of the book "Practical Formatting Using XSLFO" (First Edition ISBN 1-894049-07-1 at the time of this writing) published by Crane Softwrights Ltd., written by G. Ken Holman.

2. Basic concepts of XSLFO

Here we review basic aspects of the XSLFO semantics and vocabulary, to gain a better understanding of how the technology works and how to use the specification itself.

Layout-based vs. content-based formatting

Two very different approaches to the formatting of information are contrasted. Layout-based formatting respects the constraints of the target medium, where limitations or capacities of the target may constrain the content or appearance of the information on a page. Content-based formatting respects the quantity and identity of the information, where as much of the target medium is generated to accommodate the information being formatted.

Formatting is different than rendering

The distinction between formatting and rendering is overviewed, comparing how to express what you want formatted vs. expressing how it is to be accomplished on the target device. This contrast is similar to the difference between declarative- and imperative- style programming methods, or the difference between XSLT's "transformation by example" paradigm vs. other algorithmic transformation approaches using programming languages.

Formatting model and vocabulary extends what is currently available for web presentation

The XSLFO semantics and vocabulary address different requirements than infinite-length web user agent windows to meet the needs of imposed arbitrary page boundaries on the presentation of information. These new semantics are inspired by the Document Style Semantics and Specification Language (DSSSL) International Standard ISO/IEC 10179, but in practice diverge from DSSSL towards Cascading Style Sheets 2 (CSS2) for compatibility with web-based processing.

The semantics are classified based on their relationship to similar CSS properties:

- CSS properties by copy (unchanged CSS2 semantics)
- CSS properties with extended values
- CSS properties "broken apart" to a finer granularity
- XSLFO-specific properties

The XSLFO support of multiple writing directions and a reference orientation are important concepts inherited from DSSSL that are not present in CSS2.

Differing processing model concepts are expressed using unambiguous terminology

The XSLFO specification, and this book as well, attempts to be very careful in using precise terminology when what is being referred to has similar concepts that could be confused with other constructs. For example, an XSLFO instance contains elements and their attributes. This is similar to the corresponding formatting object tree with objects and their properties. This is, in turn, similar to the corresponding refined formatting object tree with objects and their area traits. This is, finally, similar to the corresponding area tree with areas and their traits.

XSLFO objects related to basic issues

The XSLFO objects addressing functionality in this area are summarized as:

- `<root>` (6.4.2)
  - the document element of the XSLFO instance
- `<layout-master-set>` (6.4.6)
  - the collection of definitions of page geometries and page selection patterns
- `<page-sequence>` (6.4.5)
  - the definition of information for a sequence of pages with common static information
- `<flow>` (6.4.18)
  - the content that is flowed to as many pages as required

2.1 Basic concepts

2.1.1 Layout-based vs. content-based formatting

Layout-based formatting accommodates the medium being used to present information. The constraints of the medium, or the layout design of the graphic artist, often demands absolute positioning, column location specification, or page number specification. Consider that a magazine may need a particular columnist's article to appear on the right-had edge of page 7, while the three lead stories must be headlined within the first four pages.

This focus on layout places more emphasis on the appearance and location of information than the information itself, dictating the quantity and presentation of the content. Such layout is typically unstructured in both the authoring and the formatting processes, as typified by desktop publishing, journalism, etc.

Content-based formatting accommodates the information being presented with the available medium. The constraints of layout are expressed as rules associated with the information dictating how given information is to be positioned or presented. Consider that a single aircraft maintenance manual cannot have each of its 40,000 to 60,000 pages individually formatted.

This focus on information places more emphasis on the content and rules of layout, rather than on the medium, dictating the automatic layout and presentation of constructs found in the information stream. Such layout is typically highly structured in both the authoring and the formatting processes, as typified by technical publications found in pharmaceutical, aerospace, automotive, or other industries where either vast amounts of information are presented, or the information must be interchanged in a neutral form with other players.

XSLFO is more oriented to content-based formatting than layout-based formatting, though there do exist certain controls for the positioning, cropping, and flowing of information to particular areas of pages in page sequences. XSLFO can express the repetition of page geometries, mechanically accommodating the content as flowed by a transformation of the information into the formatting vocabulary. There is only limited support of the order of specific page sequences, and high-caliber copy-fitting requirements often cannot be met with mechanical unattended transformations.

Note that while XSLFO is not oriented to loose-leaf publishing, that does not prevent it perhaps from being used by a vendor to express the content of pages being maintained in a loose-leaf-based environment. A loose-leaf environment supports "change pages" (a.k.a. "A pages") through a database of effective pages and page contents.

XSLFO has no inherent maintenance facilities for past versions of individual pages, and no inherent support of lists of effective pages. Such facilities could be provided outside the scope of individual page presentations. XSLFO is more oriented to the unrestricted flowing of information to as much of the target medium is required to accommodate the content.

2.1.2 Formatting vs. Rendering

When creating XML we should be designing the structures around our business processes responsible for maintaining the information, instead of the structures used for presentation. An XSLFO instance describes the intent of how that stream of information is to be formatted in the target medium in a paginated fashion. This instance is typically generated by a stylesheet acting on the instance of XML information, rearranging and restructuring the information into the order and presentation desired.

This reordering takes the `#PCDATA` content and attribute content of the instance, repackaging it according to our intent based on our understanding of the semantics of the XSLFO vocabulary. We can reify this reordering as an intermediate file of syntax we can use for diagnostic purposes. We could also take the opportunity to store this reordering as an XML instance for "store and forward" strategies where the formatting takes place later or remotely from where the transformation takes place.

Unlike interactive formatting tools such as desktop publishing products or interactive formatting tools, there is no feedback loop from the XSLFO formatter to the stylesheet creating the XSLFO vocabulary. Therefore, the XSLFO information must be complete with respect to all desired behaviors of the formatter. Any special formatting cases or conditions can be accommodated through contingencies expressed in the XSLFO semantics.

The information arranged in the elements and attributes of our source vocabularies is repackaged into the elements and attributes of the XSLFO formatting vocabulary that express the formatting objects and their properties of the XSLFO semantics. Each formatting object specifies an aspect of either layout, appearance and impartation, or the pagination and flow.

The layout semantics express the intent of locating information positioned on the target medium. Areas of content are specified as located and nested within other areas, in a hierarchical tree of rectangles on each page.

The appearance and impartation semantics express the intent of how the information is to be conveyed to the reader. For visual media, this conveyance includes font, size, color, weight, etc. For aural synthesis, this conveyance includes voice, volume, azimuth, pitch, etc.

The pagination and flow semantics express the intent of how the stream of information being presented is to be parceled within the layout areas. The final pagination is the result of accommodating the amount of flow being presented within the areas that have been defined.

Each of the formatting objects is expressed in an XSLFO instance as an element. It is not necessary to know all formatting objects to get effective formatted results.

An XSLFO formatter is responsible for interpreting the intent to be rendered, as expressed in the XSLFO semantics corresponding to the elements and attributes in the instance created by the stylesheet. Following the Recommendation, the formatter determines what is to be rendered where by interpreting the interaction between formatting objects. How the formatter does this interpretation is defined in excruciating detail in the W3C Recommendation, as this document is written more for implementers than for stylesheet writers.

The properties expressed for each of the objects influence or are included in the structure of the resulting areas. Some of these properties are specifically targeted for certain media and are otherwise ignored by media for which they do not apply.

The Recommendation does not describe in detail the semantics of rendering. Any device-specific rendition is interpreted based on the semantics of the formatting objects that create the trees of areas and the traits found in those areas that are derived from the properties. How the rendering agent actually accomplishes the task of effecting the result of formatting to the target medium is entirely up to the agent, as long as it produces the same result as the intent described by the Recommendation.

The rendering, itself, may be a multiple-step process, producing the final form through a staged expression of rendering through interpretation on a given medium. For example, the rendering may require production of another intermediate formatting language such as TeX. Rendering may directly produce a final-form page description language such as the Portable Document Format (PDF), or the Standard Page Description Language (International Standard ISO/IEC 10180). The physical final form would then be produced from the intermediate form or final page representation. Indeed, there could be many steps to obtain a final result, e.g.: XML to XSLFO to TeX to PDF to paper.

This is a prose version of an excerpt from an edited version of the book "Practical Formatting Using XSLFO" (First Edition ISBN 1-894049-07-1 at the time of this writing) published by Crane Softwrights Ltd., written by G. Ken Holman.

2.2 Processing model

2.2.1 Processing model of formatting

The Recommendation describes the processing model for XSLFO as a series of formal steps in the derivation of the content to be rendered from the instance expressing the intent of formatting, as depicted in Figure 2.1. The Recommendation does not cover the creation of the XSLFO instance, nor the detailed semantics of rendering, but focuses entirely on how to get from the former to the latter.

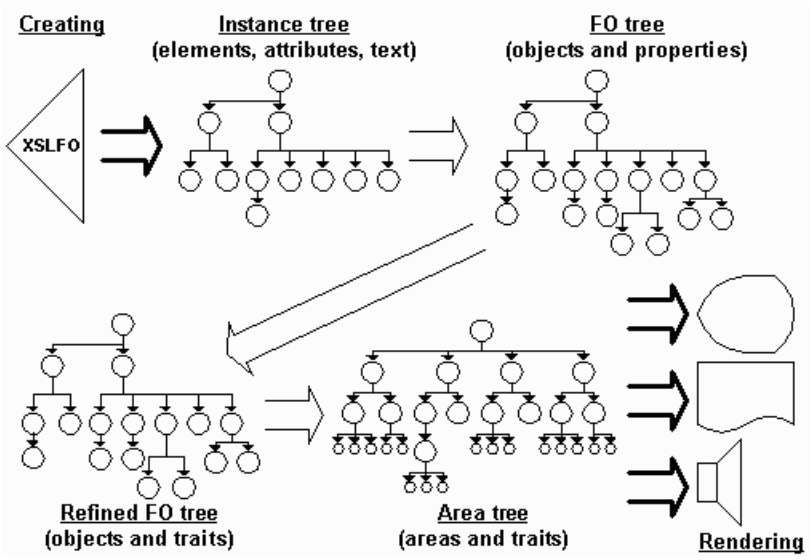

Figure 2.1: XSL processing model flow summary

Note 5:	Although the processing model is described in the Recommendation using constructs and procedural steps following a well-defined sequence, there is no obligation on a vendor that a particular implementation perform the steps as documented. The only obligation on a formatter is that it produce the rendered result *as if it were implemented according to the steps described* in the text. This nuance is important to vendors in that it allows them to implement any algorithm producing equivalent results, without constraining the innovation or flexibility to accomplish the results using any algorithm they wish. One ramification of this flexibility is that none of the intermediate results described in the processing model can be standardized or be required of a particular implementation. Conformance testing would be far simpler if there were a serialization of the abstract result of the interpretation of the formatting intent, without needing to interpret a rendered result as having successfully met the criteria.

First, the instance of elements, attributes, and text becomes a node tree of abstract nodes representing these constructs for processing. It is possible that this node tree is passed directly from the result of transforming some source XML into result XSLFO without instantiating the result as markup characters. However, if the information is presented to a formatter as an instance of markup characters, this must be interpreted into a node tree suitable for the formatter to work with.

This node tree of elements, attributes, and text represents the expression of the intent of what the designer desires in the rendered result. This is called the Instance Tree and includes all of the content, including references to external foreign objects not expressible in XML, that is to appear in the target medium. It is the way the designer expresses the interaction of the documented semantics described in the XSLFO Recommendation.

The Instance Tree is interpreted into the Formatting Object Tree that is comprised entirely of formatting objects and their properties. This requires the (abstract) breaking of text nodes into sequences of character formatting objects, and the creation of properties from attributes.

Note that certain whitespace-only text nodes of the Instance Tree are irrelevant to the formatting process and do not create text nodes in the Formatting Object Tree. Also removed for later access by the formatter or rendering agent are in-stream foreign objects (expressions of the result that are expressed in XML but not in the XSLFO vocabulary, e.g.: a Scalable Vector Graphics (SVG) fragment), and any objects not from the XSLFO namespace that are used in the `<declarations>` formatting object.

The Formatting Object Tree is interpreted into the Refined Formatting Object Tree that is comprised of objects and traits. Properties can specify two kinds of traits: formatting traits (e.g. size and position) or rendering traits (e.g. style and appearance). Some property specifications are shorthand expressions that encompass a number of separate trait specifications and their values.

Computed property expression values are evaluated and the resulting values assigned to the traits. For example a property value of `2em` when the current font size is `20pt` produces a trait value of `40pt`.

Inheritance plays an important role in trait derivation. Some traits are derived from the closest ancestral corresponding property specification. Some traits that are not inherited by default can have their value inherited by the explicit `inherit` property value.

Once all traits that are applicable to all formatting objects are determined, all traits not applicable to each object are removed. At this point the information is comprised of all objects that are used to create areas and each object has all the traits and only the traits that are applicable to them.

The Refined Formatting Object Tree is interpreted into the Area Tree that is comprised of areas and traits. A given object may create areas at different branches of the Area Tree. Most objects produce exactly one area, and some objects do not produce any areas.

Each area has a geometric position, z-layer position, content, background, padding and borders. Areas are nested in the tree within ancestral areas up

to the highest (and largest) area which is the page area.

Page areas are the children of the root node in the Area Tree. Page areas are ordered by their position in the Area Tree, but they are not geometrically related to each other in any way.

The rendering agent effects the impartation of the areas in the Area Tree according to the medium. The Recommendation gives guidelines on the rendering of areas in either visual or aural media. Some missing trait values can be arbitrarily inferred by the rendering agent, such as font or volume. This allowance leads to differing renderings by different tools when an XSLFO instance does not express the missing trait values.

The Recommendation document is written to direct a formatter implementer in carrying out the requirements of interpreting the formatting intent. Certain traits are boolean values targeted solely to the implementer and reflecting an area's role or relative order to other areas. These traits are not specifiable in the XSLFO instance but are indicated in the Recommendation to make implementation easier.

The rigor of the Recommendation language is necessary in order to ensure proper interpretation of finely-tuned typographical nuances. This makes the Recommendation difficult to read for many people just wanting to write stylesheets. Fortunately, simple things can be done simply once you get around the necessary verbosity of the Recommendation document.

Where To Go From Here?

The work on XSL and XSLT continues:

- all XSLT, XPath and XSLFO are now full W3C Recommendations
- long list of future feature considerations already being examined for new releases of the technology
- new products are continually being announced
- feedback is necessary from users like you!
  - use the XSL mail lists to contribute:
    - `http://www.mulberrytech.com/xsl/xsl-list/`
    - `http://groups.yahoo.com/group/XSL-FO`
    - `http://lists.w3.org/Archives/Public/www-xsl-fo/`
  - contact the XSL editors with comments about the specification:
    - `xsl-editors@w3.org`

Further XML.com Coverage

XML.com has published a practical tutorial to developing documents with XSL Formatting Objects, written by J. David Eisenberg: Using XSL Formatting Objects, Part One and Using XSL Formatting Objects, Part Two.

This is a prose version of an excerpt from an edited version of the book "Practical Formatting Using XSLFO" (First Edition ISBN 1-894049-07-1 at the time of this writing) published by Crane Softwrights Ltd., written by G. Ken Holman.
Copyright © Crane Softwrights Ltd.